Нужная информация всегда под рукой!

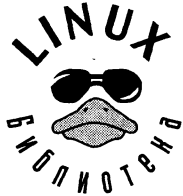
Андрей Шевель

- базовые компоненты Linux
- редакторы текста
- анализ, преобразование и печать текста
- система контроля версий CVS

LINUX ОБРАБОТКА ТЕКСТОВ

Специальный справочник

ПИТЕР



Эта книга входит в универсальную библиотеку Linux.

В составе библиотеки книги самых разнообразных направлений: руководства администратора, книги по программированию, описания различных дистрибутивов, книги для начинающих и книги для профессионалов.

Ищите книги с логотипом нашего проекта на обложках.

ПИТЕР®

Андрей Шевель

LINUX ОБРАБОТКА ТЕКСТОВ

Специальный справочник

Санкт-Петербург
Москва • Харьков • Минск
2001

ПИТЕР®

Андрей Шевель

Linux. Обработка текстов. Специальный справочник

Главный редактор	*В. Усманов*
Заведующий редакцией	*Е. Строганова*
Руководитель проекта	*А. Пасечник*
Научный редактор	*А. Выскубов*
Литературный редактор	*Д. Лещев*
Художник	*Н. Биржаков*
Корректор	*С. Журавина*
Верстка	*Р. Гришанов*

ББК 32.973-018.2

УДК 681.3.066

Шевель А.

Ш37 Linux. Обработка текстов. Специальный справочник. — СПб.: Питер, 2001. — 384 с.: ил.

ISBN 5-272-00039-0

Эта книга призвана ответить на вопросы пользователя: «Какие преимущества даст мне операционная система Linux? Зачем она мне? Что я смогу с ее помощью делать?» Книга посвящена всему, что связано с обработкой текстовой информации. Сюда входит набор, редактирование текстов, оформление, структурирование их, поиск, автоматическое преобразование, верстка, создание хранилищ текстовой информации и баз данных, создание, отправка и прием электронной корреспонденции, описание офисных приложений, словом, все то, что входит в круг обязанностей любого работника современного офиса: от менеджера до секретаря.

ISBN 5-272-00039-0

Лицензия ИД № 01940 от 05.06.2000.

Налоговая льгота – общероссийский классификатор продукции ОК 005-93, том 2; 95 3000 – книги и брошюры.

Подписано к печати 20.12.2000. Формат 84x108$^{1}/_{32}$. Усл. п. л. 20,16. Тираж 5000 экз. Заказ № 976.

ЗАО «**Питер Бук**». 196105, Санкт-Петербург, ул. Благодатная, 67.

Отпечатано с готовых диапозитивов в ГИПК «Лениздат» (типография им. Володарского) Министерства РФ по делам печати, телерадиовещания и средств массовых коммуникаций. 191023, Санкт-Петербург, наб. р. Фонтанки, 59.

Краткое содержание

Содержание

Об этой книге

Автор впервые услышал про Linux в 1992 году, когда он пытался разыскать подходящую бесплатную операционную систему типа UNIX для установки на персональный компьютер. Косвенно о скорости распространения Linux говорит, например, такой факт из личного авторского опыта. Оказавшись как-то в 1995 году в книжном магазине Гамбурга, автор нашел там только 2—3 книги про Linux, а в 1997 году в том же магазине их уже было более 10.

Естественно, что в Гамбурге продавались книги на немецком и английском языках. На русском языке книг о Linux немного, хотя она — одна из популярных операционных систем.

Нужна ли книга о Linux на русском языке?

Зачем вообще нужна какая-то книга, если Интернет полон всякой информации про все, что вам угодно, в том числе и про Linux? Автор должен сказать, что книга и информация в Интернет не заменяют, а лишь дополняют друг друга. А как известно, недостаток одного компонента невозможно компенсировать избытком другого.

Почему же именно Linux? В практическом плане Linux имеет целый ряд преимуществ перед коммерческими операционными системами.

Во-первых, Linux поставляется свободно, и значит, используется множеством людей, которые не имеют средств на покупку коммерческих операционных систем. Обычно такие люди весьма изобретательны и часто хотят продемонстрировать свою изобретательность всем желающим, что приводит к появлению потока свободного программного обеспечения (а также программного обеспечения, распространяемого за символическую плату) для платформы Linux.

Во-вторых, если у вас возникли проблемы с вашей системой, вы можете прибегнуть к помощи большого количества экспертов, которые уже используют Linux. В частности,

воспользовавшись группами новостей по Linux, например, группами иерархии comp.os.linux.* (среди которых — announce, hardware, misc, networking, setup), вы сможете как задать вопрос и в течение 24–48 часов получить на него ответ, так и, если повезет, сразу найти ответ в архиве группы.

В-третьих, Linux представляет собой весьма стабильную многозадачную многопользовательскую операционную систему, которая успешно работает не только на платформе Intel, но и на многих других платформах.

В-четвертых, все больше компаний-производителей прикладного программного обеспечения сообщают, что их продукты портированы для Linux.

Все это вместе взятое позволяет полагать, что Linux становится заметной альтернативой прочим операционным системам для ноутбуков, настольных компьютеров, серверов средней производительности и является одной из важных операционных платформ наравне с AIX, IRIX, HPUX, NT, Solaris.

А надо ли вообще глубоко изучать устройство Linux, чтобы ей пользоваться? Совсем не обязательно всем становиться автомеханиками, но иметь общие представления об устройстве автомобиля и его возможностях очень полезно, чтобы успешно водить машину в разных условиях или иметь возможность сравнить различные марки автомобилей.

С использованием операционной платформы Linux можно выполнять любую работу, однако в книге мы обратимся в основном к работе с текстами: описаниями, исходными текстами программ на различных языках, письмами, романами, руководствами, рефератами, наконец, просто заметками для себя. Нетрудно понять, что в любой деятельности, связанной с компьютером, почти всегда имеет место работа с каким-либо текстом.

Хотелось бы обратить внимание отдельно, что иногда тексты создаются рассредоточенной в географическом смысле группой лиц. И даже в тех случаях, когда вы создаете текст в одиночестве, вы пользуетесь какими-то текстовыми материалами из Интернет, интранет или локальной базы дан-

ных на вашем компьютере. Иначе говоря, создание новых текстов есть творческий процесс группы лиц по преобразованию имеющихся текстов. Итак, перед вами книга о том, как анализировать и преобразовывать текстовые файлы в системе Linux.

Настоящая книга адресована широкому кругу читателей: как новичку, так и опытному пользователю Linux.

Большая часть средств, рассматриваемая автором, является свободно распространяемыми продуктами, разработанными под эгидой проекта GNU (http://www.gnu.org).

1. Linux и источники информации о системе

Основные факты о Linux

История появления Linux изложена во множестве книг, а также в Интернете; здесь мы ограничимся лишь перечислением основных событий.

Linux — это один из вариантов или диалектов популярной операционной системы UNIX, которая была разработана в начале 70-х. Популярность UNIX легко объяснить с точки зрения портируемости UNIX, то есть возможности установки UNIX на машинах различной архитектуры. Модульность и структурированность системы позволяла, переделав относительно небольшое количество составляющих программных компонентов, запустить систему на совершенно новой машине за относительно небольшое время. Это свойство оказалось решающим в 80-е при выборе операционных платформ для процессоров типа RISC: почти на всех процессорах общего назначения были установлены различные варианты UNIX. Так, компания IBM разработала свой вариант UNIX, названный AIX; диалекты UNIX, разработанные компаниями SGI, CRAY, Hewlett Packard, Sun Microsystems, SCO называются соответственно IRIX, UNICOS, HPUX, Solaris, SCO Unix. Существуют и многие другие варианты UNIX.

Кстати, о названии UNIX (произносится «юникс»): слово «UNIX» до появления операционной системы ничего не означало. Это название появилось в противовес названию проекта грандиозной операционной системы конца 60-х годов MULTICS. Как правило, UNIX пишется латинскими буквами и воспринимать это слово следует как своеобразную идиому.

Ядро операционной системы Linux для процессора Intel 80386 было разработано финским студентом по имени Linus

Torvalds (Линус Торвальдс), который продолжает развивать ядро с помощью многих десятков добровольных помощников со всего мира. Основная часть остальной системы — утилиты, языки программирования, сервисные подсистемы — была взяты из проекта GNU. Все это вместе составило новую свободно распространяемую операционную систему.

Нетрудно видеть, что название Linux (читается «линукс»[1]) образовано по аналогии с UNIX. Это тоже идиома.

Кому принадлежат права на Linux

Таким образом, права на ядро системы принадлежит Линусу Торвальдсу, который объявил, что оно может распространяться на условиях GNU General Public License (GNU GPL — основная общественная лицензия GNU). В соответствии с этой лицензией ядро может свободно копироваться, использоваться в любых целях (модификация, продажа, прочее), но вы не имеете права накладывать ограничения на распространение продуктов, которые вы создали на основе ядра, раз вы получили его в соответствии с GNU GPL. Иными словами, если кто-то купил у вас модифицированное ядро, он также имеет право продавать этот продукт, не спрашивая вашего разрешения. Итак, любой человек может свободно копировать любой доступный вариант Linux и поступать с ним так, как считает нужным.

Полностью детали политики копирования можно прочесть в файле COPYING, который входит в исходные тексты ядра Linux.

Различные варианты Linux

В действительности имя Linux используется для большого семейства операционных систем с практически одинаковым ядром (kernel), но с различным составом прикладного про-

[1] Некоторые произносят иначе. Если вы хотите услышать, как произносит это название Линус Торвальдс, вы можете загрузить звуковой файл english.au или swedish.au с сервера ftp://ftp.funet.fi из каталога /pub/Linux/PEOPLE/Linus/SillySounds.

граммного обеспечения. Мы перечислим несколько наиболее популярных компаний, которые поставляют Linux.

RedHat (http://www.redhat.com/) — компания, которая поставляет один из распространенных вариантов Linux. ОС на основе варианта RedHat используется в ряде крупнейших в мире ядерно-физических исследовательских центров, например, в CERN (Conseil Europeen pour la Recherche Nucleaire, Швейцария) и в FNAL (Fermi National Accelerator Laboratory, США). В составе RedHat можно получить библиотеку Motif, клиентскую часть CDE, другие программные пакеты. RedHat Linux поддерживает различные аппаратные платформы: Intel, Alpha, Sparc. RedHat внедрила пакет RPM для распространения перекомпилированного программного обеспечения. RPM стал стандатом де-факто для многих вариантов Linux.

Caldera (http://www.caldera.com/) поставляет систему OpenLinux,wabi в которую входят такие программы, как Wabi (позволяет запускать в Linux приложения для MS Windows 3.x), StarOffice (офисный пакет), ряд утилит для интеграции Linux с NetWare.

S.u.S.e. (http://www.suse.de/e/linux.html) — немецкая компания, которая поставляет Linux вместе с описаниями на нескольких языках (немецкий, английский, французский). В состав набора из 6 компакт-дисков входит довольно много программного обеспечения и разнообразной информации о Linux. В поставку входят СУБД ADABAS, офисный пакет Applixware, а также программа Wabi.

Slackware (http://www.slackware.com) — популярный вариант Linux, который поддерживается энтузиастом по имени Patrick Volkerding. Slackware Linux свободно доступен для копирования в Интернете.

Debian (http://www.debian.org/) — некоммерческий дистрибутив Linux, поддерживающий несколько различных аппаратных платформ и предлагающий большой выбор некоммерческого программного обеспечения.

UrbanSoft (http://www.usoft.spb.ru/) — российская компания, находящаяся в Санкт-Петербурге, которая поставляет Linux и другое открытое программное обеспечение.

Как показывает опыт, выбор варианта Linux должен определяться вашими рабочими контактами. Так, если все ваши коллеги используют, например, SuSe, то использовать без серьезных на то причин другой вариант неразумно. Если вы устанавливаете Linux дома, и у вас нет особых предпочтений, то можно поставить любой вариант. Драматических отличий вы не увидите.

На настоящий момент существует уже несколько десятков компаний (и постоянно появляются все новые), которые поставляют тот или иной вариант Linux. Читатель сможет сам в этом убедиться, если попробует разыскать список поставщиков Linux в Интернете.

Linux и информация о ней в World Wide Web

Замечание о WWW

В пространстве WWW имеется огромное количество информации обо всем, в том числе и о Linux. Однако пространство Web весьма переменчиво: то, что вчера было доступно, сегодня сменило адрес или вообще исчезло из поля зрения. Поэтому автору традиционной книги приходится указывать несколько адресов в Web, поскольку часть из них может попросту пропасть к моменту выхода книги: например, из-за смены системного администратора сервера или просто по недосмотру. Чем больше времени прошло с момента написания книги, тем менее вероятно, что вы что-то найдете по указанным адресам. Вероятно — это жизнь. Тем не менее попробуйте — говорят, что информацию невозможно уничтожить.

Несколько популярных сайтов, посвященных Linux

Операционная система Linux довольно богато представлена во Всемирной паутине. Только перечисление даже хорошо известных адресов заняло бы немало страниц. Здесь мы кратко охарактеризуем лишь несколько сайтов.

http://hepwww.ph.qmw.ac.uk/HEPpc/

Linux Resources for High Energy Physics (Ресурсы Linux для физики высоких энергий) — один из наиболее привлекательных адресов для студентов и исследователей, работающих в области физики высоких энергий. Но и любой «юниксоид» найдет на этих страницах полезную информацию.

http://www.linux.org/

Одна из наиболее известных страниц по всем вопросам, связанным с программным обеспечением для Linux.

http://sunsite.unc.edu/mdw/linux.html

Linux Documentation Project (LDP) — проект «Документация для Linux». На этой странице можно найти ссылки на всевозможные документы, описания и вообще любые тексты, имеющие отношения к Linux. Нетрудно догадаться, что подавляющее большинство документов — на английском языке или на похожем на него техническом жаргоне.

Полезная информация имеется также на Web-серверах компаний, которые продают Linux или программные продукты для Linux.

Еще один способ найти что-то конкретное о Linux — это использовать любую поисковую систему в Интернете, например, http://www.metacrawler.com/.

Например, если вас интересуют программное обеспечение для сетевых адаптеров и связанные с ним проблемы, в качестве ключей для поиска вводите «Linux network card». В ответ вы получите одну или несколько страниц, содержащих список адресов, по которым можно найти информацию о сетевых адаптерах под Linux. Воспользовавшись указанной поисковой машиной и введя упомянутые ключи для поиска, я получил список адресов на трех страницах.

Кроме Web-страниц, полезно пользоваться группами новостей по Linux, часть из которых перечислена во введении.

Описания компонентов Linux и родственная информация на русском языке

http://www.linux.org/books/basic.html

На этой странице можно найти ссылки на страницы руководства (man pages) по Unix (Linux), переведенные на русский язык. Многие из переводов выполнены Виктором Вислобоковым (victor_v@permonline.ru). На сентябрь 1998 года было переведено уже около 200 описаний команд.

http://xtalk.price.ru/linux/

Страница содержит разнообразную информацию о Linux и ссылки на русскоязычные сайты о Linux.

http://www.nevod.ru/linux/doc/

Тут вы найдете переводы различных пособий по программам для UNIX, таким как sed и awk. Кроме того, с этой страницы доступен перевод книги М. Уэлша «Инсталляция Linux и первые шаги» (M. Welsh «Linux Installation and Getting Started»).

Библиотека Мошкова

Можно также заглянуть на страницу «Библиотека Мошкова» http://lib.ru, на которой, среди прочего, вы найдете и тексты, связанные с UNIX и Linux.

IRC и Linux

На странице http://koi.aha.ru/~agb/ указаны серверы, на которых вы можете найти IRC-канал \#ruslinux. На этом IRC-канале можно на русском языке поговорить о Linux. Среди этих серверов: irc.mo.net, irc.emory.edu, irc-w.primenet.com, irc-e.primenet.com, irc.sprynet.com).

Вы также можете попробовать найти русские ресурсы по Linux на перечисленных ниже страницах:

- http://cclib.nsu.ru/projects/gnudocs/;
- http://hh.demos.su/~slackl/;
- http://knot.pu.ru/faq/xfaq.html;

- http://m66.nevod.perm.su/service/linux/doc/;
- http://nexus.odessa.ua/linux/;
- http://win.www.netclub.ru/Russian/linux.html;
- http://www.dkd.ot.lt/hompag/linux/default.htm;
- http://www.linux.inf.ru;
- http://www.uco.ru/~garris/;
- http://www.usoft.spb.ru.

FTP-серверы с информацией о Linux

Вы также можете найти огромное количество документации и программ для Linux на FTP-серверах. Поиск по FTP-серверам можно произвести с помощью поисковой машины, например – http://ftpsearch.ntnu.no/. Если вы зададите в качестве ключа поиска слово «Linux», то полученный список FTP-серверов будет состоять из многих сотен строк. К счастью, есть возможность ограничивать количество найденных серверов. В качестве примера я также попробовал найти FTP-серверы по слову «Linux», ограничив их количество тридцатью. Результат приведен ниже.

Case insensitive substring search for linux

```
-------------------------------------------------------------
ftp.doc.ic.ac.uk        /Mirrors/ftp.tex.ac.uk/
                        tex-archive/tools/zip/info-zip/
                        UNIX/LINUX
ftp.doc.ic.ac.uk        /Mirrors/ftp.cdrom.com/pub/
                        infozip/OLD/UNIX/LINUX
ftp.doc.ic.ac.uk        /Mirrors/ftp.cdrom.com/pub/
                        infozip/UNIX/LINUX
ftp.doc.ic.ac.uk        /Mirrors/ftp.ncsa.uiuc.edu/HDF/
                        contrib/LINUX
ftp.kreonet.re.kr       /.3/CTAN/tools/zip/info-zip/UNIX/
                        LINUX
ftp.kreonet.re.kr       /.3/tools/compress/infozip/OLD/
                        UNIX/LINUX
ftp.kreonet.re.kr       /.3/tools/compress/infozip/UNIX/
                        LINUX
ftp.kreonet.re.kr       /.3/tools/compress/infozip-nonUS/
                        UNIX/LINUX
```

```
ftp.kreonet.re.kr       /.3/tools/compress/infozip-nonUS/
                        OLD/UNIX/LINUX
ftp.in-chemnitz.de      /afs/tex/tools/zip/info-zip/UNIX/
                        LINUX
ftp.ncsa.uiuc.edu       /HDF/contrib/LINUX
ftp.cis.nctu.edu.tw     /Vendors/TekrAm/SCSI/LINUX
ftp.cs.columbia.edu     /.archives/networking/uunet/
                        archival/zip/UNIX/LINUX
ftp.univ-evry.fr        /.01/archiving/infozip/UNIX/LINUX
ftp.univ-evry.fr        /.01/archiving/infozip/OLD/UNIX/
                        LINUX
ftp.chg.ru              /.3/TeX/CTAN/tools/zip/info-zip/
                        UNIX/LINUX
ftp.iij.ad.jp           /TeX/CTAN/pub/tex-archive/tools/
                        zip/info-zip/UNIX/LINUX
ftp.jaist.ac.jp         /.cached2/lang/CTAN/tools/zip/
                        info-zip/UNIX/LINUX
ftp.cdrom.com           /.1/tex/ctan/tools/zip/info-zip/
                        UNIX/LINUX
ftp.univ-evry.fr        /.04/text/TeX/CTAN/tools/zip/
                        info-zip/UNIX/LINUX
ftp.cv.nrao.edu         /aips/15APR98/LINUX
ftp.cv.nrao.edu         /aips/BIN/LINUX
ftp.cdrom.com           /.12/infozip/UNIX/LINUX
ftp.cdrom.com           /.12/infozip/OLD/UNIX/LINUX
ftp.doc.ic.ac.uk        /Mirrors/quest.jpl.nasa.gov/pub/
                        UNIX/LINUX
ftp.doc.ic.ac.uk        /Mirrors/quest.jpl.nasa.gov/pub/
                        LINUX
ftp.cv.nrao.edu         /aips/DA00/LINUX
ftp.tekram.com          /SCSI/LINUX
ftp.stalker.ru          /Pub/HardSoft/SoftForHard/Net/
                        SureCom/EP325/LINUX
ftp.vislist.com         /SHAREWARE/ENVIRONMENTS/LINUX
Next
-------------------------------------------------------------
30 repored hits
0.014 seconds prospero
0.017 seconds HTTP
0 partial writes.
DONE
-------------------------------------------------------------
Hardware by FAST ASA, ITEA and IDI. Network by UNINETT.
NTNU - Norwegian University of Science and Technology
This server runs FreeBSD and is located in Trondheim,
Norway FAST FTP Search, Copyright (C) 1998 FAST ASA.
```

Рекомендуем также особое внимание обратить на сервер ftp://metalab.unc.edu — один из важнейших FTP-серверов, связанных с Linux.

Прикладные программные пакеты для Linux

Естественно, один из первых серверов, на которые следует заглянуть при поиске необходимых пакетов — это www-сервер проекта GNU http://www/gnu.org. Как известно, GNU is not Unix (то есть GNU — это не Unix). Эта шутливая рекурсия — расшифровка аббревиатуры GNU. Проект GNU посвящен созданию свободно распространяемого программного обеспечения. Более подробные сведения о проекте и его продуктах можно найти на сервере GNU и на многих других серверах по всему миру.

Одним из полезных мест в WWW-пространстве является Scientific Applications on Linux (SAL) — коллекция научных прикладных программ http://SAL.KachinaTech.COM/index.shtml. Термин «научные программы» не означает «неприменимые в обычной жизни». Просто это один из больших списков полезных программных продуктов, работающих под Linux, которые подходят и для научных исследований. Эти программы имеются на многих серверах в Интернете. Вы можете обратиться на ближайший к вам сервер из нижеприведенного списка:

- Australia http://sal.rising.com.au/;
- Austria http://nswt.tuwien.ac.at/scicomp/sal/;
- Colombia http://linuxsite.univalle.edu.co/sal/;
- Finland http://sal.jyu.fi/;
- France http://www-sor.inria.fr/mirrors/sal/;
- Germany http://ftp.llp.fu-berlin.de/lsoft/;
- Italy http://chpc06.ch.unito.it/linux/;
- Japan http://ec.tmit.ac.jp/koyama/linux/sal/;
- New Zealand http://nix.tmk.auckland.ac.nz/sal/;

- Poland http://www.tuniv.szczecin.pl/linux/doc/other/sal/;
- Portugal http://www.idite-minho.pt/sal/;
- Russia (Moscow) http://www.sai.msu.su/sal/;
- Russia (Novosibirsk) http://www.siblug.org/sal/;
- South Africa http://web.ee.up.ac.za/sal/;
- South Korea http://bioinfo.bioneer.com/sal;
- Spain http://ceu.fi.udc.es/sal/;
- Turkey http://sal.raksnet.com.tr/;
- United Kingdom http://www.ch.qub.ac.uk/sal/;
- USA http://SAL.KachinaTech.com/.

Другие интересные прикладные пакеты появляются на странице http://www.redhat.com/linux-info/linux-app-list/linapps.html, которую поддерживает компания RedHat. Также обратите внимание на страницу с анонсами нового программного обеспечения для Linux — http://freshmeat.net.

Для каких аппаратных платформ доступна Linux

Ответ на этот вопрос очень прост — для всех широко известных и еще для нескольких, которые известны лишь узкому кругу специалистов. Тем не менее, автор хотел бы указать несколько конкретных источников по Linux для ряда не-Intel платформ.

Информацию о Linux на платформе PowerPC можно найти на страницах http://www.linuxppc.org/ и http://www.yellowdoglinux.com. Информация о Linux/m68k на платформах Motorola 68020, 68030, 68040, 68060 и близких к ним может быть почерпнута по адресу http://www.linux-m68k.org/. FAQ (список часто задаваемых вопросов и ответов на них) на эту же тему находится по адресу http://www.linux-m68k.org/faq/faq.html.

Довольно богато представлена информация по Linux на процессорах Alpha (Digital) в группе новостей news:comp.os.linux.alpha. Информация о переносе Linux на платформу 64-bit DEC Alpha/AXP имеется на http://www.azstarnet.com/

~axplinux/. За информацией о платформе MIPS можно обратиться на серверы ftp://ftp.fnet.fr/linux-mips и ftp://ftp.linux.sgi.com/pub/mips-linux. Информация о Linux на платформе Sun Sparc имеется во многих местах, например, можно посмотреть сервер компании RedHat http://www.redhat.com. Имеются архивы по Sun Linux – на странице http://www.geog.ubc.ca/sparclinux.html, а также на FTP-сервере ftp://vger.rutgers.edu/pub/linux/Sparc.

Для обсуждения конкретных деталей переноса Linux можно пользоваться списком рассылки по ядру Linux, который поддерживается на узле vger.rutgers.edu. Довольно полный список платформ, на которые перенесена Linux, а также список различных вариантов Linux приведен на странице http://www.linuxhq.com/dist-index.html, а также на странице http://www.ctv.es/USERS/xose/linux/linux_ports.html.

Книги по Linux

Что касается литературы по Linux, то существует множество книг по этой теме – самого разного назначения. Например, если взглянуть на страницу http://www.linux.org/books/basic.html, то можно найти список, состоящий из кратких аннотаций.

Далее на странице приведен список из 30 или 40 названий книг разных издательств, направлений и объема – от 160 до 1600 страниц. Цена может варьироваться от $19 до $70.

Таким образом, есть из чего выбирать. Кроме книг, имеется ряд газет и журналов, посвященных разным аспектам применения Linux. Объем печатной продукции по Linux на английском языке таков, что обычный человек не сможет прочесть все это даже за целую жизнь. Этот факт лишний раз подчеркивает популярность операционной системы Linux.

Книги по Linux на русском языке

На серверах различных российских издательств нетрудно отыскать несколько книг по Linux. Однако большинство этих книг – переводы соответствующих англоязычных оригиналов.

Поскольку Linux – один из вариантов UNIX, то возникает естественный вопрос, а можно ли использовать книги по

UNIX на русском языке для обучения работе с Linux? Во многих случаях можно. Если вы начали пользоваться системой недавно, то на уровне пользовательского интерфейса у них есть много общего, и вы не допустите серьезных ошибок. Что касается пакетов, разработанных GNU, то они просто одинаковы во всех вариантах операционных систем и отличия определяются лишь версией самого пакета.

Другие источники информации о Linux

Существует отличный комплект англоязычных документов по Linux, так называемых «HOWTO» («как сделать что-либо»), например, «как настроить принтер». В этих документах можно найти ответы на большинство практических вопросов. Получить вы их можете на многих серверах, например, http://sunsite.unc.edu/pub/Linux/docs/HOWTO/.

Кроме того, имеется несколько групп новостей, посвященных различным аспектам Linux. Если вы планируете работать в Linux, то вам необходимо знакомиться с новостями группы comp.os.linux.announce. Тут вы найдете информацию о появлении и обновлении программного обеспечения для этой ОС. Сообщения в эту группу должны направляться по адресу: linux-announce@news.ornl.gov. Другие полезные группы перечислены ниже:

- comp.os.linux.setup;
- comp.os.linux.hardware;
- comp.os.linux.networking;
- comp.os.linux.x;
- comp.os.linux.development.apps;
- comp.os.linux.development.system;
- comp.os.linux.advocacy comp.os.linux.misc.

Информационные команды системы

Найти какую-либо информацию, имея под руками систему с установленной Linux, можно несколькими различными способами. Рассмотрим наиболее важные.

Команда uname

Команда `uname` поможет вам узнать, какой вариант Linux установлен на вашей машине:

```
uname -a
```

В ответ система напечатает что-то в духе:

```
Linux pcfarm.pnpi.spb.ru 2.0.33 \#18 Thu Jun 4 11:54:03
MSD 1998 i686 unknown
```

Это означает: операционная система — Linux, имя машины в сети — pcfarm.pnpi.spb.ru, версия ядра операционной системы — 2.0.33, версия обновления ядра и дата создания ядра системы — `\#18 Thu Jun 4 11:54:03 MSD 1998`, тип машины — `i686`, тип процессора — `unknown`.

Как прошла загрузка системы

Сообщения, выведенные ядром системы при загрузке, могут быть получены при помощи команды `dmesg`.

Программа procinfo

С помощью программы `procinfo` можно получить массу полезной информации (как статической, так и динамической) о работающей системе. Какую именно информацию выводит `procinfo`, можно узнать с помощью команды:

```
procinfo -h
```

Команда man

Чтобы получить краткое описание какой-то команды или термина в системе, можно использовать команду `man`. Если вас интересует информация о термине modem, то можно использовать команду

```
man -k modem
```

Параметр `-k` обозначает, что далее следует ключевое слово, по которому должен производиться поиск. После выполнения указанной команды на экран будет выведен список команд и функций, которые как-то связаны с заданным ключевым словом. Так, в RedHat Linux вы можете увидеть следующее:

```
efax (1)        - send/receive faxes using
                  Class 1 or 2 fax modems
mgetty (8)      - smart modem getty
rx, rb, rz (1) - XMODEM, YMODEM, ZMODEM (Batch) file
                  receive
sendfax (8)     - send group 3 fax files (G3 files)
                  with a class 2 faxmodem
statserial (1) - display serial port modem status lines
sx, sb, sz (1) - XMODEM, YMODEM, ZMODEM file send
zplay (1)       - modem utility to record and play
                  voice files
XF86VidModeQueryExtension, XF86VidModeQueryVersion,
XF86VidModeGetModeLine, XF86VidModeGetAllModeLines,
XF86VidModeD eleteModeLine, XF86VidModeModModeLine,
XF86VidModeValidateModeLine, XF86VidModeSwitchMode,
XF86VidModeSwitchToMode, XF86VidModeLockModeSwitch,
XF86VidModeGetMonitor, XF86VidModeGetViewPort,
XF86VidModeSetViewPort (3) - XFree86-VidMode
extension interface functions
```

Далее вы сможете снова воспользоваться командой man, чтобы просмотреть справочные страницы, соответствующие элементам полученного списка.

Вариант программы man для работы в среде X Window называется xman.

Команда info

Очень удобна также команда info. Она имеет формат info раздел. Если раздел не указан, выводится страница, содержащая список всей установленной в системе документации в формате info. Одной из важных программ, к которой поставляется документация в виде info-файлов, является компилятор языка C gcc (попробуйте набрать команду info gcc).

Команда apropos

Команда apropos также помогает найти информацию среди man-страниц (страницы руководства, от слова «manual»), имеющихся на вашей машине. Если вы попробуете дать команду

```
apropos modem
```

то заметите, что она выдаст примерно то же, что и команда man -k.

Команда helptool

По команде `helptool` появляется графическое окно, в котором вы сможете задать интересующий вас термин. Команда просматривает все файлы документов (вы можете сконфигурировать, какие документы следует просматривать при поиске). По завершении поиска вам будет выдан список файлов, где встречается данный термин. Если щелкнуть мышкой на элементе списка, то появится дополнительное окно, в котором будет отображаться выбранный вами файл. При этом файл будет отображаться в том формате, в каком он хранится на вашей машине: страницы `info`, страницы `man` и т. п.

Команда locate

Наконец, может оказаться полезной программа `locate`. Например, для того чтобы найти файлы, название которых содержит слово `modem`, используйте следующую команду:

```
locate modem
```

Вы увидите на экране примерно следующее:

```
/usr/bin/modemtool
/usr/doc/util-linux-2.7/README.modems-with-agetty
/usr/farm/gcc-2.7.2.3/modemap.def
/usr/lib/rhs/control-panel/modemtool.init
/usr/lib/rhs/control-panel/modemtool.xpm
/usr/lib/rhs/control-panel/python/modem.py
/usr/lib/rhs/control-panel/python/modem.pyc
/usr/share/usernet/1.0.5/modem.xpm
```

Команда rpm

Программа `rpm` используется в большинстве вариантов Linux для установки, удаления, модернизации других программных продуктов. В каждой системе `rpm` ведет базу данных, содержащую информацию об установленных программных пакетах. Программа `rpm` довольно обширна по своим возможностям, и мы отметим лишь несколько ее свойств, которые помогут понять, какое же программное обеспечение установлено на вашей машине.

Команда получения информации о программных пакетах имеет вид:

```
rpm -q параметры
```

Имеется два подмножества параметров команды rpm для получения информации: про какой пакет(ы) (табл. 1.1) и какого типа информация будет напечатана (табл. 1.2). Например, для того чтобы получить весь список установленных программных пакетов на вашей машине, можно использовать:

```
rpm -q -a
```

Таким образом, если вас интересует, установлен ли какой-то вариант пакета LaTeX на машине, введите:

```
rpm -q -a | grep -i latex
```

И вот что вы можете увидеть в результате:

```
tetex-latex-0.4pl8-9
```

А таким образом

```
rpm -q -i tetex-latex-0.4pl8-9
```

можно получить более подробное описание пакета:

```
Name                :   tetex-latex
Distribution        :   Hurricane
Version             :   0.4pl8
Vendor              :   Red Hat Software
Release             :   9
Build Date          :   Wed Oct 22 23:36:05 1997
Install date        :   Fri Jan 16 17:01:13 1998
Build Host          :   porky.redhat.com
Group               :   Applications/Publishing/TeX
Source RPM          :   tetex-0.4pl8-9.src.rpm
Size                :   9911252
Packager            :   Red Hat Software bugs@redhat.com
URL                 :   http://www.tug.org/teTeX/
Summary             :   LaTeX macro package
Description : LaTeX is a TeX macro package. The LaTeX
macros encourage writers to think about the content of
their documents, rather than the form. The ideal, very
difficult to realize, is to have no formatting commands
(like «switch to italic» or «skip 2 picas») in the
document at all; instead, everything is done by specific
markup instructions: «emphasize», «start a section».
```

Таблица 1.1. Параметры команды rpm, определяющие, о каких пакетах приводится информация

Параметр	Описание
-whatrequires service	Показать список пакетов, которые требуют service
-whatprovides virtual	Показать список пакетов, которые обеспечивают службу virtual
-f file	Показать пакет, которому принадлежит файл с именем file
-p file	Показать пакет, которому принадлежит файл с именем file, даже если пакет был удален

Таблица 1.2. Параметры команды rpm, отвечающие за тип предоставляемой информации

Параметр	Назначение
-i пакет	Печатает информацию о пакете
-changelog пакет	Печатает информацию об изменениях в программном пакете
-l пакет	Печатает список файлов пакета
-s пакет	Печатает список файлов пакета вместе с информацией о статусе каждого файла
-d пакет	Печатает список файлов документов указанного пакета
-c пакет	Печатает список конфигурационных файлов пакета
-dump	Печатает всю проверяемую информацию для каждого файла; должна использоваться с параметрами -c, -d, -s, -l
-provides пакет	Печатает свойства, которые обеспечивает данный пакет
-requires пакет	Печатает список файлов и каталогов, которые необходимы для установки данного пакета
-R пакет	То же самое, что -requires
-scripts пакет	Печатает сценарии, которые используются для установки или удаления данного пакета

Примеры получения информации из базы rpm

Поиск пакета, содержащего файл

```
$ rpm -q -f /usr/share/texmf/tex/latex/misc/umlaute.sty
tetex-latex-0.9-17
```

На вопрос, в какой пакет входит файл `../umlaute.sty`, `rpm` ответила, что данный файл входит в пакет `tetex-latex-0.9-17`.

Поиск пакетов, зависящих от данной программы

По команде

```
$ rpm -q -whatrequires tetex
```

на экран выводится список установленных пакетов, зависящих от наличия в системе программы tetex:

```
OB_Java-1.0-1
PSCyr-0.3-3
tetex-afm-0.9-17
tetex-dvilj-0.9-17
tetex-dvips-0.9-17
tetex-latex-0.9-17
tetex-russian-cyrplain-2.1-4
tetex-russian-cyrsam-2.1-4
```

tetex-t2-2.1-4 (определение зависимостей)

Вы также можете определить, от наличия каких файлов и каталогов зависит данный пакет. Пример определения зависимостей пакета a2ps приведен ниже.

```
$ rpm -q -requires a2ps
/sbin/install-info
/bin/sh /sbin/ldconfig
ld-linux.so.2
libc.so.6
libm.so.6
/bin/sh /usr/bin/perl
libm.so.6(GLIBC_2.1)
libc.so.6(GLIBC_2.1)
```

libc.so.6 (GLIBC_2.0, информация из файловой системы /proc)

Файловая система /proc является псевдофайловой системой, и когда вы попробуете заглянуть в нее — будут вызваны соответствующие программы, чтобы отобразить для вас полезную информацию о ядре Linux. Обычно любая информация в /proc отображается командой cat. Здесь мы кратко охарактеризуем различные элементы каталога /proc.

Если выполнить команду ls /proc, то вы увидите систему файлов и подкаталогов внутри этого каталога. Часть элементов имеют имена в виде целых чисел, другая часть имеет имена типа bus, cmdline, cpuinfo и т. д. Вначале рассмотрим числовые имена. Каждое числовое имя в каталоге /proc обозначает псевдо-каталог, который содержит ряд файлов и подкаталогов, содержащих информацию о процессе с таким номером. Например, если существует процесс с номе-

ром 18919, команда `ls /proc/18919` выведет на экран примерно следующий список файлов:

```
cmdline        cpu cwd      environ exe      fd
maps       mem rootstatstatm        status
```

Эти файлы содержат информацию о состоянии конкретного процесса. Содержание этих файлов описано в табл. 1.3.

Почти каждый файл из файловой системы `proc` может быть просмотрен посредством команды `cat`. Например, если вы хотите получить информацию о процессоре, то можно использовать команду

```
cat /proc/cpuinfo
```

Таблица 1.3. Описание файлов и подкаталогов в каталоге `/proc/n`, где n — номер процесса

Имя файла	Содержание
cmdline	Последняя полная командная строка, если только процесс не свопирован на диск или если процесс является зомби
cwd	Ссылка к рабочему каталогу процесса
cpu	Информация о том, сколько процесс потребил времени и на каком процессоре
environ	Значения переменных окружения
exe	Ссылка к двоичному коду исполнямой процессом программы
fd	Псевдокаталог, который содержит файлы с целочисленными именами: 0, 1, 2 и т. д.). Сами файлы из каталога /proc/n/fd являются ссылками к реальным файлам, открытым процессом с номером n. Как обычно, файл /proc/n/fd/0 есть стандартный ввод, /proc/n/fd/1 – стандартный вывод, /proc/n/fd/2 – вывод сообщений об ошибках и т. д.
maps	Распределение областей памяти процесса и доступ к отдельным фрагментам памяти процесса
mem	Карта оперативной памяти до выполнения трансляции адресов
root	Ссылка на корень файловой системы, то есть /
stat	Информация о состоянии процесса. Отсюда берет информацию команда Linux ps
status	Дополнительная информация о процессе

Теперь рассмотрим нечисловые имена; они сведены в табл. 1.4.

Таблица 1.4. Нечисловые имена и содержание файлов и подкаталогов в каталоге /proc

Имя файла	Содержание
cpuinfo	Данные, отражающие архитектуру системы и тип процессора. Формат данных может быть отличен для разных архитектур. Всегда присутствуют два элемента: тип процессора и величина BogoMIPS;, которая вычисляется во время инициализации ядра системы. Эта величина иногда близка к тактовой частоте процессора. Однако частоту следует смотреть в строке cpu MHz
devices	Текстовый список, который содержит перечисление основных номеров и групп периферийных устройств
dma	Список имеющихся каналов DMA
filesystems	Перечисление типов файловых систем, которые были включены в ядро системы во время компиляции ядра
interrupts	Количество прерываний на каждом уровне прерываний IRQ
ioports	Список зарегистрированных используемых портов ввода/вывода
kcore	Физическая память системы в формате core
kmsg	Содержание этого файла может быть использовано вместо утилиты syslog (только для пользователя root)
ksyms	Ссылки в ядре системы, использующиеся для определения внешних программных модулей, которые загружаются динамически по мере необходимости. Можно смотреть man depmod, man modprobe
loadavg	Информация в следующем формате: n.nn m.mm k.kk l/ppp gggg, где n.nn m.mm k.kk – есть средние величины загрузки процессора за последнюю минуту, за последние пять минут и за последние 15 минут. Значение загрузки определяется как среднее количество исполняемых процессов; l/ppp означает, что l процессов из общего числа ppp процессов в машине находятся в состоянии выполнения (в состоянии R)
malloc	Специальный файл, который присутствует, если только во время компиляции ядра был указан параметр CONFIGDEBUGMALLOC
meminfo	Информация об использовании физической оперативной памяти, своп памяти, разделяемой памяти и буферах
modules	Список системных программных модулей, которые загружены системой в данный момент
net	Псевдокаталог, который содержит различные псевдофайлы, содержащие информацию, отражающую текущее состояние разных слоев сетевого протокола. Здесь в более детальном виде содержится информация, которую выдает утилита netstat. Смотрите также man proc
pci	Список всех устройств PCI, имеющихся на данной машине
scsi	Псевдокаталог, содержащий информацию об устройствах на шинах SCSI

Имя файла	Содержание
self	Псевдокаталог, содержащий информацию о процессе, который читает каталог /proc
stat	Псевдокаталог, содержащий информацию о системе (сколько процессор был занят, сколько простаивал, сколько страниц было считано с диска и сколько записано и т. д.)
sys	Псевдокаталог, содержащий массу информации о системе. Среди важных файлов этого каталога можно отметить /proc/sys/fs/file-max – максимальное число открытых файлов в системе (пользователь root может изменить это число)
swaps	Сведения о swap-файле, которым пользуется система
tty	Псевдокаталог, который содержит информацию о терминальных устройствах системы
uptime	Два числа: время в секундах с момента загрузки и время в секундах, когда процессор простаивал
version	Строка, которая отражает текущую версию ядра системы, например, команда cat /proc/version может дать примерно следующее: Linux version 2.2.12-20smp(root@porky.devel.redhat.com) (gcc version egcs-2.91.66 19990314/Linux (egcs-1.1.2 release)) #1 SMP Mon Sep 27 10:34:45 EDT 1999

На каком языке говорит Linux

В этом разделе мы поговорим о локализации. Локализацией называется приспособление программы или операционной системы к кодировке и стилям печати времени, даты, денежных единиц, принятых в данной стране. Стандарт описан, например, в документах POSIX 1.c. Здесь мы отметим важные особенности локализации Linux.

Многие системные программы в Linux (быть может – большинство) для определения вида кодировки и языка сообщений используют значения специальных переменных окружения (табл. 1.5).

Таблица 1.5. Переменные окружения, используемые при локализации

Имя переменной	Описание
LC_ALL	Определяет все сразу: язык сообщений, кодировку, вид даты и т. д. Причем, если эта переменная установлена, то остальные переменные вида LC_* не имеют силы. Однако непосредственная установка этой переменной является плохой идеей – лучше использовать переменную LANG

продолжение ⇙

Таблица 1.5 *(продолжение)*

Имя переменной	Описание
LANGUAGE	Эта переменная используется в основном программами из проекта GNU
LC_CTYPE	Определяет кодировку символов
LC_COLLATE	Определение алгоритма сортировки
LC_MONETARY	Определение вида денежной единицы
LC_NUMERIC	Определение национального формата печати чисел (например, она указывает, использовать точку или запятую для отделения целой и дробной части числа)
LC_TIME	Определение формата даты и времени
LANG	Эта переменная окружения используется для определения всего того, что не установлено ранее переменными вида LC_*. В принципе, из всех перечисленных выше переменных – это единственная переменная, значение которой вам следует установить (если, конечно, вы не знаете наверняка, что вам нужно поменять еще что-либо)

Чтобы увидеть, какие переменные из вышеописанных и как установлены, можно использовать программу `locale`.

```
[shevel@pcfarm BOOK]$ locale
LANG=ru_RU.KOI8-R
LC_CTYPE="ru_RU.KOI8-R"
LC_NUMERIC="ru_RU.KOI8-R"
LC_TIME="ru_RU.KOI8-R"
LC_COLLATE="ru_RU.KOI8-R"
LC_MONETARY="ru_RU.KOI8-R"
LC_MESSAGES="ru_RU.KOI8-R"
LC_ALL=
```

Чтобы узнать, какие имеются варианты локализации на вашей машине, просмотрите список подкаталогов в каталоге `/usr/share/locale/`, либо воспользуйтесь командой `locale -a`.

Общий вид строк, которые присваиваются переменным окружения, должен иметь вид `язык\_территория.кодировка`.

Следует заметить, что часто одни и те же кодировки называются чуть по-разному. Например, автору приходилось видеть следующие обозначения: `koi8`, `koi8r`, `koi8-r` и `KOI8-R`. Хотя последний вид обозначения встречается нечасто, он соответствует стандарту.

Полезные сведения о локализации можно почерпнуть в man-странице `locale(1)`. (Для ее вывода используйте команду `man locale`).

2. Базовые компоненты Linux

Оболочки

Ободочка (shell), или командный интерпретатор, представляет собой программу, которая анализирует и выполняет команды, вводимые с терминала. Командный интерпретатор не требует для себя каких-то специальных привилегий и рассматривается системой как обычная программа. Существует довольно много различных командных интерпретаторов. К типичным свойствам интерпретатора можно отнести интерпретацию командных последовательностей (scripts — сценариев или «скриптов»), расширение имен файлов, которые содержат знак * (звездочка) или другие метасимволы, реализацию программных каналов (pipes), повторное исполнение ранее выполненных команд, выполнение циклов и условных переходов, а также создание псевдонимов (aliases). Сценарий или командная последовательность представляет собой текстовый файл, который содержит команды в том виде, в котором они могли бы вводиться с терминала.

Очевидно, что записанный сценарий удобно использовать, чтобы не повторять ввод одной и той же последовательности команд интерпретатора вручную. Достаточно ввести последовательность команд лишь однажды, и после проверки (отладки) сценарий можно использовать сколько угодно раз.

Команды

Список команд Linux весьма обширен и содержит несколько сотен команд или более. Многие команды выполняют схожие функции. Совсем не обязательно знать все имеющиеся команды или приложения. Достаточно освоить некоторый минимум, который удовлетворит 90% ваших потребностей. В этой главе мы рассмотрим несколько команд, без которых нельзя продуктивно работать.

Под командой мы будем понимать последовательность слов, разделенных одним или несколькими пробелами. Первое слева слово определяет имя команды, которую следует выполнить. Команда может состоять из одного имени, например, `w`. За командой могут следовать параметры; как правило, им предшествует знак - (дефис). Параметры команды часто состоят из одной буквы. Например, `ls -l` или, в случае нескольких параметров, `ls -l -t`.

Если параметров несколько, то они могут быть объединены; например, предыдущий пример может быть переписан в виде `ls -lt`.

За параметрами могут следовать аргументы, например, имена файлов: `ls -lt A*`.

Как правило, каждый аргумент состоит из одного слова. Аргумент может относиться как к самой команде, так и к отдельному параметру. Например, в команде `awk -f AwkProg FileName` аргумент `AwkProg` относится к параметру `-f`, а аргумент `FileName` — ко всей команде.

Два дефиса подряд обозначают окончание списка параметров. Например, команда `rm -- -abc` удалит файл с именем `-abc`. Если вы попробуете выдать команду `rm -abc`, то имя файла будет воспринято как параметр, и вы получите сообщение об ошибке.

Наконец, еще одно значение знака - (дефис) состоит в том, что если знак используется на месте имени файла, то вместо файла используется стандартный ввод (или вывод).

Команды могут состоять из нескольких простых команд.

Команда, которая читает данные из стандартного потока ввода, преобразует их каким-то образом и выводит результат в стандартный поток вывода, называется фильтром.

Программным каналом (pipe) называется последовательность двух или более команд разделенных знаком | (вертикальная черта). Суть программного канала состоит в том, что стандартный вывод каждой команды, стоящей слева от вертикальной черты, передается на стандартный ввод команды, стоящей справа от вертикальной черты.

Список — это несколько (возможно, один) программных каналов, разделенных знаками ;, &, && или ||, и, возможно, ограниченных знаками ; или &. Из этих четырех знаков ; и & имеют равный приоритет, который ниже, чем приоритет у знаков && и ||. Приоритет двух последних знаков также одинаков. Точка с запятой (;) приводит к последовательному выполнению программного канала, за которым она стоит; амперсанд (&) приводит к асинхронному выполнению программного канала, за которым он стоит. Иными словами, оболочка-интерпретатор не будет ожидать завершения выполнения программного канала, а перейдет немедленно к интерпретации следующего программного канала. Символ && (||) приведет к тому, что следующий список будет интерпретироваться только в том случае, если текущий программный канал завершился с нулевым (ненулевым для ||) кодом завершения. В качестве ограничителя команд вместо точки с запятой могут использоваться также один или более знаков перевода строки.

Сценарии

Как уже было отмечено, сценарий — это файл, содержащий последовательность команд оболочки. Этот файл может быть приготовлен с помощью любых средств, как обычный текстовый файл, например, он может быть написан в любом текстовом редакторе или создан командой `cat`:

```
$ cat > TestExecFile
echo "This is first exec file."
<Ctrl-d>
$ chmod +x TestExecFile
```

ПРИМЕЧАНИЕ <Ctrl-d> в предыдущем примере означает, что на клавиатуре терминала должны быть нажаты одновременно две клавиши: <Ctrl> и <d> (буква «d»). В Linux ввод такой комбинации воспринимается как конец файла при вводе текста или как конец сеанса при вводе команд. Конец сеанса вызовет специальную процедуру выхода из системы. После выполнения такой процедуры работать с системой будет возможно лишь после выполнения процедуры входа в систему.

Если попробовать запустить созданный выше сценарий `./TestExecFile`, то на экране будет напечатана строка `This is first exec file`.

Сценарий можно создать при помощи другого сценария. Например, нижеприведенный текст создает скелет (основу) нового сценария:

```
#!/bin/bash
#set -x
#Call c_script script_name
SCRIPT_NAME=$1
if [ "" = "$1" ]; then
       echo
       echo "Usage: c_script scriptname"
       echo
       exit
fi
echo "#"   > $SCRIPT_NAME
echo "#!/bin/bash" >> $SCRIPT_NAME
echo       >> $SCRIPT_NAME
echo "#-----------------------------------------+" >>
$SCRIPT_NAME
echo "#        |" >> $SCRIPT_NAME
echo "# Usage:" $SCRIPT_NAME    >> $SCRIPT_NAME
echo "#        |" >> $SCRIPT_NAME
echo "# The script is dedicated to: |" >> $SCRIPT_NAME
echo "#        |" >> $SCRIPT_NAME
echo "# Creation date: " `date` "   |" >> $SCRIPT_NAME
echo "# The host where it was developed="`hostname` " |" >>
$SCRIPT_NAME
echo "# History of changes:     |" >> $SCRIPT_NAME
echo "#        |" >> $SCRIPT_NAME
echo "#-----------------------------------------+" >>
$SCRIPT_NAME
echo "#        |" >> $SCRIPT_NAME
echo "#  Author: Andrei Chevel. email:
Andrei.Chevel@pnpi.spb.ru |" >>$SCRIPT_NAME
echo "#-----------------------------------------+" >>
$SCRIPT_NAME
echo       >> $SCRIPT_NAME chmod +x $SCRIPT_NAME  # To
make the script executable
```

Легко видеть, что приведенный сценарий во время исполнения формирует основу нового сценария и автоматически помещает в виде комментариев ряд информационных строк.

Обратим внимание на некоторые особенности текста любого сценария.

Строка комментария помечается знаком \# (решетка). Исключение составляет первая строка: если за \# следует знак ! (восклицательный знак), то далее указывается абсолютное имя программы, которой должен быть передан сценарий для интерпретации. Если программа для выполнения интерпретации не указана, то сценарий интерпретируется оболочкой, используемой в вашей системе по умолчанию.

Полезно самому установливать тип интерпретатора. В качестве интерпретатора может использоваться любая программа. Например, если вы предпочитаете интерпретатор `bash`, то в качестве первой строки сценария можно записать:

```
#!/bin/bash
```

(Конечно, если `bash` находится именно в каталоге `/bin`.)

Для того чтобы сценарий распознавался операционной системой как исполняемый файл, следует выполнить команду `chmod u+x TestExecFile` либо просто `chmod +x TestExecFile`.

Оболочка sh

Это самая старая оболочка, которая, быть может, менее дружественна по отношению к пользователю, чем остальные, но она точно имеется на любой машине, на которой установлен вариант UNIX или Linux. Во многих случаях, если не указана оболочка, в которой следует выполнять командную последовательность, то она выполняется в оболочке `sh`.

Перенаправление вывода

До того как команда выполнится, ее ввод и/или вывод могут быть перенаправлены с использованием специальной нотации, которую интерпретирует оболочка. Перенаправление может быть также использовано, чтобы открыть или закрыть файлы для текущего окружения оболочки. Перенаправление обрабатывается оболочкой в том порядке, в котором оно указано — слева направо.

В табл. 2.1, если номер файла отсутствует и первым символом оператора перенаправления является < (знак меньше), то перенаправление подразумевает стандартный ввод (номер файла – 0). Если первый символ оператора перенаправления есть > (знак больше), то перенаправление относится к стандартному выводу (номер файла – 1). Возможные варианты перенаправления приведены в табл. 2.1.

Таблица 2.1. Перенаправление ввода/вывода в оболочке sh

Вид перенаправления	Значение
<file_in	Использовать файл с именем file_in в качестве стандартного ввода
>file_out	Использовать файл с именем file_out в качестве стандартного вывода. Если файл существует, то его содержимое заменяется. В противном случае файл создается
<<word	Оболочка-интерпретатор читает ввод до строки, которая содержит слово word или до конца файла. Предварительно над словом выполняются подстановки, если они необходимы (например, раскрываются значения подстановочных знаков). Однако если слову word предшествует знак - (дефис), то лидирующие символы табуляции удаляются из слова word после подстановки, но до начала чтения ввода; лидирующие знаки табулятора удаляются из читаемых строк до сравнения со словом word; наконец, оболочка-интерпретатор читает ввод до первой строки, которая содержит слово word или до конца файла
<&digit	Использовать файл, связанный с номером digit, в качестве стандартного ввода. Подобным же образом для стандартного вывода используется нотация >&digit
<&	Стандартный ввод закрывается. Подобная нотация используется и для стандартного вывода >&

Если за командой следует знак & (амперсанд), то стандартным вводом для такой команды по умолчанию является пустой файл `/dev/null`.

Сравнительные характеристики оболочек

Ниже приведена таблица, которая содержит информацию о сравнительных особенностях различных оболочек-интерпретаторов. Автор взял основные данные для этой таблицы со страницы http://www.looking-glass.org/shell.html.

Таблица 2.2. Сравнение оболочек

Возможность	sh	csh	ksh	bash	tcsh	zsh
Управление заданиями	есть	есть	есть	есть	есть	есть
Определение псевдонимов	есть	есть	есть	есть	есть	есть
Определение функций	есть[1]	нет	есть	есть	нет	есть
Чувствительность к переопределению ввода/вывода	да	нет	да	да	нет	да
Стек каталогов	нет	да	да	да	да	да
История команд	нет	да	да	да	да	да
Редактирование в командной строке	нет	нет	да	да	да	да
Редактирование в командной строке в стиле редактора Vi	нет	есть	есть	есть[2]	есть	есть
Редактирование в командной строке в стиле редактора Emacs	нет	нет	есть	есть	есть	есть
Переопределение режимов редактирования в командной строке	нет	есть	есть	есть	есть	есть
Возможность инициировать автоматический выход из системы после определенного числа минут неактивности	нет	есть	есть	есть	есть	есть
Возможность наблюдения за активностью отдельных пользователей или терминалов	нет	нет	нет	нет	есть	есть
Автоматическое дополнение имен файлов	нет	есть[1]	есть	есть	есть	есть
Автоматическое дополнение имен пользователей	нет	есть[2]	есть	есть	есть	есть
Автоматическое дополнение имен узлов	нет	есть[2]	есть	есть	есть	есть
Дополнение команд из файла истории	нет	нет	нет	есть	есть	есть
Полностью программируемое (конфигурируемое) дополнение	нет	нет	нет	нет	есть	есть
Многозадачность	нет	нет	есть	нет	нет	есть
Встроенные арифметические вычисления	нет	есть	есть	есть	есть	есть
Периодическое выполнение команд (set tperiod)	нет	нет	нет	нет	есть	есть
Настройка приглашения	нет	нет	есть	есть	есть	есть
Проверка написания команд	нет	нет	нет	нет	есть	есть
Подстановка процессов	нет	нет	нет	есть[2]	нет	есть
Основной (базовый) синтаксис команд	sh	csh	sh	sh	csh	sh

продолжение

Таблица 2.2 *(продолжение)*

Возможность	sh	csh	ksh	bash	tcsh	zsh
Оболочка свободно распространяется	нет	нет	нет[3]	да	да	да
Имеется проверка почтового ящика	нет	нет	нет	нет	да	да
Возможность работы с большими списками параметров	есть	нет	есть	есть	есть	есть
Имеется неинтерактивный инициализационный сценарий	нет	есть	есть[4]	есть[4]	есть	есть
Имеется инициализационный сценарий не только для оболочки, в которую логируется пользователь	нет	есть	есть[4]	есть	есть	есть
Имеется возможность обойти инициализационный пользовательский инициализационный сценарий	нет	да	нет	да	нет	да
Можно определить дополнительный инициализационный сценарий	нет	нет	да	да	нет	нет
Наличие списковых переменных	нет	есть	есть	нет	есть	есть
Полное управление сигналами прерывания	есть	нет	есть	есть	нет	есть
Возможность предупредить пользователя, что он пытается перезаписать существующий файл	нет	есть	есть	есть	есть	есть
Наличие локальных переменных	нет	нет	есть	есть	нет	есть

[1] Этой возможности не было в исходной версии, но сейчас это почти стандарт.

[2] Эта возможность довольно нова, но уже имеется во многих оболочках.

[3] Эта возможность нестандартная, однако имеются заплаты (patch), которые позволяют ее добавить.

[4] Только посредством определения файла через переменную окружения $ENV.

Здесь мы коснулись только стандартных оболочек. Однако в качестве оболочек можно использовать такие интерпретаторы, как `perl` и `python`.

Оболочка bash

Специальные файлы

Оболочка `bash` использует ряд специальных файлов, которые кратко охарактеризованы в табл. 2.3.

Таблица 2.3. Специальные файлы bash

Имя файла	Значение
/etc/profile	Этот сценарий автоматически выполняется во время процедуры входа в систему, если оболочка bash указана в файле /etc/passwd
$HOME/.bash_profile	Этот сценарий автоматически выполняется во время входа в систему
$HOME/.bashrc	Этот сценарий автоматически выполняется во время запуска оболочки
$HOME/.bash_logout	Этот сценарий автоматически выполняется во время выхода из системы
$HOME/.bash_history	В этом файле содержатся команды, которые были введены с терминала пользователем во время последних сеансов работы
/etc/passwd	Из этого файла берутся имена домашних каталогов для расшифровки аббревиатур вида ~name

Метасимволы в именах файлов

В качестве параметров команд могут использоваться имена файлов. Часто необходимо указать некоторый класс имен, например, * . с, то есть все файлы, имеющие расширение с. Для указания таких классов используются специальные знаки — метасимволы. Основные метасимволы приведены в табл. 2.4.

Таблица 2.4. Метасимволы в именах файлов

Метасимвол	Значение
* (звездочка)	Любая строка (в том числе и нулевой длины)
? (вопросительный знак)	Любой символ
[abc...]	Любой из символов, заключенных в скобки; между символами может быть знак - (дефис), чтобы обозначить символьный интервал, например [a-g] обозначает символы от а до g
[!abc...]	Любые символы, кроме заключенных в скобки
~name	Каталог пользователя с именем name
~+	Текущий рабочий каталог
~-	Предыдущий рабочий каталог

Редактирование в командной строке bash

Возможности редактирования в командной строке, которые предоставляет оболочка bash, довольно разнообразны и могут удовлетворить вкусы и предпочтения самых разных

пользователей. Автор перечислил не все возможные средства редактирования командной строки, которые реализует bash, а лишь те, что являются, по мнению автора, наиболее используемыми. Кроме того, свойства редактирования зависят от версии оболочки. Здесь описана оболочка версии GNU bash, version 1.14.7(1). Возможные комбинации редактирования командной строки приведены в табл. 2.5.

Таблица 2.5. Редактирование в командной строке bash

Комбинация клавиш	Соответствующее действие
CTRL-a	Переместить курсор в начало строки
CTRL-b	Переместить курсор на один символ влево
ESC-c	Поменять регистр символа с нижнего на верхний, а курсор переместить на первый пробел справа от текущего слова
DEL	Удалить символ слева от курсора
CTRL-d	Удалить символ под курсором
CTRL-e	Переместить курсор на самый правый символ строки
CTRL-f	Переместить курсор на один символ вправо
CTRL-k	Удалить правую часть строки, начиная с символа, на который указывает курсор
CTRL-l	Очистить экран и поместить текущую команду в верхнюю строку экрана
ESC-l	Превратить символы, начиная с символа под курсором до самого правого символа в данном слове, в прописные буквы, а курсор установить на пробел справа от слова
CTRL-m	То же, что RETURN
CTRL-o	То же, что RETURN, затем отображается очередная команда из файла истории
CTRL-t	Поменять местами два символа: символ под курсором и символ слева от курсора, затем переместить курсор на один символ вправо
ESC-t	Поменять местами два слова: слово под курсором и слово слева от него
CTRL-u	Удалить часть строки слева от курсора
ESC-u	Сделать символы слова под курсором заглавными, начиная с символа, на который указывает курсор до самого правого символа в слове, а курсор установить на пробел справа от слова
CTRL-y	Вставить в текущую строку справа от курсора часть строки, удаленную ранее при помощи операции CTRL-u
ESC-.	Вставить последнее слово из предыдущей команды перед символом, на который указывает курсор

Комбинация клавиш	Соответствующее действие
CTRL-[	То же, что ESC
ESC-_	То же, что ESC-

Специальное редактирование в bash

В оболочке bash имеются специальные комбинации управляющих символов, которые помогают сократить как время на ввод имен команд, файлов, переменных окружения, так и время на поиск нужных имен, которые могут оказаться совсем не короткими. Эти средства позволяют bash «догадаться», какое имя файла или какую команду вы имеете в виду и подставить соответствующие имена в ответ на ввод описываемых ниже кодовых комбинаций. Описания этих команд приведены в табл. 2.6.

Таблица 2.6. Специальное редактирование в bash

Комбинация клавиш	Соответствующее действие
TAB	Оболочка пытается выполнить общий алгоритм подстановки текста в командной строке. Если вы попробуете ввести в пустой командной строке символ TAB, то после первого нажатия услышите звуковой сигнал, а после второго получите примерно такое сообщение: There are 2179 possibilities. Do you really wish to see them all? (y or n). Если вы ввели несколько символов команды, а после этого нажали клавишу TAB, то вы получите значительно менее длинный список возможных команд. Если команда уже введена, то вы получите список имен файлов
ESC-?	Напечатать список возможных завершений введенного текста
ESC-/	Попытаться завершить имя файла
CTRL-x/	Напечатать список возможных завершений имени файла
ESC-~	Попытаться завершить имя пользователя
CTRL-x ~	Напечатать список возможных завершений имени пользователя
ESC-$	Попытаться завершить имя переменной окружения
CTRL-X $	Напечатать список возможных завершений для имени переменной окружения
ESC-@	Попытаться завершить имя хоста
CTRL-X @	Напечатать список возможных имен хостов
ESC-!	Попытаться завершить имя команды
CTRL-x !	Напечатать список возможных завершений имени команды
ESC-TAB	Попытаться завершить имя команды, используя список ранее использованных в сеансе работы команд

Выполнение команд bash

Выполнение команд в оболочке bash производится последовательно, по мере ввода команд. Последовательные команды могут группироваться. Можно сделать так, чтобы последовательность выполнения команд зависела от результатов их выполнения. Средства управления потоком выполнения команд в bash перечислены в табл. 2.7.

Таблица 2.7. Средства управления потоком команд в bash

Способ запуска команды	Значение
command	Обычное выполнение команды command
command &	Выполнение команды command как фонового процесса
command; command1	Последовательное выполнение команд command, затем command1
(command; command1)	Запуск дополнительной bash-оболочки для выполнения двух команд как командной группы
command | command1	Программный канал: стандартный вывод команды command используется в качестве стандартного ввода в команде command1
command `command1`	Подстановка: результат выполнения команды command1 используется в качестве параметра в команде command
command && command1	Операция логическое «И»: команда command1 выполняется только в том случае, если command выполнена успешно
command || command1	Операция логическое «ИЛИ»: команда command1 выполняется только в том случае, если command выполнена неуспешно
{ command; command1 }	Выполнить последовательность команд в текущей оболочке. Следует заметить, что между фигурными скобками и командами должны быть пробелы, иначе оболочка даст информацию об ошибке

Встроенные команды bash

Таблица 2.8. Встроенные команды bash

Команда	Описание
#	Игнорировать оставшуюся часть строки. Используется для комментариев. Исключение составляет случай, когда в первой строке за знаком # следует восклицательный знак
#!	Используется для указания оболочки, которая будет интерпретировать данный сценарий. Например: #!/usr/local/bin/tcsh

Команда	Описание
:	Пустая команда. Возвращает код завершения 0. Иногда используется в качестве первой команды в сценарии, чтобы указать, что это сценарий именно для bash
.	То же, что команда source
. file [arguments]alias [name[='cmd']]	Задать name в качестве псевдонима для команды cmd. Если аргумент cmd опущен, то выводится текущее значение псевдонима name; если аргумент name также опущен, то производится вывод списка всех определенных псевдонимов. В команде может быть задано сразу несколько пар «псевдоним – команда»
bg [jobIDs]	Поместить указанное задание в фоновый режим
bind [options] bind [options] keys: function	Эта команда позволяет установить связь между комбинациями клавиш и функциями библиотеки readline или макросами. Аргумент options может принимать следующие значения: -m – После этого параметра должен быть указан используемый тип раскладки. Чаще всего используются типы emacs и vi; полный список можно получить при помощи команды help bind. -l – Вывести список функций. -v – Вывести список функций вместе с их привязками к комбинациям клавиш. -d – То же, что и -v, но список будет выведен в формате, пригодном для повторного чтения оболочкой bash. -f filename – Произвести привязку комбинаций клавиш к функциям в соответствии с файлом filename. -q function – Вывести, с какой комбинацией клавиш связана данная функция
break [n]	Выйти из самого внутреннего цикла одного из типов for, while, until или, если указан параметр n, выйти из цикла, который находится выше текущего на n-1 ступень. Кроме этого, команда используется для выхода из тела оператора select
built-in command [arguments]	Выполнить команду, которая встроена в оболочку, а не другую программу с тем же именем
case case stringin regex) commands ;; ... esac	Если string удовлетворяет регулярному выражению regex, то выполнить команды commands
cd [dir]	Если аргумент dir не задан, то команда выполняет переход в домашний (основной) каталог пользователя. Если аргумент задан, то выполняется переход в каталог dir
command [options] command [arguments]	Выполнить команду command, игнорируя функции, определенные в оболочке с помощью конструкции function name {commands}. Команда может быть либо встроенной, либо находиться в каталоге $PATH. Код завершения устанавливается равным коду, возвращаемым командой command или 127 в случае, если command не найдена. Аргумент options может принимать следующие значения: -p – произвести поиск команды command в каталогах по умолчанию, используя значение $PATH; -v – только напечатать каталог, где находится команда, не выполняя саму команду; -V – только сообщить, как выполняется по умолчанию данная команда оболочкой: как внешняя или как встроенная

продолжение

Таблица 2.8 *(продолжение)*

Команда	Описание
continue [n]	Перейти к следующей итерации в циклах типа for, while, until. Если присутствует числовой параметр, то происходит переход к началу n-1-го охватывающего цикла
declare [options] [name[=value]] typeset [options] [name[=value]]	Объявляет переменные окружения и/или присваивает им значения и/или атрибуты. Если аргумент name не задан, то печатаются имена и значения всех переменных окружения. Возможные значения options: -f – использовать только имена функций; -r – присвоить атрибут "только для чтения"; -x – пометить переменную для последующего экспорта; -i – присвоить переменной атрибут «арифметическая». Использование знака + (плюс) вместо знака - (минус) перед параметрами снимает соответствующие атрибуты. Когда эта команда используется в теле функции, то объявленные переменные будут локальными для этой функции. Код завершения команды 0, за исключением случая, в котором предпринята попытка выполнить некорректные действия, как то: попытка вывести имя несуществующей функции, попытка сменить статус переменной, для которой установлен атрибут «только для чтения»
dirs [options]	Напечатать стек имен каталогов (см. pushd/popd). Возможными значениями аргумента options могут быть: +n – вывести n-ное имя (отсчет начинается с 0); -n – вывести n-ное имя, начиная с конца; l – вывести подробную информацию
echo [options][string]	Вывести строку на стандартное устройство вывода. Вывод заканчивается символом <NL>. Если строка, которую следует вывести, не задана, то выводится только символ <NL>. Возможные значения options: -n – Не выводить в конце строки символ <NL>. -e – Разрешить интерпретацию escape-последовательностей. В этом случае полезно строку string поместить в кавычки, дабы интерпретация была произведена командой echo, а не самой оболочкой bash при разборе команды. Список допустимых escape-последовательностей: \a – звуковой сигнал; \b – возврат каретки на один символ назад(<BS>); \c – запретить вывод символа <NL> в конце строки (то же, что и параметр -n); \f – символ <FF>; \n – символ <NL>; \r – символ <CR>; \t – горизонтальная табуляция; \v – вертикальная табуляция; \\ – символ \; \nnn – символ с восьмеричным кодом nnn
enable [options] [built-in-command]	Разрешить (если аргумент options равен -a) или запретить (в случае, если аргумент options равен -n) интерпретацию встроенной в оболочку команды built-in-command. Если аргументы опущены, то будет выведен список внутренних команд оболочки, интерпретация которых разрешена. Если аргумент options имеет значение -a, а аргумент built-in-command опущен, то будет выведен список всех встроенных команд оболочки вместе

Команда	Описание
	с информацией о состоянии (enabled или disabled – разрешена или запрещена интерпретация). Команда enable удобна при определении в сценарии своих собственных функций с именами, совпадающими с именами внутренних команд оболочки bash
eval [command args...]	Выполнить команду, передав ей указанные аргументы. Код завершения команды eval устанавливается равным коду завершения команды command. Если команда не найдена, то код завершения равен 127
exec [[-] command [arguments]]	Выполнить команду command, заместив ею текущий процесс (обычно создается новый процесс). Иными словами, происходит замещение текущей оболочки. Если первый аргумент есть -, то оболочка помещает этот знак в нулевой параметр, который передается команде, так же, как это делает процедура login. Если команда не может быть выполнена по каким-либо причинам, то неинтерактивный вариант оболочки завершает свою работу (то есть в этом случае в сценарии exec вызовет завершение его работы). Однако если определена переменная окружения no_exit_on_failed_exec, то будет выведено диагностическое сообщение, сформирован ненулевой код завершения, и выполнение сценария будет продолжено. Если аргумент command опущен, то команда exec используется для перенаправления потоков ввода/вывода текущей оболочки, а ее код возврата будет равен 0. Например, чтобы заменить оболочкой tcsh текущую оболочку, выполните команду exec /bin/tcsh. Чтобы переназначить стандартный ввод на файл input_file, используйте команду exec<input_file
exit [n]	Выйти из сценария с кодом завершения n. Если параметр пропущен, то код завершения сценария будет установлен равным коду завершения последней команды внутри сценария
export [options] [VariableNames[=value]]	Экспортировать переменные окружения, то есть сделать их глобальными. Аргумент options может принимать следующие значения: -- – последующую информацию воспринимать как переменные и их значения, а не как параметры; -f – рассматривать имя глобальной переменной как имя функции; -n – отменить экспорт конкретной переменной; -p – вывести список переменных, экспортированных данной оболочкой
fc [options][first][last] fc –s [old=new] [command]	Эта команда предназначена для работы с историей команд. Она позволяет как вывести список команд, данных пользователем в данной сессии, так и отредактировать и повторно выполнить какие-либо команды из этого списка. Аргумент options может принимать следующие значения: -e – следующий параметр есть имя редактора для редактирования строки: по умолчанию имя редактора содержится в переменной окружения FCEDIT; если переменная не определена, то используется редактор vi; -l – показать сохраненный список команд; -n – не показывать

продолжение

Таблица 2.8 *(продолжение)*

Команда	Описание
	номера команд в отображаемом списке; -r – показать сохраненный список команд в обратном порядке; -s old=new – заменить подстроку old на подстроку new в команде command и выполнить эту команду повторно. Смотрите также примеры в разделе «Примеры подстановок из файла истории»
fg [jobIDs]	Перевести текущее задание или задание с номером jobIDs из режима фоновой задачи в режим задачи переднего плана
for x [in list] docommandsdone	В цикле последовательно присваивать переменной x слова из списка list. В теле цикла выполнять команды commands. Если аргумент list опущен, то предполагается использование списка позиционных параметров $@. Смотрите пример в разделе «Пример использования цикла вида for-do-done»
function name () {list}	Определить функцию с именем name. Последовательность команд оболочки list, которая составляет тело функции, может использовать значения позиционных параметров функции так же, как и в обычном сценарии ($1, $2, и т. д.).
getopts optstring name [args]	Команда используется процедурами оболочки, чтобы разделить позиционные параметры и аргументы. Обычно применяется внутри сценария для разбора параметров. Строка optstring содержит распознаваемые параметры, обозначаемые одним символом. Если за символом следует двоеточие, то данный параметр требует аргумент, который должен быть отделен от параметра одним пробелом. Например, если значение аргумента optstring есть ax:b, то возможно использование трех параметров: a, x и b, причем параметр x требует указания аргумента после него. При каждом вызове getopts помещает значение параметра в переменную окружения с именем name, инициализируя переменную, если это необходимо. Индекс следующего параметра помещается в переменную окружения $OPTIND.$OPTIND устанавливается в 1 каждый раз, когда вызывается оболочка или сценарий. Если параметр требует указания аргумента, то getopts помещает этот аргумент в переменную окружения $OPTARG. Оболочка не переустанавливает переменную $OPTIND автоматически. Иными словами, если имеет место многократный вызов getopts в одном сценарии, то необходимо побеспокоиться об установке $OPTIND в исходное значение. Смотрите примеры в разделе «Пример использования встроенной команды getopts»
hash [-r] [commands]	Произвести поиск команд, указанных в аргументе commands и запомнить результаты поиска – для каждой из указанных команд имя каталога, в котором она находится. Таким образом, при последующем использовании указанных команд уже не потребуется производить их поиск. Если аргумент commands не указан, то команда hash выводит список команд, путь к которым уже запомнен. При печати указывается, сколько раз использовалась каждая команда из списка. Если указан параметр -r, то информация об указанных командах удаляется из памяти (если опущен параметр commands, то производится удаление информации обо всех командах)

Команда	Описание
help [string]	Эта команда выводит информацию о встроенных командах оболочки, имя которых начинается на string. Например, команда help h выдаст на экран следующее: hash: hash [-r] [name ...] help: help [pattern ...] history: history [n] [[-awrn] [filename]]. При этом каждая команда сопровождается дополнительными, иногда весьма обширными пояснениями, которые в нашем примере опущены
history [lines] [options]	Эта команда выводит пронумерованный список команд, сравнительно недавно выданных пользователем. Если указан аргумент lines, то отображаются последние lines строк. Аргумент options может принимать следующие значения: -a – Оболочка bash поддерживает файл .bash_history в домашнем пользовательском каталоге. Данный параметр указывает оболочке добавить в файл .bash_history список команд, выданных в текущей сессии до команды history. -n – Добавить в список команд те, что еще не были прочитаны из файла .bash_history. -r – Использовать файл .bash_history как список выданных команд вместо текущего рабочего списка. -w – Записать в файл .bash_history текущий список выданных команд (замещая предыдущее содержание файла)
if test-cmd	Начинает условный оператор. Команда имеет следующие форматы. Простейший формат: if test-cmd then command fi Формат с оператором else: if test-cmd then command1 else command2 fi Формат с оператором elif: if test-cmd then command1 elif test-cmd2 then command2 else commmand3 fi
jobs [options][jobIDs] jobs -x command [args]	Команда выводит список всех заданий (выполняющихся или остановленных) или заданий, которые определяются параметром jobIDs. Возможные значения аргумента options: -l – в дополнение к обычной информации выводятся идентификаторы процессов; -p – выводятся только идентификаторы процессов, являющихся лидерами группы; -n – выводятся только задания, статус которых изменился с момента последнего уведомления. Если указан параметр -x, команда jobs замещает любой идентификатор задания, найденный в аргументах command, или args, соответствующим групповым идентификатором процесса и выполняет команду command, передавая ей аргументы args
kill [options] IDs	Отправка сигнала (по умолчанию SIGINT, вызывающего прекращение работы процесса), всем процессам, указанным в аргументе ID. Чтобы отправить процессу сигнал, вы должны быть владельцем процесса или привилегированным пользователем. Аргумент options может принимать следующие значения: -signal – отправить сигнал с указанным номером или именем (список сигналов можно получить при помощи команды kill –l); -- – указать, что остаток командной строки содержит аргументы, а не параметры; -l – Вывести список всех доступных сигналов; -s signal – отправить сигнал с указанным номером или именем

продолжение

Таблица 2.8 *(продолжение)*

Команда	Описание
let expressions	Выполнить целочисленные арифметические операции. Выражение expressions состоит из целых чисел, знаков арифметических действий и имен переменных окружения, которые не нуждаются в предшествующем знаке $. Выражения должны быть заключены в кавычки, если они содержат пробелы или другие специальные символы. Например: let k=k+1 или let "k = k + 1". И в том, и в другом случае значение переменной k увеличивается на 1
logout	Выход из оболочки. Команда может быть использована, если планируется выход из оболочки, в которую производился вход. В противном случае следует использовать команду exit
popd [options]	Одна из команд для работы со стеком каталогов. По умолчанию из стека изымается самый верхний каталог и производится переход в него. Возможные значения аргумента options: +n – удалить n-й каталог из стека (отсчет начинается с вершины стека, которой присваивается номер 0); -n – удалить n-й каталог из стека (отсчет начинается снизу стека с 0)
pushd directory pushd [options]	Еще одна команда для работы со стеком каталогов. В первом варианте команда pushd добавляет указанный каталог на вершину стека и делает его текущим каталогом. Во втором варианте команды pushd, если параметры опущены, то меняются местами два верхних каталога стека каталогов. Если задан аргумент +n, то n-й каталог от вершины из стека каталогов перемещается на вершину. Если задан аргумент -n, то верхним каталогом становится n-й каталог, считая от дна стека каталогов
pwd	Вывести абсолютное имя текущего каталога. Если командой set установлен параметр -p, то команда pwd не будет печатать имя символической ссылки
read [-r] var1 [var2 ...]	Производится чтение одной строки из стандартного ввода. Переменные var1, var2, ... получают значения слов из прочитанной строки (по порядку). Если переменные исчерпаны, то остаток строки записывается в последнюю переменную. В частности, если указана лишь одна переменная, то в нее будет записана вся строка. Если не указано ни одной переменной, то строка записывается в переменную окружения $REPLY. Если указан параметр -r, то обратный слэш не будет восприниматься как символ продолжения строки
readonly [options] var1 [var2 ...]	Предотвратить изменение значений переменных (установить для них атрибут «только для чтения»). Возможные значения аргумента options: -- – рассматривать остаток командной строки как аргументы, а не как параметры; -f [name(s)] – указывает функцию; -p – вывести все имена переменных с атрибутом «только для чтения»

Команда	Описание
return [n]	Эта команда используется внутри определения функции для выхода из функции с кодом завершения n. Если n опущено, то код завершения равен коду завершения команды, выполненной перед return
select name [in wordlist;] docommandsdone	Выбрать значение для имени name среди значений в wordlist. При выполнении команды на экран выводится нумерованный список значений из wordlist. Когда оператор вводит номер значения, то выполняется последовательность команд commands, затем снова производится выбор нового значения, если только последовательность commands не содержит команд break, return или exit
set [options] [arg1 arg2 ...]	Без параметров команда set печатает значения всех переменных окружения, определенных в текущей оболочке. Каждый из параметров может иметь значение «разрешить» (если он начинается с «-») или «запретить» (если он начинается с «+»). Параметры этой команды могут быть установлены при вызове оболочки bash. Аргументы присваиваются в естественном порядке: первый аргумент записывается в переменную $1, второй – в $2 и т. д. Возможные значения аргумента options: - – Аналогично -v и -x; оставшаяся часть командной строки будет рассматриваться как аргументы. -- – Используется как признак конца параметров; оставшаяся часть командной строки будет рассматриваться как аргументы. -a – Пометить переменные, которые будут автоматически экспортированы после того, как им будут присвоены значения. -b – Выводить информацию о состоянии фонового задания в момент завершения его работы (не ожидая очередного приглашения оболочки). -e – Немедленно завершить выполнение оболочки, если какая-либо команда вернула ненулевой код ошибки. -f – Запретить генерацию расширения имен файлов (установить режим noglobbing). -h – Определить расположение функциональных команд и запомнить их сразу же, как определены функции. -k – Присваивание параметров переменным окружения будет происходить вне зависимости от их местоположения в командной строке; обычно аргументы должны предшествовать имени команды. -l – Если команда for использует переменную окружения, которая уже используется в данном сценарии, то после выхода из цикла for переменная будет иметь то же значение, которое она имела до входа в цикл. В противном случае (если не установлен параметр -l) по выходе из цикла переменная, использованная в цикле for, будет иметь последнее присвоенное ей в цикле значение. -m – Установить режим мониторинга. Это разрешает управление заданиями, а фоновые задания выполняются в отдельной группе процессов. Обычно этот режим устанавливается автоматически. -n – Отладочный режим: оболочка читает команды, проверяет их, но не исполняет.

продолжение

Таблица 2.8 *(продолжение)*

Команда	Описание
	Режим удобен для проверки сценариев (команда set, естественно, должна быть внутри сценария). -o – Вывести список установленных режимов. Многие режимы могут быть установлены различными способами (не только командой set). Названия режимов: monitor – то же, что set –m; noclobber – то же, что set –C; noexec – то же, что set –n; noglob – то же, что set –f; notify – то же, что set –b; nounset – то же, что set -u; physical – то же, что set –P; posix – режим POSIX-совместимости; privileged – то же, что set –p; verbose – то же, что set –v; vi – использовать стиль редактирования vi; xtrace – то же, что set –x; -p – Режим для работы привилегированного пользователя: не обрабатывать файл $HOME/.profile. -t – Выполнение одной команды и немедленный выход. -u – При подстановках рассматривать использование неинициализированных переменных (переменных, которым не присвоено никакого значения) как ошибку. -v – Выводить на экран каждую прочитанную команду сценария. -x – Выводить на экран команды и их аргументы в тот момент, когда они выполняются. -C – Запретить перенаправление потока вывода при помощи оператора > в существующий файл. -H – По умолчанию установлено; разрешить команды ! и !!. -P – При использовании команды pwd выводить абсолютное имя текущего каталога
source file [arguments]	Прочитать и выполнить команды из файла file. При этом файл file не обязан быть исполняемым, но должен находиться в группе каталогов, указанных в переменной $PATH
suspend [-f]	Приостановка задания. Часто используется, чтобы приостановить команду su. Параметр -f заставляет приостановить работу оболочки даже в том случае, если это оболочка, которая была запущена программой login
test condition	Вычислить условие и, если условие истинно, вернуть нулевой код завершения. В противном случае вернуть ненулевой код завершения. Альтернативная форма проверки условия заключается в использовании квадратных скобок вместо ключевого слова test. Выражения, которые могут использоваться для вычисления условий, перечислены ниже. Проверка состояния файла(ов): -b file – file существует и является специальным файлом блочного устройства; -c file – file существует и является специальным файлом символьного устройства; -g file – file существует и имеет установленный бит set-group-id; -k file – file существует и имеет установленный бит sticky; -p file – file существует и является именованным программным каналом (named pipe); -r file – file существует и доступен для чтения; -s file – file существует и имеет размер в байтах больше нуля; -t n – открытый (командой open) дескриптор файла n связан

Команда	Описание
	с терминальным устройством; по умолчанию, n=1; -u file – file существует и имеет установленный бит set-user-id; -w file – file существует и доступен для записи; -x file – file существует и является исполняемым файлом; -G file – file существует и его группа является эффективной группой процесса, проверяющего условие; -L file – file существует и является символической ссылкой; -O file – file существует и принадлежит пользователю, являющемуся владельцем процесса, выполняющего проверку; -S file – file существует и является сокетом; fA –ef fB – файлы с именами fA и fB существуют и связаны между собой (другими словами, это один и тот же файл); fA –nt fB – файл с именем fA имеет более позднее время модификации, нежели файл с именем fB; fA –ot fB – файл с именем fA имеет более раннее время модификации, нежели файл с именем fB. Сравнение строк: -n sA – sA имеет ненулевую длину; -z sA – sA имеет нулевую длину; sA=sB – строка sA равна строке sB; sA!=sB – строка sA не равна строке sB; string – string не пуста. Сравнение целых чисел: n1 –eq n2 – n1 равно n2; n1 –ge n2 – n1 больше или равно n2; n1 –gt n2 – n1 больше n2; n1 –le n2 – n1 меньше или равно n2; n1 –lt n2 – n1 меньше n2; n1 –ne n2 – n1 не равно n2. Логические комбинации условий: !condition – истинно, если условие condition ложно; C1 –a C2 – истинно, если условия C1 и C2 истинны одновременно (логическое «И»); C1 –a C2 – истинно, если хотя бы одно из условий C1 или C2 истинно (логическое «ИЛИ»). Отметим, что при построении логических выражений могут использоваться более чем два условия
times	Вывести объем процессорного времени, затраченного пользователем и системой в данной сессии
trap [-l] [commands] [signals]	Выполнить команды commands, если получен любой сигнал из множества signals. Общие сигналы включают сигналы с номерами 0, 1, 2 и 15. Если commands содержит несколько команд, то они должны быть объединены в группу и разделены точкой с запятой. Если commands представляет собой пустую строку, например, (trap "" signals), тогда сигналы signals будут игнорироваться данной оболочкой. Если команды commands полностью опущены, то обработка сигналов signals будет производиться стандартным образом. Если аргументы commands и signals опущены, то будут выведены текущие установки обработки сигналов. Если задан параметр -l, то на экран будет выведен полный список сигналов. Список некоторых сигналов: 0 – выход из оболочки (обычно возникает при завершении работы с оболочкой); 1 – hangup (обычно возникает при завершении работы с оболочкой); 2 – прерывание (обычно возникает при нажатии CTRL-c); 3 – выход (quit); 4 – неверная машинная команда (illegal instruction); 5 – трассировка (trace trap); 6 – аварийное завершение (abort); 8 – исключительная ситуация

продолжение

Таблица 2.8 *(продолжение)*

Команда	Описание
	при операциях с плавающей точкой (floating point exeption); 9 – завершение (termination); 10 – определяется пользователем; 11 – неправильная адресация памяти; 12 – определяется пользователем; 13 – ситуация записи в программный канал, который не открыт на чтение; 14 – аварийный сигнал по тайм-ауту; 15 – завершение программы (обычно по команде kill); 16 – ошибка в стеке сопроцессора (coprocessor stack fault); 17 – завершение процесса-наследника (termination of childprocess); 18 – продолжение после остановки; 19 – остановить процесс (stop process); 20 – остановка вывода на терминал; 21 – фоновый процесс ожидает ввода с терминала; 22 – фоновый процесс ожидает вывода с терминала; 23 – ошибка ввода-вывода; 24 – превышение квоты на процессорное время; 25 – превышение квоты на размер файла. Несколько примеров использования trap: trap "" 2 – игнорировать сигнал 2 (прерывание); trap 2 – восстановить стандартную реакцию на сигнал 2; trap "rm -f tmp; exit" 0 1 2 – удалить файл tmp в случае получения сигналов 0, 1 или 2
type [options] commands	Вывести тип ключевых слов: является ли ключевое слово именем программы (сценария), ключевым словом, используемым оболочкой, встроенной командой оболочки, или же это неопознанная комбинация символов. Аргумент options может принимать следующие значения: -- – рассматривать оставшуюся часть командной строки как аргументы, а не как параметры; -a – вывести информацию обо всех вариантах команд commands, а не только о том, который будет вызываться по умолчанию; -p – вывести значения команд commands из внутреннего кэша оболочки; -t – определить состояние команд commands: являются ли они псевдонимами, ключевыми словами, встроенными функциями или файлами. Приведем простой пример. Предположим, что мы хотим узнать, какой тип имеют команды trap, while, LoadFarm и htpl. Для этого используем команду type trap while LoadFarm htpl. Вот результат: trap is a shell builtin while is a shell keyword LoadFarm is /home/shevel/bin/ LoadFarm type: htpl: not found Как можно видеть, оболочка сообщила, что trap есть встроенная команда, while есть ключевое слово, LoadFarm – это программа или сценарий, а htpl – не найдено
typeset	См. описание команды declare
ulimit [options] [n]	Эта команда предназначена для управления пользовательскими квотами. Если аргумент n опущен, то команда выводит информацию об имеющихся ограничениях на использование ресурсов. Если задан аргумент n, то команда устанавливает соответствующую квоту на указанный ресурс. Квоты на ресурсы могут быть «жесткими» (используется параметр -H) и «мягкими»

Команда	Описание
	(используется параметр -S). По умолчанию ulimit устанавливает обе квоты, а выводит информацию о «мягкой» квоте. Аргумент options может принимать следующие значения: -- – оставшаяся часть командной строки содержит аргументы, а не параметры; -a – вывести все имеющиеся квоты; -H – вывести «жесткие» квоты; -S – вывести «мягкие» квоты; -c – квота на соге-файлы; -d – квота на размер сегмента данных процесса; -f – квота на размер файлов, создаваемых оболочкой; -m – квота на размер резидентного набора; -n – квота на количество описателей файлов; -P – квота на размер программного канала; -s – квота на размер стека. -t – квота на количество времени процессора в секундах; -u – квота на количество процессов на пользователя; -v – квота на объем виртуальной памяти, используемый оболочкой
unset [-f] [-v] [name...]	Команда позволяет для каждого имени name удалить соответствующую переменную или функцию. Если задан параметр -v, то команда unset будет влиять только на переменные (а не функции). Если указан параметр -f, то unset будет действовать лишь на функции (а не на переменные). Если параметры не указаны, то unset сначала пытается воздействовать на переменные, а уже потом на функции. Некоторые переменные, например, $PATH или $IFS, не могут быть удалены
wait [ID]	Команда запрашивает паузу в выполнении до тех пор, пока процесс с номером ID или задание с номером ID не будут завершены. Если аргумент ID опущен, то производится ожидание завершения всех фоновых заданий. (Удобно пользоваться переменной окружения $!, которая содержит номер фонового процесса, запущенного последним.) Если не включена возможность управления заданиями, то параметр ID может обозначать только номер процесса. Например, последовательность команд $ StartPar & $ wait $! запустит в качестве фонового процесса сценарий StartPar и дождется его завершения
while condition; do commands; done	Выполнить все подстановки в последовательности команд commands и повторять выполнение этой последовательности до тех пор, пока условие condition не примет значение «ложь»

zsh

Оболочка `zsh` по своим свойствам ближе всего к оболочке Korn shell (`ksh`). С момента появления `zsh` она становится все более и более совместимой с `ksh`. Улучшились возможности редактирования командной строки, возможности определения поведения оболочки, развиваются возможности,

позволяющие пользователям, знакомым с языком C и оболочкой `csh`, чувствовать себя уверенней. Также появились некоторые полезные возможности, присущие оболочке `tcsh` (например, дополнительная «пользовательская» оболочка).

Трудно утверждать, что лишь `zsh` аккумулирует полезные свойства всех остальных оболочек, поскольку все они в той или иной степени наследуют полезные свойства более ранних оболочек. Тем не менее, можно обратить внимание на следующие преимущества `zsh` перед такими оболочками, как `csh`, `bash`, `tcsh`:

- Редактирование командной строки:
  - программируемое завершение: встроенная возможность использовать глобализацию имен файлов (compctl--g);
  - редактирование команд, состоящих из нескольких строк, как единого буфера (можно даже редактировать команды, содержащиеся в файлах!);
  - редактирование переменных (vared);
  - стек команд;
  - печать текста прямо в буфер для последующего редактирования (`print -z`);
  - выполнение не связанных (unbound) команд;
  - меню завершения;
  - расширение команд из истории внутри строки переменных.
- Исключительно мощная глобализация имен файлов (globbing):
  - рекурсивная глобализация (аналогично программе `find`);
  - задание параметров файла (размер, тип, прочее — аналогично программе `find`);
  - полные альтернативы и отрицания образцов имен файлов.
- Управление множественными перенаправлениями (более простое, нежели при помощи команды `tee`).

- Большое количество возможностей настройки.

  Расширение пути поиска программы (например, `=foo -> /usr/bin/foo`).

  Подстраиваемые сообщения (включая условные выражения).

  Именованные каталоги.

  Гибкая целочисленная арифметика.
- Манипулирование массивами (включая обратное индексирование).
- Исправление ошибок в неверно написанных словах.

Прочие оболочки

tcsh, csh

tcsh — это улучшенный вариант `csh`, которая является одной из старых оболочек, включенной почти во все варианты UNIX/Linux.

esh

До сих пор разработчики предлагают новые варианты оболочек. Среди этих вариантов — оболочка `esh`. Основным отличием этой оболочки является то, что синтаксис команд в значительной степени следует традициям языка Lisp. Это позволяет оболочке занимать очень немного оперативной памяти, но иметь при этом больше возможностей программирования, нежели у традиционных оболочек. Информацию о ней и саму оболочку `esh` можно найти по адресу http://esh.netpedia.net/.

Дополнительные примеры

В этом разделе мы приведем несколько примеров использования оболочки `bash`.

Примеры подстановок из файла истории

Вывести пять последних введенных команд:

```
$ fc -l -5
```

Вывести команды из истории команд с номера 520 по номер 530:

```
$ fc -l 520 530
```

Заменить k9q на orion в ранее выданной команде telnet, после чего выполнить команду:

```
$ fc -s k9q=orion telnet
```

С помощью редактора pico отредактировать 15-ю команду от конца списка ранее введенных команд и выполнить ее по завершении редактирования:

```
$ fc -e pico -15
```

Пример использования цикла вида for-do-done

```
$ for user in `w -h | awk 'print($1)' | sort | uniq`
> do
> finger $user
> done
```

Эта команда выводит информацию о пользователях, работающих в системе в настоящий момент.

Обратите внимание, что слово do стоит во второй строке. Если do поставить в той же строке, что и for, то перед do надо поставить точку с запятой.

Пример использования встроенной команды getopts

В качестве примера рассмотрим простой сценарий.

```
 #!/bin/bash
echo 0=$0 1=$1 2=$2 3=$3 4=$4
while getopts ax:b NAME;do
echo OPT=$NAME OPTIND=$OPTIND OPTARG=$OPTARG CC=$?
done
```

Сохраним сценарий под именем G. Попробуем вызвать это сценарий с параметрами: $ ./G -a -b -x. Вот результат:

```
$ ./G -a -b -x
0=./G 1=-a 2=-b 3=-x 4=
OPT=a OPTIND=2 OPTARG= CC=0
OPT=b OPTIND=3 OPTARG= CC=0
./G: option requires an argument - x
OPT=? OPTIND=4 OPTARG= CC=0
```

Если указать для сценария не только параметры, но и аргумент (`$ ./G -a -b -x FileName`), то получим следующее:

```
$ ./G -a -b -x FileName
0=./G 1=-a 2=-b 3=-x 4=FileName
OPT=a OPTIND=2 OPTARG= CC=0
OPT=b OPTIND=3 OPTARG= CC=0
OPT=x OPTIND=5 OPTARG=FileName CC=0
```

Вход в систему

Первоначальный вход в систему производится при помощи программы `login`, которая вызывается из программ `init` или `getty`. Пользователь может вызвать программу `login` и самостоятельно (естественно, если он уже вошел в систему перед этим):

```
$ exec login user-name [var1=vvar1 var2=vvar2 ...]
```

После имени пользователя можно присвоить значения переменным окружения, правда, не всем. К последним относятся: `SHELL`, `TERM`, `USER`, `HOME`. Если программа `login` вызывается не пользователем, а системой, то выдается приглашение, которое хранится в файле `/etc/issue`./`etc/issue`.

После ввода пароля система выводит сообщения из файла /etc/motd на экран. Полезно помнить, что запретить вывод сообщений на экран можно, разместив в домашнем каталоге $HOME файл нулевой длины с именем `.hushlogin`.

Выход из системы

Выход из системы производится при выходе из запущенной при входе в систему оболочки. Из оболочки вы может выйти, например, выполнив команду `exit`. Выйти из оболочки, в которую вы вошли по команде `login`, можно также при помощи команды `logout`. Во многих случаях для выхода из оболочки можно использовать комбинацию клавиш Ctrl-d.

Файловая система Linux

Введение

В Linux файловая система имеет иерархический характер (рис. 2.1). Дерево каталогов начинается с корневого каталога, который обозначается символом / (наклонная черта или слэш). Любой каталог может содержать файл(ы) или/и подкаталог(и). Если имя файла начинается с обозначения корневого каталога, то говорят, что задано абсолютное имя файла. Например, `/etc/passwd` является абсолютным именем. Если имя файла не начинается с косой черты, то такое имя называется относительным именем файла. Например, `passwd` является относительным именем файла. Если вам необходимо указать, что файл находится в текущем каталоге, то следует использовать обозначение `./file-name` (точка и слэш). Обозначение `../file-name` указывает, что файл с именем `file-name` находится в родительском по отношению к текущему каталоге. Можно также использовать более сложные обозначения, например, файл `../../../file-name` находится на три уровня выше текущего каталога.

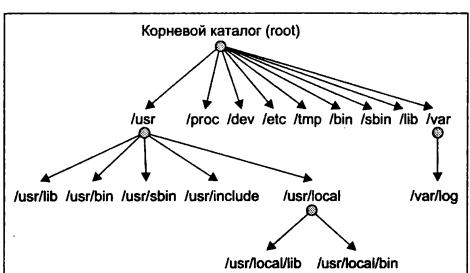


Рис. 2.1. Структура каталогов Linux

В Linux каждый пользователь имеет свой каталог, который называется основным или домашним каталогом данного пользователя. Домашний каталог для краткости обозначает

ется знаком ~~. Например, команда ls ~ выведет содержание вашего домашнего каталога. Для указания домашнего каталога другого пользователя достаточно указать его имя после тильды: команда `ls ~ivanov` выведет оглавление домашнего каталога пользователя с именем `ivanov`.

Существуют несколько типов файлов: простые (обычные) файлы, которые содержат данные; специальные файлы, описывающие физические устройства ввода/вывода; специальные файлы, которые используются для организации очередей типа FIFO или программных каналов.

Полезные сведение о свойствах файлов можно почерпнуть на страницах руководства `info file`.

Поскольку операционная система Linux является вариантом системы UNIX, то общая организация файловой системы одна и та же, хотя есть и некоторые отличия.

По состоянию на 1999 год Linux является 32-битной операционной системой. Этот факт означает, что ни при каких условиях в Linux невозможно адресовать пространство памяти большее, чем можно указать в 32-битном адресе. При этом максимальный размер одного файла в системе не может превышать 2 GB. Предполагается, что 64-битная версия Linux появится в 2001 году.

Типы файлов

Если вы выполните команду `ls -l` в любом каталоге, то, помимо имен, вы увидите атрибуты находящихся в нем файлов, например: .

```
-rw-r--r-- 1 shevelusers    2629 Aug 23 17:06
A2PS_Delegation.tex
```

Здесь самый левый символ (`-`) содержит информацию о типе файла:

`d` — каталог;

`-` — обычный файл;

`l` — символическая ссылка;

`c` — символьное устройство;

`b` — блочное устройство;

`p` — именованный программный канал (named pipe).

Не обязательно помнить наизусть все эти обозначения. Можно воспользоваться командой `file`, которая сообщит вам тип файла, например:

```
$ file /usr/lib/libjpeg.so.62
/usr/lib/libjpeg.so.62: symbolic link to
libjpeg.so.62.0.0
$ file t
t: fifo (named pipe)
```

Следующие девять символов (после символа, определяющего тип файла), задают режим доступа к файлу: левые три символа описывают режим доступа для владельца файла, средние три символа — режим доступа для пользователей, которые состоят в той же группе, что и владелец файла, а последние три символа — режим доступа для всех остальных пользователей.

Значение позиций (в тройке символов) и символов в них:

первая позиция — чтение файла, `r` — разрешено;

вторая позиция — запись в файл, `w` — разрешена;

третья позиция — исполнение для обычных файлов или просмотр для каталогов, `x` — разрешено ;

`-` — знак дефиса в любой позиции означает запрет на проведение соответствующей операции с файлом.

Отметим кратко значения остальных полей. Число, которое следует сразу за полем описания режима доступа, представляет собой количество «жестких» ссылок (hard link) на данный файл. Далее следует имя владельца файла, название группы, к которой принадлежит файл, размер файла в байтах, время создания или модификации файла, и, наконец, имя файла.

Манипуляции с файлами

Изменение владельца и группы файла, а также режима доступа

Режим доступа к файлу можно изменить с помощью команды `chmod` (change mode). Например, команда

```
$ chmod a+r t
```

означает, что для всех пользователей устанавливается разрешение читать файл с именем `t`.

Следует заметить, что пользователь сможет реально прочесть данный файл, если только он имеет права доступа к оглавлению каталога, в котором находится данный файл, а также к оглавлениям всех вышележащих каталогов вплоть до каталога `/home/username`

Для того чтобы разрешить всем пользователям просматривать оглавление каталога, используйте команду

```
$ chmod a+x catalogname
```

Для смены владельца файла или каталога используется команда `chown` (change owner).

Изменить группу, которой принадлежит файл или каталог, можно командой `chgrp` (change group).

Создание файлов и каталогов

Создание файлов устройств выходит за рамки этой книги, и мы не будем его здесь рассматривать.

Специальный файл для организации именованной очереди типа FIFO (First In First Out, первым вошел — первым вышел, иначе — именованного программного канала, named pipe) создается при помощи команды `mkfifo`. Например, по команде `mkfifo TranSport` будет создан специальный файл с именем `TranSport`, который можно использовать в качестве среды передачи информации от одного процесса к другому независимо выполняющемуся процессу:

```
$ mkfifo TransPort
$ echo Hello > TranSport &
$ cat TranSport
Hello
```

Здесь один процесс, который выполняет команду `echo Hello`, помещает данные в очередь, а другой процесс, выполняющий команду `cat TranSport`, получает их из очереди (и выводит на экран).

Создание обычных файлов может выполняться любой программой, которая записывает данные на диск. Это может быть редактор текстов, готовая к исполнению пользовательская программа и т. п. Например, команда

```
$ cp /dev/null Temp
```

создаст файл нулевой длины с именем `Temp`.

Для создания каталога используется команда `mkdir`. Например, для создания каталога с именем `T` используйте команду `mkdir T`. В созданный каталог можно перейти при помощи команды `cd T` (change directory — сменить каталог).

Уплотнение файлов и каталогов и перенос каталогов

Часто оказывается необходимым уплотнить (сжать) редко используемые текстовые файлы, чтобы они занимали меньше места или для сокращения времени передачи их по сети. Для этого используется несколько программ. Стандартными программами для уплотнения/разуплотнения данных, которые есть во всех UNIX/Linux-системах, являются программы `compress` и `uncompress`.::

Во многих случаях при уплотнении текста лучшие результаты дает программа `gzip` (и парная ей программа `gunzip`. Например, типичной задачей является уплотнение PostScript-файла: при уплотнении файла размером 1 907 497 байт программой `gzip` у меня получился файл в 580 655 байт, а при использовании программы `compress` — 737 319 байт.

Обе программы (`gzip` и `gunzip`), как правило, можно найти во всех вариантах Linux.

Копирование каталогов

Часто встречающейся задачей является перенос каталогов из одного места в другое. Каталог со множеством подкаталогов, содержащий много вложенных уровней, часто непросто или даже невозможно пофайлово перенести на новое место, например, в другую файловую систему или на другую машину. Имеется несколько способов, как решить эту проблему. Например, можно воспользоваться командой `rcp` (либо `scp` — в зависимости от настройки вашей системы):

```
$ rcp -r catalogname remotehost:newcatalog
```

Другой способ — перейти внутрь данного каталога, создать его архив при помощи программы `tar`, затем уплотнить полученный результат, а уже уплотненный файл передать по сети. Например,

```
$ cd catalogname
$ tar zcvf My.tar
$ rcp My.tar.gz remotehost:.
```

После этого в системе с именем remotehost нужно выполнить обратную операцию:

```
tar zxvf My.tar.gz
```

Удаление файлов и каталогов

Удаление файлов производится командой rm, например,

```
$ rm file-name1 file-name2 ...
```

Одной командой можно удалить несколько файлов. Если вы хотите удалить непустые каталоги, то это делается так:

```
$ rm -r catalog-name1 catalog-name2 ...
```

Так вы можете удалить сразу несколько каталогов вместе с файлами и подкаталогами внутри. Следует отметить, что команда rm -r далеко не безопасна, и ей надо пользоваться с большой осторожностью, ибо можно удалить огромное количество файлов без каких-либо предупреждений.

Для удаления пустого каталога лучше использовать команду rmdir, которая не позволит вам случайно удалить каталог, содержащий файлы.

Если требуется удалить файл с именем, которое содержит дефис, например, -C, то перед указанием имени файла следует поставить двойной дефис: rm -- -C. В противном случае команда rm попытается истолковать имя файла как параметр, что приведет к ошибке или даже уничтожению вовсе не тех данных, которые вы собирались удалить.

Операция удаления является наиболее опасной. В современных системах вы можете иметь доступ сразу к большому количеству данных, например, ко многим тысячам или десяткам тысяч файлов. Если вы удаляете одной командой много файлов, то даже незначительная ошибка может привести к потере вашего труда и труда ваших коллег за большое количество времени. Поскольку в Linux нет автоматически встроенной операции восстановления удаленных файлов, то необходимо принимать специальные меры, чтобы обезопасить себя на случай своих ошибок при удалении файлов.

Одним из хороших методов является не удаление файлов или каталогов, а перемещение их в определенный каталог

(«мусорную корзину») с помощью команды mv (move — переместить). Например,

```
$ mv Test Trash/
```

Данной командой вы перемещаете файл Test (или каталог) в каталог Trash. Таким образом, если вы «удалили» нечто ошибочно, то есть возможность вернуть удаленный объект назад. Тем не менее, время от времени вам придется чистить каталог Trash (он ведь тоже занимает место на диске!), то есть физическое удаление информации будет происходить, но в отложенном режиме. В графических интерфейсах пользователя, которые мы рассмотрим чуть позже, как правило, имеются средства для поддержания технологии отложенного удаления, пример которой приведен выше.

Вы также можете использовать команду rm с параметром -i. В этом случае перед удалением каждого файла система будет запрашивать у вас подтверждение:

```
$ rm -i /tmp/*
rm: remove `/tmp/a'? y
rm: remove `/tmp/bar'? y
rm: remove `/tmp/foo'?
```

Иерархия системных файлов

Linux имеет целый ряд каталогов специального назначения, содержимое которых обусловливает работу Linux. Наиболее важные из них перечислены в табл. 2.9.

Таблица 2.9. Важные каталоги в Linux

Имя файла	Назначение
/	Это корневой каталог. Все каталоги системы, включая каталоги, созданные пользователями, являются подкаталогами этого каталога (возможно, не первого уровня вложенности)
/bin	Этот каталог содержит программы, большинство из которых необходимы в однопользовательском системном режиме во время запуска системы (или при ее отладке). Также здесь содержится «базовый» набор основных Unix-команд, таких как cp, ls, more, tar и т. п.
/boot	Содержит файлы для загрузчика. Файлы из этого каталога, как правило, нужны только во время загрузки системы

Имя файла	Назначение
/dev	Каталог, содержащий файлы устройств. Дополнительную информацию вы найдете на man-странице mknod(1)
/etc	Каталог содержит конфигурационные файлы, которые являются локальными для данной системы. Крупные прикладные пакеты, например, X Window System, могут иметь свой каталог для конфигурационных файлов. Обычно он размещается ниже по файловой иерархии. Конфигурационные файлы, общие для группы машин, помещаются, как правило, в каталог /usr/etc. Тем не менее, часть конфигурационных файлов может находиться как в /etc, так и в /usr/etc
/etc/skel	Когда в системе создается новая учетная запись, то файлы из этого каталога копируются во вновь созданный домашний каталог пользователя
/etc/X11	Каталог для конфигурационных файлов системы X11
/home	Обычно в этом каталоге находятся домашние каталоги пользователей
/lib	Этот каталог должен содержать разделяемые библиотеки, которые необходимы для загрузки операционной системы и для выполнения команд в корневой файловой системе
/mnt	В этом каталоге обычно содержатся точки монтирования для временно смонтированных файловых систем
/proc	Это точка монтирования для файловой системы proc, которая обеспечивает информацию о выполняющихся процессах, ядре, оборудовании вычислительной установки и т. д. Это псевдофайловая система. Подробности о ней можно найти в man-странице proc(5)
/sbin	Этот каталог подобен каталогу /bin, но содержит в основном программы, необходимые для загрузки операционной системы или для администрирования системы
/tmp	Каталог для временных файлов. В любой момент пользовательские файлы из этого каталога можно удалить без большого ущерба для остальных пользователей. Однако не стоит удалять файлы из этого каталога, если вам не стало ясно, что конкретный файл или группа файлов мешает продолжению продуктивной работы на машине
/usr	Этот каталог обычно содержит библиотеки или данные, предназначенные лишь для чтения. Каталог /usr на вашей машине может быть смонтирован на других Linux-машинах посредством NFS

продолжение

Таблица 2.9 *(продолжение)*

Имя файла	Назначение
`/usr/X11R6`	Файлы, относящиеся к системе X Window версии 11, ревизия 6.
`/usr/X11R6/bin`	Готовые к исполнению программы системы X Window версии 11, ревизия 6
`/usr/X11R6/lib`	Файлы и библиотеки, связанные с системой X Window
`/usr/X11R6/lib/X11`	Каталог содержит различные файлы, необходимые для работы системы X Window
`/usr/X11R6/include/X11`	Содержит файлы заголовков, необходимые для компилирования программ, которые используют библиотеки системы X Window
`/usr/bin`	Готовые к исполнению программы, которые часто вызывают обычные пользователи
`/usr/bin/X11`	Обычное место для расположения готовых к исполнению программ из X Window в Linux. Часто это символическая ссылка на `/usr/X11R6/bin`
`/usr/dict`	Этот каталог содержит файлы со словарным запасом для программ проверки корректности написания слов
`/usr/etc`	Здесь содержатся конфигурационные файлы для группы машин или для всей организации. Однако команды и программы должны смотреть в каталог `/etc`, в котором должны быть ссылки на файлы в каталоге `/usr/etc`
`/usr/include`	Файлы заголовков для программ на языке C
`/usr/include/X11`	Файлы заголовков, необходимые для компиляции программ на языке C, использующих систему X Window. Обычно это символическая ссылка на каталог `/usr/X11R6/include/X11`
`/usr/include/asm`	Файлы заголовков, содержащие определения ряда функций ассемблера
`/usr/include/linux`	Этот каталог содержит файлы, которые могут меняться от версии к версии Linux. Часто это имя является символической ссылкой к каталогу `/usr/src/linux/include/linux`. Отсюда Linux получает специфическую информацию для системы. Но, например, в Debian Linux это реализовано по-иному
`/usr/include/g++`	Каталог содержит include-файлы для использования в GNU C++
`/usr/lib`	В данном каталоге содержатся объектные библиотеки подпрограмм, динамические библиотеки, некоторые готовые к исполнению программы, которые не вызываются непосредственно. Сложные

Имя файла	Назначение
`/usr/lib/X11`	Обычное место для помещения файлов, связанных с X Window, а также конфигурационных файлов самой системы X Window. В Linux это обычно символическая ссылка на каталог `/usr/X11R6/lib/X11`
`/usr/lib/gcc-lib`	Содержит готовые к исполнению программы и файлы заголовков для компилятора GNU C (gcc)
`/usr/lib/groff`	Файлы для системы форматирования текстов `groff`
`/usr/lib/uucp`	Файлы для системы UUCP
`/usr/lib/zoneinfo`	Файлы для определения временной зоны (часового пояса). Смотрите также страницы руководства `named-xfer(8)`, `tzfile(5)`, `tzselect(8)`, `zdump(8)`, `zic(8)`
`/usr/local`	Обычно здесь помещают программы и подкаталоги, которые являются локальными для данной машины
`/usr/local/bin`	Обычно здесь помещают готовые к исполнению программы, которые являются локальными для данной машины
`/usr/local/doc`	Локальная документация обычно помещается здесь
`/usr/local/etc`	Конфигурационные файлы для локально установленных программ
`/usr/local/lib`	Библиотеки и файлы для локально установленных программ и систем
`/usr/local/info`	Страницы описаний, которые просматриваются посредством программы `info`, для локально установленных программ
`/usr/local/man`	Страницы описаний, которые просматриваются посредством программы `man`, для локально установленных программ
`/usr/local/sbin`	Локальные программы системного администратора
`/usr/local/src`	Исходные тексты программ локального значения, установленных на данной машине
`/usr/man`	Страницы руководств
`/usr/man/<locale>/man[1-9]`	Эти каталоги содержат страницы руководств в исходной форме. Системы, которые используют один язык и один кодовый набор, могут не использовать подстроку `<locale>`
`/usr/sbin`	Этот каталог содержит готовые к исполнению программы для системного администрирования, которые не используются во время загрузки
`/usr/src`	Исходные тексты различных частей Linux

продолжение

Таблица 2.9 *(продолжение)*

Имя файла	Назначение
`/usr/src/linux`	Исходные тексты для ядра Linux
`/usr/tmp`	Еще одно место для хранения временных файлов. Это символическая ссылка на каталог `/var/tmp`. Не рекомендуется использовать этот каталог для хранения важной информации
`/var`	Этот каталог содержит файлы, которые могут сильно изменяться по размеру, например, протоколы (иначе, журналы или log-файлы), временные файлы и т. д.
`/var/adm`	Этот каталог должен быть символической ссылкой на каталог `/var/log`
`/var/backups`	Этот каталог используется, чтобы хранить резервную копию важных системных файлов
`/var/catman/cat[1-9]`	Этот каталог используется, чтобы хранить уже сформированные страницы руководств в соответствии с номером главы
`/var/lock`	Здесь содержатся управляющие файлы системы, которые используются для резервирования использования тех или иных ресурсов системы
`/var/log`	Различные файлы журналов (log-файлы)
`/var/preserve`	Здесь редактор `vi` сохраняет сессии редактирования при ненормальном завершении выполнения. Таким образом, тексты могут быть восстановлены позже
`/var/run`	Переменные файлы времени выполнения различных программ. Они содержат идентификаторы процессов (PIDs) и записывают текущую информацию (utmp). Файлы в этом каталоге обычно очищаются во время загрузки системы
`/var/spool`	Файлы различных программ, поставленные в очередь на обслуживание к разным системам: на печать, на передачу по электронной почте
`/var/spool/at`	Файлы заданий, запущенных посредством команды `at`
`/var/spool/cron`	Файлы системы cron
`/var/spool/lpd`	Файлы, ожидающие вывода на печать
`/var/spool/mail`	Пользовательские почтовые ящики
`/var/spool/news`	Файлы системы news
`/var/spool/uucp`	Файлы системы uucp
`/var/tmp`	Временные файлы

Важные конфигурационные файлы Linux

Конфигурационные файлы в Linux — это текстовые файлы с описанием параметров и свойств всей системы или отдельных ее компонентов. В разделе «Конфигурационные файлы в каталоге /etc» приведен список важных конфигурационных файлов, которые находятся в системном каталоге `/etc`.

Иногда такого типа файлы называют инициализационными файлами. Это те же конфигурационные файлы, но они обычно прочитываются во время старта программы, которая их использует. Довольно часто они располагаются в домашнем каталоге пользователя и имеют вид `.имяrc`, где `имя` — имя программы, для которой предназначен инициализационный файл. Этот файл прочитывается программой лишь тогда, когда она вызывается тем пользователем, которому принадлежит домашний каталог.

В то же время имеется каталог `/etc`, в котором обычно находятся главные конфигурационные и инициализационные файлы. Эти файлы прочитываются программой вне зависимости от того, какой пользователь вызвал конкретную программу. Конфигурационные файлы из каталога `/etc` прочитываются раньше файлов из домашнего каталога пользователя. Иными словами, значения параметров, установленные пользователем в своем каталоге, оказываются более приоритетными по сравнению со значениями, установленными файлами в каталоге `/etc`.

Для прикладных программ, например, `pine` или `a2ps`, в `/etc` помещаются, как правило, конфигурационные файлы общие для всех пользователей. Однако любой пользователь может установить свою персональную конфигурацию, которая описывается соответствующим файлом в каталоге, задаваемом переменной `$HOME`.

Если вам потребовалось поэкспериментировать с содержимым конфигурационного файла, не делайте этого с файлом в каталоге `/etc`! Перед внесением любых изменений в любой конфигурационный файл в каталоге `/etc` убедитесь, что вы понимаете, что будет происходить в системе или как будет себя вести прикладная программа.

Автор не планировал подробно и детально перечислять все без исключения мыслимые конфигурационные файлы и их форматы. Такая задача представляется малопродуктивной из-за относительно частого изменения состава программ и систем, а также форматов самих файлов. Основная идея состоит в том, чтобы дать сведения лишь об основных конфигурационных файлах.

Конфигурационные файлы в каталоге /etc

amd.conf

Файл `/etc/amd.conf` используется программой автоматического монтирования файловых систем `amd`. Программа `amd` является демоном, который автоматически монтирует требуемую файловую систему при обращении к ней и размонтирует ее, если она некоторое время не используется. Подробную информацию вы можете найти на man-страницах `man adm.conf`, `man amd`, `man amq`.

apcupsd.conf

Файл `/etc/apcupsd.conf` представляет собой конфигурационный файл для демона, который следит за состоянием вашего устройства бесперебойного питания — УБП (UPS).

dosemu.conf

Файл `/etc/dosemu.conf` представляет собой конфигурационный файл для эмулятора MS DOS `dosemu`. Подробности следует искать в документации, сопровождающей пакет `dosemu`.

gated.conf

Файл `/etc/gated.conf` представляет собой конфигурационные данные для работы демона `gated`, который работает с протоколами маршрутизации RIP, BGP, EGP, HELLO, OSPF. Дальнейшую информацию вы можете найти на странице `man gated`.

gpm-root.conf

Файл `/etc/gpm-root.conf` представляет собой файл настройки программы `gpm-root`, которая предназначена для

управления мышкой в консоли Linux. Более подробная информация может быть получена при помощи команд `man gpm-root.conf`, `man gpm`, `info gpm`.

group

Файл `group` представляет собой системный файл, содержащий описания групп, к которым принадлежат пользователи. Каждая строка файла имеет следующий формат:

```
group_name:passwd:GID:user_list
```

Более подробная информация может быть получена на man-страницах `group` и `groupadd`.

host.conf

Файл `/etc/host.conf` представляет собой конфигурационные данные для комплекта программ `resolver`, обеспечивающих работу службы определения имен (DNS). Среди прочего, в файле должен быть описан порядок нахождения (разрешения) имени. Подробности могут быть найдены на странице руководства `host.conf`.

inetd.conf

Файл `/etc/inetd.conf` представляет собой конфигурационные данные для программы `inetd`. Файл `/etc/inetd.conf` часто называют базой данных интернет-служб, которые могут использоваться в системе. Все строки этого файла должны содержать следующие поля:

- имя службы (описано в файле `/etc/services`);
- тип сокета;
- имя протокола;
- `wait/nowait` (вариант `wait` имеет смысл только для дейтаграммных сокетов `wait/nowait`, остальные службы должны иметь в этом поле значение `nowait`);
- `пользователь[.группа]`;
- имя программы сервера;
- аргументы программы сервера.

Примеры:

```
login stream  tcp nowait  root/usr/sbin/tcpd
in.rlogind
talk  dgram   udp waitroot/usr/sbin/tcpd  in.talkd
ftp   stream  tcp nowait  root/usr/sbin/tcpd  in.ftpd
-l -a
telnet stream  tcp nowait  root/usr/sbin/tcpd
in.telnetd
auth  stream  tcp nowait  nobody  /usr/sbin/in.identd
in.identd
```

Дополнительную информацию вы найдете на страницах `man inetd.conf` и `man inetd`.

issue

Файл `/etc/issue` представляет собой текст сообщения, которое будет выводиться на экран перед приглашением ввести имя пользователя (`login:`). Смотрите также страницу описания `man issue`.

ld.so.conf

Файл `/etc/ld.so.conf` представляет собой конфигурационные данные для компоновщика-загрузчика и содержит список каталогов, в которых следует искать библиотеки. Подробную информацию вы найдете на страницах `man ld.so` и `man ldconfig`.

По файлу `/etc/ld.so.conf` строится файл `/etc/ld.so.cache`, который содержит упорядоченный список библиотек, найденных в каталогах, имена которых перечислены в файле `/etc/ld.so.conf`. Файл `/etc/ld.so.cache` можно просмотреть при помощи команды `ldconfig -p`.

lilo.conf

Файл `/etc/lilo.conf` — конфигурационный файл для программы `lilo`, которая используется для загрузки системы. Подробная информация может быть найдена на страницах `man lilo.conf` и `man lilo`.

logrotate.conf

Файл `/etc/logrotate.conf` — конфигурационный файл программы `logrotate`, которая предназначена для упроще-

ния администрирования файлов журнала (log-файл), создаваемых различными программами. Утилита `logrotate` позволяет автоматически начать файл журналов заново, удалить, сжать файл журнала, отправить его по электронной почте. Эти действия можно выполнять ежедневно, еженедельно, ежемесячно или при достижении файлом определенного размера.

Обычно `logrotate` запускается при помощи планировщика заданий `cron`. Смотрите также `man logrotate`.

manpath.config

Файл `/etc/man.config` является конфигурационным файлом для программы `man`. Дальнейшую информацию вы найдете в руководстве по системе `man`.

mtools.conf

Файл `/etc/mtools.conf` является конфигурационным файлом для пакета программ `mtools`, предназначенного для обеспечения доступа к дискетам, записанным в формате MS DOS. Как обычно, подробности могут быть почерпнуты в `man mtools`.

named.conf

Файл `/etc/named.conf` является конфигурационным файлом для демона `named`. Подробности в `man named`.

nscd.conf

Файл `/etc/nscd.conf` является конфигурационным файлом для службы Name Service Cache (служба кэширования имен). Программа `nscd` кэширует имена, которые получены посредством этой службы. Кэширование существенно сокращает время ожидания при повторных запросах NIS+ и DNS. Структура файла описана в документации к самой программе.

nsswitch.conf

Файл `/etc/nsswitch.conf` является конфигурационным файлом для целой группы библиотечных программ, которые связаны с получением имен узлов и использованием

этих имен локально (обычно в пределах вашей организации). Дополнительная информация в `man nsswitch`.

ntp.conf

Файл `/etc/ntp.conf` является конфигурационным файлом для средств, реализующих Network Time Protocol. Обмениваясь сообщениями в соответствии с данным протоколом, хосты могут синхронизировать время между собой. Смотрите также `man ntp`.

paper.config

Файл `/etc/paper.config` описывает форматы бумаги, используемые принтерами.

syslog.conf

В файле `/etc/syslog.conf` содержится конфигурационная информация для демона `syslogd`, который обеспечивает запись диагностических сообщений системы. Более подробная информация о протоколировании системных сообщений содержится в `man syslog.conf`, `man syslogd`, `man sysklogd`.

passwd

Файл `/etc/passwd` содержит список пользователей компьютера вместе с дополнительной информацией. Подробности содержатся в `man 5 passwd`.

pwdb.conf

Файл `/etc/pwdb.conf` является конфигурационным файлом для библиотеки `libpwdb`. Библиотека состоит из набора функций, предназначенных для работы с различными файлами, содержащими информацию о пользователях системы (`passwd`, `shadow`, `group` и т. п). Библиотека представляет единые средства доступа к этой информации.

shadow

Файл `/etc/shadow` содержит зашифрованные пароли пользователей вместе с дополнительной информацией. Подробности содержатся в `man 5 shadow`.

services

Файл `/etc/services` содержит конфигурационную информацию о видах сетевых служб на данной машине: названия, номера портов, типы протоколов. Каждая строка файла имеет вид:

```
service-name   port/protocol   [aliases ...]
```

Подробности вы найдете на странице `man services`.

hosts

В файле `/etc/hosts` содержится информация об адресах и именах каких-либо узлов (как правило, в вашей собственной сети). Эта информация может оказаться очень полезной в момент загрузки, когда еще недоступна служба `named`.

hosts.equiv

Файл `/etc/hosts.equiv` содержит информацию о хостах и пользователях, которым разрешено (или запрещено) использовать r-команды (`rlogin`, `rsh`, `rcp`) для доступа к данной системе. Информация о структуре файла приводится на странице `man hosts.equiv`.

motd

В файле `/etc/motd` содержится системное сообщение, которое выдается на экран при входе в систему. Дополнительная информация — на странице `man motd`.

hosts.access, hosts.deny

Файлы `/etc/hosts.access` и `/etc/hosts.deny` содержат описания правил доступа к вашему узлу. Язык описания правил позволяет весьма гибко указать, кому и каким службам разрешен доступ, отсекая тем самым все попытки несанкционированного подключения.

Смотрите `man hosts _access` и `man hosts_options`.

at.allow, at.deny

Файлы `/etc/at.allow` и `/etc/at.deny` содержат правила, определяющие, кто из пользователей может использовать планировщик заданий `at`, а кто — нет.

updatedb.conf

Файл `/etc/updatedb.conf` содержит конфигурационную информацию для утилиты updatedb, используемой для построения базы данных для быстрого поиска файлов при помощи команды locate.

Для получения дополнительной информации смотрите содержимое самого файла `/etc/updatedb.conf`, а также страницу man updatedb.

ypserv.conf

Файл `/etc/ypserv.conf` является конфигурационным файлом, в котором определяется ряд переменных для службы `ypserv`, обеспечивающей работу системы NIS. Дополнительная информация — на страницах `man ypserv. conf` и `man ypserv`.

yp.conf

Файл `/etc/yp.conf` является конфигурационным файлом для программы `ypbind`, которая является частью системы NIS. Дополнительная информация — на странице `man ypbind`.

Языки программирования в Linux

Языки программирования (а точнее их реализации) условно разделяются на компиляторы и интерпретаторы. Таким способом подчеркивается характер порождаемого кода, который получается в результате процесса трансляции.

Если исходный текст языка сначала проверяется и транслируется целиком, а затем порождаемый код является в основном кодом машинного языка, то такой исходный язык является компилятором. Если строка исходного языка исполняется после прочтения во время выполнения программы, то такой язык — интерпретатор. Оба типа имеют свои преимущества в определенных ситуациях. Так, компиляторы обычно дают более эффективные по времени выполнения программы. Однако стоимость компилятора и сопутствующих программ высока, а иногда затраты на генерацию кода сложной программы (компиляция, сборка), оказыва-

ются неприемлемыми, особенно в том случае, если программу приходиться часто изменять.

Интерпретаторы удобны тем, что строки языка исполняются по мере их прочтения. Таким образом, сразу видны недочеты или ошибки при написании программы. Кроме того, во многих случаях неважно, будет программа выполнена за 1 секунду или за 0.0001 секунды, например, если это программа организации диалога с оператором.

Естественно, что между этими типами имеется масса промежуточных вариантов. Примерами компиляторов являются C, FORTRAN, другие процедурно ориентированные языки. Примерами интерпретаторов являются APL (впервые появился на машинах IBM), ИНФ (на машинах типа «Днепр»), REXX (IBM 370), bash, tcsh, Perl, Python, Java.

Со временем языки программирования стали все более ближе к интерпретаторам. Это заметно упрощает транслятор (что, в свою очередь, уменьшает вероятность ошибок), увеличивает возможности переноса языка на машины с другой архитектурой, сокращает весь цикл приготовления программы: от разработки до получения программного продукта.

FORTRAN

FORTRAN — один из самых старых языков программирования, рассчитанный исключительно на компиляцию. Первый вариант транслятора с языка Фортран появился в 1952 году. Название происходит от английского «formula translator» — транслятор формул, FORTRAN — весьма простой и широко распространенный язык для научных и инженерных вычислений. Реализации этого языка существуют для всех без исключения аппаратных платформ. На Фортране написано много программ, которые используются до сих пор. Высказывается мнение, что стоимость используемых программ на FORTRAN примерно равна или превышает стоимость компьютеров, предназначенных для вычислений.

На FORTRAN легко писать программы связанные с расчетами, поскольку язык специально ориентирован для этих целей. Программировать на FORTRAN обработку текста

намного сложнее, нежели создавать вычислительные программы.

Информацию о FORTRAN можно найти на странице http://www.fortran.com/fortran/metcalf.html.

f2c

На современных рабочих станциях часто не используют официальные трансляторы с языка Фортран, а пользуются бесплатными конвертерами из FORTRAN в C. Конвертер `f2c` (FORTRAN to C) является одним из них.

g77

Компиляторы C и FORTRAN интегрированы в версии GNU. Программа `g77` вызывает `gcc` с возможностями распознавания текстов, написанных на FORTRAN (а именно — на диалекте ANSI FORTRAN 77, который часто называется просто F77).

Компилятор `gcc` обрабатывает вводные файлы в несколько (от одной до четырех) последовательных стадий: макрообработка, компиляция, ассемблирование и сборка. Полное описание продукта `g77` можно найти в документации по GNU FORTRAN. Вы также можете найти сведения о `g77` при помощи команд `info g77` и `man g77`.

Исходные файлы программ на языке F77 обычно имеют суффиксы `.f` или `.for`; для файлов, которые будут обрабатываться препроцессором `cpp`, используют суффикс `.F` или `.fpp`.

Следует обратить внимание, что в том случае, когда исходный текст, написанный на Фортране, не полностью соответствует стандарту ANSI FORTRAN 77, то при компиляции посредством `g77`, сборке, а также при выполнении готовой программы могут возникать проблемы.

Реализация FORTRAN-90

Имеется также своеобразная реализация языка FORTRAN-90 — VAST/f90. Этот конвертер перекодирует текст FORTRAN-90 в текст FORTRAN-77. Сведения о конвертере можно найти на сервере http://www.psrv.com.

C

Реализации языка C имеются на всех аппаратных платформах и на всех операционных платформах. Это язык разработан в 70-х годах. Система UNIX написана с использованием C. Многие воспринимают C как машинный язык. Это не так, хотя он находится ближе к машинному коду, чем, например, FORTRAN или Pascal. На C можно писать любые программы, включая программы обработки и анализа текста.

По языку C имеется обширная литература как в электронном, так и в бумажном виде. В группе новостей comp.answers один или два раза в месяц рассылается FAQ по языку C.

Также обратите внимание на архив FAQ http://www.faqs.org. Среди Web-страниц можно отметить http://www.lysator.liu.se/c/ или http://www.strath.ac.uk/CC/Courses/CCourse/CCourse.html. На русском языке описание языка C с примерами можно найти как во многих традиционных книгах, так и в Интернете: http://cclib.nsu.ru/projects/gnudocs/texts/bogatir/book.html.

C++

Язык C++ является объектно-ориентированным языком. В основе синтаксиса C++ положен синтаксис языка C с включением объектно-ориентированных операций. В C++ очень важно, какой набор библиотек классов используется, так как основой программирования на C++ является использование уже написанных классов. Различные библиотеки описываются в FAQ (см. например, http://www.trumphurst.com/cpplibs1.html). Полезно также обратить внимание на группу новостей comp.lang.c++.

На C++ можно писать программы любого назначения, в том числе и программы для анализа и преобразования текста.

Perl

Одним из важных языковых инструментов является язык Perl (Practical Extraction and Report Language – язык практического извлечения информации и формирования отчетов). Это богатый язык, ориентированный в первую очередь

на программистов, системных администраторов и вообще любых лиц, которым необходимо быстро извлечь какую-то полезную информацию из операционной или файловой системы и произвести небольшие вычисления. Удобен для анализа и преобразования текста.

Язык Perl богаче многих мощных средств преобразования текстов в UNIX, например, `awk` (раздел «Подсистема сканирования, анализа и обработки текстов awk») или `sed` (раздел «Потоковый редактор текста sed»). Для того чтобы перекодировать в Perl программы, составленные на языке `awk`, используется конвертер `a2p`. Конвертер `s2p` используется для перекодировки программ на `sed` в Perl-программы.

На Perl можно составлять сценарии так же, как на `tcsh` или `bash`. Perl активно используется для составления CGI-сценариев. CGI

С использованием Perl написана масса разнообразных приложений. Perl свободно доступен на множестве сайтов, например, http://www.perl.com или http://www.gnu.org/software/perl/perl.html. Может оказаться полезным FAQ http://cpan.perl.org/doc/FAQs/.

Tcl/TkTcl/Tk – еще один свободно распространяемый интерпретатор, который часто используется для написания сценариев и организации диалога с интенсивным использованием графических возможностей. Название Tcl является аббревиатурой Tool Command Language (инструментальный командный язык). Tk обычно означает Tool Kit (инструментальный набор).

Как правило, Tcl/Tk является частью поставки Linux, но может быть получен и отдельно, например, с сайта ftp://ftp.funet.fi/pub/languages/tcl. «Домашний» сайт Tcl/Tk – http://www.scriptics.com.

Язык Tcl/Tk весьма развит и используется в сотнях приложений. Следует заметить, что Tcl и Tk часто хранятся отдельно, поскольку это вообще-то отдельные продукты. Tk представляет собой набор команд и описателей для создания и манипулирования с X-виджетами. Интерпретатор, который интерпретирует Tcl/Tk, обычно называется `wish`. При вызове этого интерпретатора появляется строчное

приглашение (обычно %) и графическое окно. Есть возможность вызвать лишь интерпретатор команд Tcl, который работает в текстовом режиме. Его также можно использовать как обычную оболочку вместо, например, bash. Пример короткой тестовой программы на Tcl приведен ниже.

```
#!/usr/bin/tclsh puts stdout "";
puts stdout "This is test program for Tcl";
set x 4; set y 19;
set Result [expr sqrt($x * $x + $y * $y)];
puts stdout Result=$Result ; \endverbatim
```

При выполнении сценарий печатает пустую строку, затем

```
This is test program for Tcl
Result=19.4164878389
```

Python

Python является одним из объектно-ориентированных интерпретаторов. Python обычно входит в состав поставки Linux или может быть свободно получен с сайта http://www.python.org.

Python имеет средства для описания высокоуровневых структур данных. Имеется разработанный интерфейс для расширения Python при помощи библиотек, написанных на C и C++, а также при помощи других языков программирования. Графический интерфейс для Python реализуется посредством пакета Tk из Tcl/Tk (есть и другие возможности организации графического интерфейса, например, при помощи библиотеки `Gtk`).

С использованием Python удобно анализировать и преобразовывать текст любого вида.

Во многих случаях выбор интерпретатора является просто волей случая. Если вы уже участвуете в крупном проекте, то, как правило, там уже утвердились те или иные соглашения на используемые языки и даже на способы написания программ при помощи уже выбранного языка.

Java

Java является наиболее свежим и современным интерпретатором. Сейчас Java — реальный претендент на роль межплатформенного средства для разработчиков, то есть име-

ется большая уверенность, что если программная система написана на языке Java, то она будет работать на всех вычислительных платформах. С другой стороны, Java — высокоуровневый объектно-ориентированный язык программирования.

Java — неплохая платформа для реализации алгоритмов анализа и обработки текста.

Прочие языки

Ada

Язык Ada появился в конце 60-х. Благодаря хорошо проработанной стандартизации интерфейсов и учету конкретных свойств вычислительного оборудования, язык Ada широко используется для программирования бортового вычислительного оборудования, а также различного вида контроллеров.

Одной из реализаций Ada является система GNAT. Информация о GNAT может быть получена на страницах http://www.gnu.org/software/gnat/gnat.html.

Pascal и конвертер p2c

Конвертер `p2c` представляет собой средство для трансляции (перекодировки) исходных текстов программ на языке Pascal в программы на языке C. Вводом может служить любой из следующих диалектов языка Pascal: HP Pascal, Turbo/UCSD Pascal, DEC VAX Pascal, Oregon Software Pascal/2, Macintosh Programmer's Workshop Pascal, Sun/Berkeley Pascal. Также поддерживается синтаксис языка Modula-2.

Большинство программ на Pascal транслируются в C и не требуют дальнейших усилий по модификации, чтобы сделать программу работоспособной. Хотя, в ряде случаев, могут печататься предупреждения, что в отдельных точках требуется ручная коррекция исходного текста.

Pascal является языком общего назначения, поэтому подходит и для разработки программ анализа и преобразования текста.

Prolog

Сокращение Prolog означает «Programming in Logic» (программирование в логике). Prolog, как предполагается, освобождает программиста от массы компьютерных деталей, чтобы он мог сосредоточиться на логике решаемой задачи. Транслятор можно найти во многих местах, например, ftp://swi.psy.uva.nl/pub/SWI-Prolog/ или ftp://clement.info.umoncton.ca/BinProlog/UNCOMPRESSED/bin. Дополнительную информацию о языке Prolog можно найти на страницах http://aisun0.ai.uga.edu/~jae/ai-lang.html.

Prolog используется в работах по искусственному интеллекту и в других столь же нетривиальных областях применения.

Lisp

Lisp — это язык обработки списков. Он часто используется в системах искусственного интеллекта. Имеется множество различных реализаций и диалектов Lisp. Подробную информацию о языке можно найти, например, на страницах http://www.elwood.com/alu/table/contents.htm или ftp://sunsite.unc.edu/pub/Linux/devel/lang/lisp/. Одна из реализаций Lisp является частью известного редактора текста `emacs` (раздел «emacs, xemacs»).

Lisp удобен для написания весьма сложных программ анализа и преобразования текста.

3. Редакторы текста

В Linux широко используются около десятка редакторов текста. Каждый редактор имеет свои особенности, и в каждом отдельном случае может оказаться полезным тот или другой, в зависимости от реального контекста использования, включая психологические предпочтения пользователя.

Редакторы текстов разделяются также и по степени сложности выполняемых операций. Как правило, более богатый по возможностям редактор требует большего количества различных ресурсов: оперативной и дисковой памяти и даже времени процессора. Последнее может быть и звучит необычно в применении к процессу редактирования текста, однако может оказаться весьма важным обстоятельством. Дело в том, что мощные редакторы, такие как vim, emacs, во многих случаях при выполнении нетривиальных операций могут обращаться к внешним программам или последовательности программ, на запуск которых вполне может требоваться немало процессорного времени. Особенно это касается emacs.

Возникает вопрос: «А каким же редактором следует пользоваться?» Ответ заключается в том, что используемый редактор должен соответствовать вашим потребностям. Так, если 90% обрабатываемых текстов — это конфигурационные файлы в несколько десятков строк, то смело пользуйтесь простейшими редакторами, например, pico. А если вам приходится редактировать большие и сложные тексты, то следует рассмотреть использование emacs или vim.

Заметим, что стиль использования относительно простых редакторов и наиболее «продвинутых» заметно отличаются. Так, для простых редакторов характерен режим, когда вы вызываете редактор для ввода или коррекции текста, а по окончании операций редактирования вы выходите из редактора. Когда понадобится редактировать другой файл, то

редактор текста вызывается снова, и т. д. При использовании «продвинутых» редакторов, которые могут выполнять массу команд, экономя ваш труд, эффективнее применять другой стиль работы, при котором редактор вызывается лишь однажды и в дальнейшем все операции с текстами, оглавлениями, вызовом других программ производится в среде редактора.

Почти все редакторы имеют описания в форматах `man` и/или `info`. Чтобы вывести описание на печать, проще всего воспользоваться программой `a2ps`, например

```
$ man vi | a2ps -1 --catman
```

По этой команде на печать будет выведено описание редактора `vi` (см. раздел «Подсистема печати текста a2ps»).

В данной главе предлагается краткий обзор нескольких популярных редакторов текста, используемых в Linux. Чуть подробнее рассматриваются наиболее важные редакторы: `vim`, `emacs` и потоковый редактор `sed`.

vi и vim

vi — один из старейших и наиболее мощных экранных редакторов. Он отлично работает даже на аппаратных терминалах типа VT-100 или XTerm. Обычно `vi` является частью поставки Linux.

По этому редактору текстов довольно часто рассылаются тексты с «FAQ» (Frequently Asked Questions, или ЧАсто задаваемые ВОпрОсы — «ЧАВО») в группе новостей comp.answers. Архив группы новостей может быть найден по адресу http://www.faqs.org.

Как сам список FAQ по редактору vi, так и родственная информация могут быть найдены на перечисленных ниже узлах:

- ftp://alf.uib.no/pub/vi
- ftp://ftp.uwp.edu/pub/vi
- ftp://ftp.uu.net/pub/text-processing/vi
- ftp://ftp.cc.monash.edu.au/pub/vi
- ftp://ftp.s.u-tokyo.ac.jp/misc/vi-archive

Существует улучшенный вариант редактора vi под названием vim (Vi IMproved — улучшенный vi). Редактор vim полностью эмулирует все команды редактора vi и в то же время имеет расширенный набор параметров при вызове и множество дополнительных возможностей:

- многоуровневый процесс «undo»;
- использование нескольких окон и буферов редактирования;
- редактирование командной строки;
- механизм дополнения имени файла, если вы напечатали только начало имени;
- встроенная справка (для доступа к ней следует ввести в запущенном редакторе команду `:help`).

Далее мы будем обсуждать только редактор vim.

vim отличается весьма высокой скоростью работы, легко редактируются файлы размером в пару сотен тысяч строк. При этом имеется возможность работать как в текстовом режиме, так и в графическом. В обоих режимах поддерживается многооконная работа.

vim имеет весьма богатый набор команд для просмотра и редактирования текстов любого вида, включая двоичные файлы. Редактор особенно удобен при работе с большим количеством текстовых файлов, которые необходимо корректировать согласованно. При этом файлы могут быть расположены как в одном, так и в нескольких каталогах. Для реализации этих возможностей имеется несколько способов: специальные таблицы *тегов* (теговских файлов, см. раздел «Использование таблицы тегов»), использование механизма меток, которые могут быть установлены в любом месте любого файла и к которым позже можно просто перейти посредством ввода имени метки. Имеется возможность использовать теговские таблицы, построенные для редактора emacs.

vim имеет интерфейсы для обращения к различным внешним программам и системам, в том числе SNiFF+, cscope и другим, а также к интерпретаторам Perl, Python, tcl.

Редактор имеет встроенный язык для описания поведения редактора в зависимости от различных меняющихся условий. На этом языке можно писать сценарии и позже использовать их в своей работе (см. раздел «Сценарии использования редактора vim».

vim может автоматически настраиваться на редактирование файлов определенных типов, например, исходных текстов программ, различных архивов и т. д. При этом редактор имеет большой набор параметров (около 100), которые позволяют значительно изменять возможности по преобразованию текстов, включая автоматический сдвиг строк, форматирование комментариев в программах, нетривиальные преобразования текста как командами редактора, так и путем обращения к внешним программным фильтрам и многое другое.

Поскольку обсуждаемый редактор является одним из наиболее мощных средств редактирования наравне с emacs, то в него встроено полное описание команд и возможностей. Страницы описания могут быть получены путем ввода команды редактора `:help`. Полезно взглянуть на таблицу filesdes на с. filesdes, которая содержит список файлов со страницами документов, доступных в редакторе. Vim может быть использован для редактирования или первоначального ввода текстового файла на любом языке (включая русский, фарси и другие языки).

Наконец, редактор постоянно улучшается, поэтому последние изменения смотрите на сайте http://www.vim.org/.

Технические особенности редактирования текста в vim

Редактирование может начинаться вводом команды:

```
vim
```

При этом вызывается редактор vim и никакие файлы не открываются. Теперь можно посмотреть страницы документации по редактору посредством команды редактора `help` или начать редактирование файла, набрав команду:

```
:edit filename
```

где `filename` есть имя файла, который вы предполагаете редактировать. По этой команде, текст из файла с именем `filename` будет прочитан во внутренний буфер редактора. Если редактируется несколько файлов, то и внутренних буферов может быть несколько. Количество и тип буферов, используемых в данный момент, можно получить командой:

```
:buffers
```

Количество редактируемых файлов часто не совпадает с числом буферов. Какие файлы редактировались ранее и какие находятся в работе, сейчас легко посмотреть командой:

```
:files.
```

Кроме этого, редактор поддерживает несколько специальных таблиц, которые помогают в быстром ориентировании в большом количестве файлов. К таким таблицам, например, относится таблица скачков (перемещений) курсора. Редактор ведет протокол перемещений курсора, которые производятся командами редактора. Позже вы можете воспользоваться данной информацией, чтобы вернуться в прежнее место. Эту таблицу легко посмотреть командой:

```
jumps
```

Аналогичным образом ведется таблица меток, которыми вы можете отмечать определенные точки в тексте. Таблица меток просматривается командой:

```
:marks
```

vim ведет таблицу (стек) тегов, которые вы использовали в своей работе. Редактор позволяет вам перемещаться по стеку использованных тегов. Стек тегов отображается командой:

```
:tags
```

В редакторе имеются специальные области внутренней памяти, которые называются регистрами. Вы можете использовать регистры как механизм временного хранения частей текста или команд редактора. Подробнее концепция регистров обсуждается в разделе «Регистры».

Как любой «солидный» редактор, vim имеет команды как для выполнения простейших операций редактирования, так и для более сложных преобразований текста. Использова-

ние простых команд не представляет труда. Например, вы начинаете редактировать существующий файл с именем `test`:

```
vim test
```

После прочтения файла во внутренний буфер вы можете перейти в конец файла, введя команду:

```
G
```

Чтобы добавить новую строку к файлу, вы можете ввести команду:

```
o
```

тем самым редактор перейдет в режим ввода, и вы можете вводить сколько угодно строк текста. Завершение ввода производится нажатием одной из комбинаций клавиш `Esc` или `Ctrl+c` или `Ctrl+[`.

Удаление одной строки производится командой

```
dd
```

Другие команды удаления элементов текста приведены в табл. 3.1.

Существует еще много десятков команд для выполнения простого редактирования и перемещения курсора по тексту. Однако для более сложных преобразований текста требуются и более сложные средства.

Элементы текста, с которыми работает vim

Редактор vim может оперировать с различными элементами текста: *символ*, *слово*, *предложение*, *параграф*, *раздел*. Здесь под термином *оперировать* мы будем иметь в виду перемещение курсора, пропуская определенное количество заданных объектов текста, удаление объектов и т. д.

Слово представляет собой последовательность символов, которые не являются разделителями слов. Слова могут разделяться пробелами, знаками табуляции TAB, символами конца строки EOL.

Предложение или фраза (sentence) определяется как слово или группа слов, за которой следует символ . (точка), или символ ! (восклицательный знак), или символ ? (вопросительный знак), а затем через любое количество закрываю-

щих символов ограничителей),], ", ' следует конец строки, табуляция или пробел. Конец параграфа или раздела являются одновременно и концом предложения.

Параграф (paragraph) начинается после каждой пустой строки, а также в местах, где имеются пары управляющих символов макроопределений параграфа (в дальнейшем, макросам), которые устанавливаются параметром `paragraphs`. По умолчанию, `paragraphs=IPLPPPQPP LIpplpipbp`. Это соответствует управляющим макросам `.IP` и `.LP`, которые должны стоять в первой колонке.

Раздел (section) начинается после символа <FF> (Form Feed) в первой колонке и в местах макросов разделов, которые определяются парами символов в параметре `sections`.

Команды редактора] и [преобразуются к виду { и }, если они находятся в первой колонке. Это удобно для поиска, например, функции в языке С. Более детальная информация может быть получена посредством команды `:h section`.

Команды редактирования с использованием элементов текста

Ниже перечислены основные операции удаления текста, которые производятся, когда vim находится в обычном (нормальном) режиме.

Таблица 3.1. Команды удаления текста vim

Команда	Действие
dl или x	Удалить символ, на который указывает курсор
diw	Удалить слово
daw	Удалить слово вместе с последующим пробелом
dd	Удалить строку
dis	Удалить фразу
das	Удалить фразу вместе с обрамлением
dib	Удалить внутренний блок ( )
dab	Удалить блок ( ) вместе с обрамлением
dip	Удалить внутреннюю часть параграфа
dap	Удалить параграф вместе с обрамлением
diB	Удалить внутренний блок { }
daB &	Удалить блок {} вместе с обрамлением

имы работы редактора

ктор vim функционирует в нескольких основных режи- (полезно посмотреть `:help vim-modes`).

омандный, или *обычный* (normal), режим, в котором роизводится ввод команд редактора. Именно в этом ежиме находится редактор сразу после старта.

ежим ввода *командной строки* (Cmdline). В этот режим едактор переводится из обычного режима вводом слеующих символов:

- `:` (двоеточие), после которого может следовать любая команда, например `:help` или `:quit`;
- `/` (наклонная черта) или `?` (знак вопроса), которые начинают операции поиска регулярных выражений;
- `:!` (двоеточие и восклицательный знак), после которых следует фильтр, или любая команда Linux, например, `:!date`.

режим *ввода* текста (insert) вы можете перевести редак- ор из обычного режима несколькими способами, напри- ер, введя символ `i`. При этом внизу экрана появляется адпись:

```
-INSERT --
```

ежим ввода завершается нажатием клавиши `Esc` или дной из комбинаций: `Ctrl+[` или `Ctrl+c`. По заверше- ии ввода текста с клавиатуры редактор переходит в *ко- андный* режим.

вод текста может производиться не только с клавиату- ы, но и из файла, как, например

```
at File | vim -
```

изуальный (visual) режим. Сразу после перехода в этот ежим любое перемещение курсора выделяет текст, про- едший под курсором. Когда вы отметили желаемую об- асть текста, вы можете выполнить какие-то операции с ыделенным текстом. В визуальный режим можно пе- йти из *обычного* несколькими способами:

- вводом символа `v`, что означает начало *визуального* посимвольного режима, то есть последующие операции выполняются над отмеченными цепочками символов;
- вводом символа `V`, что означает начало *визуального* построчного режима, то есть последующие операции выполняются над отмеченными строками;
- вводом символа `Ctrl+V`, что означает начало *визуального* поблочного режима, то есть последующие операции выполняются над отмеченным блоком.

После выполнения команды редактор выходит из *визуального* режима. Просто отменить режим можно повторно введя символ `v` (или `V`, или `Ctr+V`, соответственно), а также нажав клавиши `Ctrl+c`. В табл. 3.2 приведены возможные переходы между обычным и различными визуальными режимами.

- Режим *выбора* (select) является вариантом *визуального* режима. Он вводится нажатием клавиш `Ctrl+g` в *визуальном* режиме. При этом внизу экрана появляется надпись:

  ```
  -- SELECT--
  ```

  Дополнительную информацию можно получить, почитав `:help select-mode`.

Имеются дополнительные режимы, которые удобно использовать в частных случаях.

- Режим *замены* (replace mode) представляет собой специальный случай режима *ввода*. Переход в этот режим осуществляется командой `R` из *обычного* режима или нажатием клавиши `Insert` в режиме *ввода*. В отличие от обычного режима ввода, символы печатаются поверх существующих. По умолчанию внизу экрана появится надпись:

  ```
  -- REPLACE --
  ```

- Ввод команд в режиме *ввода* (insert command mode) — может быть осуществлен нажатием клавиш `Ctrl+o`. При этом редактор перейдет в *обычный* режим, но только для выполнения одной команды. Как только команда будет выполнена, редактор вернется в режим *ввода* текста.

Таблица 3.2. Таблица перехода в визуальном режиме

Исходное состояние	Новый режим после ввода символа: v	V	Ctr+v
Обычный	Визуальный посимвольный	Визуальный построчный	Визуальный поблочный
Визуальный посимвольный	Обычный	Визуальный построчный	Визуальный поблочный
Визуальный построчный	Визуальный посимвольный	Обычный	Визуальный поблочный
Визуальный поблочный	Визуальный посимвольный	Визуальный построчный	Обычный

Редактирование текста производится в том месте, где стоит курсор. Перемещение курсора может производиться различными клавишами и командами, как описывается в разделе curmoving.

Редактор *понимает* сокращения имен команд. Если вы переведете редактор в режим ввода командной строки, то клавишами перемещения курсора можно посмотреть *историю*, то есть ранее выполнявшиеся команды.

Если вы не понимаете по каким-то причинам, в каком режиме находится редактор, то вы всегда сможете перевести его в *обычный* режим, дважды нажав `Esc`. Если вы находитесь в *строчном командном* режиме (*Ex*), то следует ввести команду редактора:

```
:visual
```

Далее приведена табл. 3.3 переходов из одного режима в другой.

Вызов редактора

Формат вызова редактора:

```
vim [options] [file ...]
vim [options] -
vim [options] -t tag
vim [options] -q errorfile
ex
view
gvim | gview
rvim | rview | rgvim | rgview
```

Таблица 3.3. Команды редактора vim для перехода из режима в режим

	В обычный режим	В визуальный режим	В режим выбора	В режим ввода	В режим замены	В режим командной строки	В строчный командный режим
Из обычного режима		v V Ctrl+V	[1]	[2]	R	: / ? !	Q
Из визуального режима	[3]		Ctrl+G	c C	нет	:	нет
Из режима выбора	[4]	Ctrl+O Ctrl+G		[5]	нет	:	нет
Из режима ввода	Esc	нет	нет		Insert	нет	нет
Из режима замены	Esc	нет	нет	Insert		нет	нет
Из режима командной строки	[6]	нет	нет	:start	нет		нет
Из строчного командного режима	:vi	нет	нет	нет	нет	нет	

[1] Переход из *обычного* (normal) режима в режим *выбора* (select).

[2] Переход в режим *ввода* (insert) из *обычного* (normal) режима происходит после ввода команд i, I, a, A, o, O, c, C, s, S.

[3] Переход из *визуального* режима (visual) в *обычный* (normal) режим происходит после ввода команды, не связанной с перемещением курсора, а также после нажатия Esc, v, V, Ctrl+c или Ctrl+C.

[4] Переход из режима *выбора* (select) в *обычный* режим (normal) происходит при использовании команд перемещения курсора.

[5] Переход из режима *выбора* (select) в режим *ввода* (insert) происходит посредством ввода обычного изображаемого символа.

[6] Переход из *строчно-командного* режима (Cmd-line) в *обычный* режим (normal) происходит после нажатия клавиш Enter (или <NL>), а также Ctrl+U или Esc.

Редактор vim ведет себя по-разному в зависимости от имени, под которым он был вызван.

`vim` — обычный режим работы.

`ex` — работа в режиме *ex*.

`view` просмотр файла в режиме *только чтение*. Эквивалентно использованию `vim -R`.

- `gvim | gview` — использование версии с GUI (графическим интерфейсом). То же самое можно получить с помощью `vim -g`.
- `rvim | rview | rgvim | rgview` — почти то же самое, что и предыдущее, но с рядом ограничений: главным образом, нельзя запускать команды оболочки из редактора, а также приостановить редактор. То же самое можно получить с использованием `vim -Z`.

Таблица 3.4. Параметры при вызове редактора vim

Параметр	Значение
file1 file2 file3... (список имен файлов)	Файл с именем, которое находится в списке первым слева, будет прочитан в буфер редактора vim и станет текущим файлом. Курсор будет указывать на первую строку буфера. Вы сможете перейти к следующему по списку файлу, дав команду :next
- (минус)	Файл для редактирования читается с устройства стандартного ввода. При этом команды считываются с устройства stderr (стандартное устройство для вывода диагностики об ошибках), которое должно быть назначено терминалу (tty). Этот параметр позволяет использовать vim в конце цепочек программных каналов, например: ls -ltaR | grep MyCatalogs | vim -
-t tag	Файл для редактирования, а также начальная позиция курсора зависят от значения поля tag. Значение tag ищется в специальном файле тегов. Ассоциированный с тегом tag файл станет текущим файлом, а курсор будет установлен в позицию, которая также ассоциирована тегу tag. Это свойство удобно для редактирования программ, например, на языке C. Тогда в качестве тегов могут использоваться имена функций и других синтаксических элементов. В этом случае, имея заранее заготовленный файл ассоциаций с именем tags или TAGS, вызывая редактор с именем конкретной функции, например vim -t Mul, редактор сразу считает в буфер модуль, содержащий функцию с именем Mul, а курсор будет установлен на начало функции. Теговский файл готовится с помощью программы ctags или etags (раздел «Программы построения таблицы тегов»)

продолжение

Таблица 3.4 *(продолжение)*

Параметр	Значение
-q [errorfile]	Включить режим быстрой коррекции. Прочитывается файл с именем errorfile и отображается первая ошибка. Если файл errorfile опущен, то будет использовано имя по умолчанию: errors.vim. Последующие ошибки могут быть получены посредством команды редактора :cn. На детали можно взглянуть посредством :help quick-fix.
+[num]	Будет прочитан первый файл, и курсор будет установлен на строке с номером num. Если значение num опущено, то курсор будет установлен на последней строке файла
+/pat	Будет прочитан первый файл, и курсор будет установлен на строке, которая удовлетворяет выражению pat. Подробнее можно посмотреть в редакторе :help search-pattern
+command -c command	Команда редактора command будет выполнена после прочтения первого файла. Выражение command интерпретируется как команда редактора ex, который является составной частью vim. Если команда command содержит пробелы, то она должна быть заключена в кавычки. Например, +"set number" FileName Замечание. Можно использовать до 10 команд + или -c
-b	Будут установлены несколько внутренних параметров, которые позволят редактировать двоичные или исполняемые файлы
-C	Установить режим совместимости (comatible) с редактором vi
-e	Запустить vim в режиме ex
-f	Используется главным образом для версий GUI. Редактор не будет порождать новых оболочек для выполнения команд. Этот параметр должен использоваться, если редактор вызывается другой программой (сценарием), которая ожидает завершения сессии редактирования
-F	Запустить редактор с поддержкой редактирования текстов на фарси, то есть справа налево, и с раскладкой клавиатуры для фарси
-g	Если редактор скомпилирован с поддержкой GUI, то он будет запущен с поддержкой GUI. Если редактор не был скомпилирован с поддержкой GUI, то он выдаст диагностику и закончит работу
-h	Выдать краткие пояснения к списку параметров, используемых в командной строке, и закончить работу
-H	Запустить редактор в режиме редактирования справа налево для раскладки клавиатуры Hebrew
-L	То же самое, что параметр -r
-l	Устанавливает режим Lisp (смотрите также описание параметра lisp)
-m	Выключается возможность модификации файлов. Иными словами запись файлов невозможна

Параметр	Значение
-N	Режим несовместимости с vi. Редактор vim начинает вести себя «лучше», но несколько иначе, чем старый редактор vi
-n	Не используется файл подкачки. Может оказаться удобным для работы на очень медленных устройствах вторичной памяти, например, дискетах. Восстановление после сбоя невозможно. Такой режим может быть также установлен посредством команды :set uc=0. Отменить такой режим можно, например, командой :set uc=200
-o[N]	Открыть N окон. Если N опущено, то открыть одно окно для каждого файла
-R	Установить режим только чтение
-r -r filename	Восстановить состояние редактируемого файла после аварийного завершения сессии редактирования. Файл восстанавливается из файла подкачки, который имеет имя filename.swp, где filename есть имя редактируемого файла. Если filename опущено, то редактор vim выдает информацию о всех swap-файлах.
-s	Запустить редактор vim в «молчаливом» режиме, то есть с минимумом диагностических сообщений. Используется для запуска ex в пакетном режиме
-s scriptin	Читается файл с именем scriptin. Символы из этого файла интерпретируются как введенные с клавиатуры. То же самое можно сделать командой редактора :source! scriptin
-T terminal	Этот параметр предназначен для сообщения редактору типа терминала, который вы используете. Параметр нужен только тогда, когда тип терминала не установился автоматически. Можно использовать лишь те типы терминалов, которые имеются в файлах termcap или/и terminfo
-u vimrc	Использовать инициализационный файл с именем vimrc. При этом все другие виды инициализации опускаются. Этот же параметр можно использовать, чтобы обойти любые виды инициализации: -u NONE. Детали можно посмотреть посредством команды редактора :help initialization
-U gvimrc	Использовать инициализационный файл с именем gvimrc для инициализации режима GUI. При этом все другие виды инициализации режима GUI опускаются. Этот же параметр можно использовать, чтобы обойти любые виды инициализации режима GUI: -U NONE. Детали можно посмотреть командой редактора vim :help gui-init
-V	«Разговорчивый» режим (verbose). Сообщаются имена файлов, которые были использованы для инициализации, и другая информация
-v	Запустить vim в режиме vi

продолжение

Таблица 3.4 *(продолжение)*

Параметр	Значение
-w scriptout	Все, что вы вводите с клавиатуры, будет копироваться в файл с именем scriptout до тех пор, пока вы не выйдете из редактора. Основное назначение данной возможности – подготовить текст для того, чтобы использовать его в команде редактора vim :source! или вместе с параметром -s. Повторное использование того же имени файла приведет к дописыванию в конец файла. Смотрите также страницы документов по vim :h complex-repeat
-W scriptout	То же самое, что и параметр -w, но при повторном использовании того же имени файла он замещается
-Z	Режим работы с ограничениями (то же самое, что и вызов редактора посредством rvim)
-- (два минуса подряд)	Означает, что список параметров завершен; далее могут следовать лишь имена файлов
--version	Печатается версия вашего редактора vim и много другой полезной информации

Получение описания команд

Для получения полезной информации в редакторе vim следует ввести команду `:help` (двоеточие и затем слово `help`) или просто h). Например, введя `:h ZZ`, вы увидите пояснение:

```
Записать текущий файл, если он был модифицирован,
и выйти из редактора (то же самое, что команда редактора
:x). Если для данного файла было открыто несколько окон,
то файл, если был модифицирован, записывается на диск
и одно окно закрывается.
```

Часто описание содержит термины, которые имеют свое описание. Например, вы можете увидеть:

```
1. Vim arguments                    |vim-arguments|
2. Vim on the Amiga                 |starting-amiga|
3. Initialization                   |initialization|
```

Если установить курсор между двумя вертикальными чертами, к примеру, на слове `initialization` и нажать `Ctrl+]`, то произойдет переход к описанию термина `initialization`. Если позже вы захотите вернуться к первоначальному описанию, то следует нажать `Ctrl+t`.

При работе с командой help, как и со всякой другой командой редактора, можно использовать автоматическое завер-

шение команд. Например, набрав `:h cm`, вы можете нажать `<TAB>` или `Ctrl+d` чтобы увидеть один из вариантов завершений. Нажав снова `<TAB>` или `Ctrl+d`, вы увидите очередной вариант завершения. Детали механизма завершения можно почерпнуть из описания `:h ins-completion`. Перед началом использования редактора полезно посмотреть `:help starting`.

Алфавитный список основных команд редактора

Редактор имеет большой список команд (несколько сотен). Поскольку vim фактически состоит из нескольких редакторов: ex, ed и собственно визуального редактора, то команды также несколько различаются. Вначале мы рассмотрим список команд визуального (экранного) редактора. Эти команды выполняются в *обычном* режиме редактора (не в режиме *ввода* или других).

Полный список команд редактора можно посмотреть, набрав:

```
:h index.txt
```

Общий формат команд экранного редактора очень прост:

`nCommand` — `n` — это счетчик числа повторений команды `Command` (см. описание команды `[0-9]`).

Например:

`n|` — установить курсор в позицию `n` в текущей строке.

Таблица 3.5. Список команд редактора vim (счетчик n в командах опущен)

Команда	Действие
Ctrl+a	Добавить число n к числу в тексте, на которое указывает курсор. Десятичным значениям может предшествовать знак. Если значение параметра nrformats равно hex или octal, то можно использовать шестнадцатеричные и восьмеричные числа без знака. Смотрите также Ctrl+x
Ctrl+b	Пропустить (прокрутить) n экранов назад
Ctrl+c	Прервать текущую операцию поиска
Ctrl+d	Пропустить n строк вниз (по умолчанию половина экрана)
Ctrl+e	Пропустить n строк вверх
Ctrl+f	Пропустить n экранов вперед
Ctrl+g	Показать имя текущего файла и позицию курсора

продолжение ↩

Таблица 3.5 *(продолжение)*

Команда	Действие
TAB	Перейти к позиции курсора, записанной ранее в n-й строке в таблице переходов. Содержание таблицы переходов можно посмотреть командой :jumps
Ctrl+i	То же что и TAB
Ctrl+l	Очистить экран и вывести информацию заново
Enter	Переместить курсор на n строк и поместить его в первую слева позицию
Ctrl+m	То же, что Enter
Ctrl+n	То же, самое что j
Ctrl+o	Установить курсор в позицию, которую он имел n строк ранее в таблице переходов
Ctrl+p	Переместить курсор на n строк вверх
Ctrl+q	Используется для управления выводом на экран (возобновить вывод)
Ctrl+r	Снова внести изменения, которые были отменены командой u (отмена)
Ctrl+s	Используется для управления выводом на экран (остановить вывод)
Ctrl+u	Переместить курсор на n строк вверх (по умолчанию на половину экрана)
Ctrl+v	Включить визуальный поблочный режим
Ctrl+w	Выбрать команды управления окнами
Ctrl+x	Вычесть из числа под курсором значение n. Смотрите также Ctrl+a
Ctrl+y	Переместить курсор на n строк вниз
Ctrl+z	Приостановить редактор (или запустить новую оболочку)
Ctrl+^	Переключиться на редактирование файла с номером n в таблице файлов (предполагается, что вы редактируете сразу несколько файлов). Если n перед символом отсутствует, то производится переключение на редактирование предыдущего файла. Если параметр autowrite был в значении on, то при переключении будут сохранены изменения в файлах, если они имели место
!motionFil	Отфильтровать строки, начиная с текущей, до той, которая указывается курсором. Здесь motion означает команду перемещения курсора, а Fil – любая команда Linux. После восклицательного знака должна следовать команда перемещения курсора. Команда выполняется в следующем порядке. Указанные курсором строки передаются на ввод фильтра и удаляются из файла, а на их место помещается вывод команды Fil. Например, если вы введете ! j, то есть восклицательный знак, затем пробел, далее j (команда перемещения курсора на одну строку вниз), то внизу появится надпись :.,+1! Если теперь ввести date, то вместо двух строк, на которые указывает курсор, вы получите одну с текущей датой, то есть вывод программы date. А если ввести date>/dev/null, то будут просто удалены две строки. Если требуется отсортировать файл, то следует установить курсор в первую строку командой :1. Затем ввести !G, после чего внизу появится надпись :.,$! затем ввести команду sort

Команда	Действие
!!Fil	Отфильтровать n строк. Команда выполняется в следующем порядке. Указанное число строк, начиная с текущей, передаются на ввод фильтра и удаляются из файла, а на их место помещается вывод команды Fil. Например, вы вводите 5!! При этом внизу появится надпись :.,.+4!. Введя команду tac (утилита в Linux), вы измените порядок строк в текущем буфере на обратный
" (кавычки)	Вводят имя регистра для последующей команды удаления (d, x, c, s или y) или put. В качестве имени регистра используется один символ, причем только: a-zA-Z0-9.%#:- Регистры с именами % # : . Могут использоваться только для put. Посмотреть содержание регистров можно командой редактора :reg. Подробнее смотрите раздел «Регистры»
#	Искать в обратном направлении n-е включение слова, на которое указывает курсор
$	Переместить курсор в конец n-й строки (отсчитывая от текущей)
%	Искать в текущей строке любого вида скобки: круглые, квадратные или фигурные. Если курсор уже указывает на скобку, то при вводе % производится поиск соответствующей парной скобки. Например, если курсор указывает на), то будет найдена ближайшая скобка ((если она существует) и т. д.
n%	Установить курсор в позицию, которая соответствует n% от начала текста
&	Повторить предыдущую команду редактора :s
`{a-zA-Z0-9} (апостроф и одна буква)	Переместить курсор на метку в тексте с указанным именем (имя состоит только из одной буквы или цифры). Например, `k – перейти к метке k, которая была введена ранее командой редактора :mark k
`` (два апострофа)	Установить курсор в позицию, в которой он был до любой последней команды, изменяющей положение курсора
`<	Переместить курсор на первый символ–не пробел в строке, где начинается (или начиналась ранее) выделенная область в текущем буфере
`>	Переместить курсор на первый символ–не пробел в строке, где заканчивается (или заканчивалась ранее) выделенная область в текущем буфере
`[	Переместить курсор на первый символ–не пробел в первой строке, где производились последние изменения, или в начало напечатанного текста
`]	Переместить курсор в конец последней измененной строки или в конец введенного текста
(	Переместить курсор на n фраз назад
)	Переместить курсор на n фраз вперед
*	Искать в прямом направлении n-е включение в тексте слова, на которое указывает курсор
+	Переместить курсор на n строк (по умолчанию на 1) вперед
-	Переместить курсор на n строк (по умолчанию на 1) назад

продолжение

Таблица 3.5 *(продолжение)*

Команда	Действие
. (точка)	Повторить последние изменения n раз (по умолчанию 1 раз)
/patternEnter	Искать вперед n-е включение pattern
/Enter	Повторить предыдущий поиск вперед
0	Установить курсор в первую слева позицию в текущей строке
[1-9]	Ввести число (счетчик) перед командой
:	Начать ввод команды редактора Ex
[0-9]:	Начать ввод команды редактора Ex с числового параметра
<motion	Сдвинуть n строк влево на shiftwidth позиций
<<	Сдвинуть n строк влево на shiftwidth позиций
>motion	Сдвинуть n строк вправо на shiftwidth позиций
>>	Сдвинуть n строк вправо на shiftwidth позиций
?pattern Enter	Искать назад n-е включение pattern
? Enter	Повторить поиск в обратном направлении
@[a-z]	Выполнить n раз содержимое именованного регистра [a-z]
@:	Повторить предыдущую команду редактора n раз
@@[a-z]	Повторить команду @[a-z] n раз
A	Ввести текст после конца строки n раз
B	Переместить курсор на n слов назад
["x]C (кавычки)	Заменить данные, начиная с текущей позиции, до конца строки и следующие n-1 строку. Одновременно записать удаляемый текст в регистр x
["x]D (кавычки)	Удалить данные, начиная с текущей позиции курсора до конца строки и следующие n-1 строку. Одновременно записать удаляемый текст в регистр x
E	Переместить курсор в конец n-го слова
Fc	Переместить курсор влево в позицию, в которой символ с встречается n-й раз
G	Переместить курсор в строку с номером n (по умолчание – в последнюю строку в файле)
H	Переместить курсор n-ю строку от верха экрана. (В левый верхний угол по умолчанию)
I	Вставить текст n раз до первого символа в строке
J	Объединить n строк в одну строку (по умолчанию 2 строки)
K	Выполнить команду, которая хранится в параметре с именем keywordprg. В качестве параметра команды будет использовано слово, которое находится под курсором. По умолчанию значение keywordprg=man
L	Переместить курсор на строку n от нижней части экрана. По умолчанию – курсор в левый нижний угол экрана

Команда	Действие
M	Переместить курсор в середину экрана
N	Повторить команду поиска / или ? в противоположном направлении
O	Начать ввод в новой строке над текущей строкой.
["x]P	Поместить текст из регистра x в строки перед позицией курсора n раз (по умолчанию 1).
Q	Перейти в режим Ex
R	Ввести режим замены символов, то есть при вводе будут замещаться имеющиеся символы
["x]S	Удалить n строк из буфера x и начать ввод текста. Это синоним для команд ^cc или 0cc
Tc	Переместить курсор в позицию после n-й встречи символа c при перемещении по строке влево
U	Вернуть строку в исходное состояние, удалив все изменения, которые имели место (undo)
V	Начать визуальный построчный режим
W	Переместить курсор на n слов вправо
["x]X	Удалить n символов до курсора (в регистре x)
["x]Y	Скопировать n строк в буфер x (синоним для команды yy)
ZZ	Записать на диск текущий файл, если он был модифицирован, и выйти из редактора
ZQ	Выйти из редактора без всяких действий и условий
]c	Группа команд закрывающей квадратной скобки (см. табл. 3.6.)
[c	Группа команд открывающей квадратной скобки (см. табл. 3.6.)
^ (шляпка)	Переместить курсор в первую позицию текущей строки
_ (подчеркивание)	Переместить курсор в первую позицию строки, которая находится ниже текущей на n-1 строк
a	Начать ввод текста сразу после символа, на который указывает курсор
b	Переместить курсор на n слов назад
["x]cmotion	Удалить текст [и поместить его в регистр x], начиная с текущего положения курсора, до позиции, куда курсор попадет после команды перемещения курсора motion, и начать ввод. Иными словами, происходит замена текста в соответствии с основной командой c.
["x]cc	Удалить n строк [и поместить их в регистр x] и начать ввод текста. Если параметр autoindent имеет значение on, то не сдвигать первую строку
["x]dmotion	Удалить текст, начиная с текущей позиции курсора, до позиции, в которую курсор установлен командой перемещения курсора motion [и поместить удаляемый текст в регистр x]
["x]dd	Удалить n строк, начиная с текущей [и поместить их в регистр x]

продолжение ↩

Таблица 3.5 *(продолжение)*

Команда	Действие
e	Переместить курсор вперед до конца n-го слова, не обращая внимания на границы строк
fc	Переместить курсор вправо до n-го включения символа с в текущей строке
gc	Расширенный список команд
h	Переместить курсор на n позиций влево
i	Вставить текст до курсора n раз
j	Переместить курсор на n строк вниз
k	Переместить на n строк вверх
l	Переместить курсор на n символов вправо
m{A-Za-z}	Установить метку {A-Za-z} на текущей строке
n	Повторить последнюю операцию поиска / или ? n раз
o	Начать ввод новой строки под текущей строкой и введенный текст поместить n раз
["x]p	Поместить текст [из регистра x] сразу после курсора n раз
q{0-9a-zA-Z"}	Записать вводимые символы в именованный регистр {0-9a-zA-Z"}. Обозначение регистра буквами {A-Z} используется, если требуется дописать, а не заменить содержимое соответствующего регистра
rc	Заменить n символов в тексте на символ с
["x]s	Удалить n символов [и поместить их в регистр x] и начать ввод в соответствии с командой редактора s (substitute – заместить). Синоним для команды cl (не строковая)
tc	Переместить курсор вправо до n-го включения символа с в строке.
u	Убрать все изменения (undo)
z Enter	Обновить экран. Строка, на которую указывает курсор, будет помещена в верхнюю строку экрана, а курсор – в первую позицию верхней строки
z.	Перестроить экран. Строка, на которую указывает курсор, помещается в центр экрана, а курсор – в первую позицию этой строки
z-	Перестроить экран. Строка, на которую указывает курсор, помещается внизу экрана, а курсор – в первую позицию этой строки
zb	Перестроить экран. Строка, на которую указывает курсор, помещается внизу экрана, а курсор – в первую позицию этой строки
zt	Перестроить экран. Строка, на которую указывает курсор, помещается верху экрана, а курсор – в первую позицию этой строки
zz	Перестроить экран. Строка, на которую указывает курсор, помещается в центр экрана, а курсор – в первую позицию этой строки
["x]C	Удалить текст, начиная с позиции курсора, до конца текущей строки, а также последующих n-1 строк [и поместить удаленную информацию в регистр x] и начать ввод. Синоним для команды c$ (не строковая)

Работа с окнами в редакторе vim

Введение

Окно в редакторе vim — это место, где вы можете осматривать и изменять содержимое внутреннего буфера редактора. При старте vim по умолчанию открывается одно окно. Однако окон может быть несколько, при этом неважно, на каком терминале в данный момент работает vim: алфавитно-цифровом vt100 или X-терминале.

Когда вы открываете в редакторе файл, vim копирует содержание файла во внутренний буфер. Когда вы изменяете содержимое окна, то вы тем самым изменяете только содержимое буфера редактора. Исходный файл на диске при этом остается неизменным. Чтобы сохранить в файле сделанные изменения, вы должны записать буфер редактора на диск.

Буфер редактора может быть в трех состояниях:

- *активный* (active) — буфер отображается на экране и может модифицироваться командами редактора.
- *невидимый* (hidden) — буфер не отображается на экране, но также может модифицироваться командами редактора.
- *неактивный* (inactive) — буфер не отображается на экране и ничего не содержит.

В дальнейшем для простоты под термином «редактируемый файл» будет подразумеваться соответствующий буфер редактора, если иное не будет оговорено особо.

Запуск редактора с несколькими окнами

По умолчанию vim стартует с одним открытым окном. Однако при разработке любого крупного текста — программы, состоящей из множества подпрограмм, или крупного отчета — вы будете иметь дело с десятками модулей по несколько тысяч строк. Часто оказывается полезным редактировать сразу несколько модулей, например:

```
vim -o mod1.f mod2.f
```

По этой команде будут прочитаны во внутренние буферы два модуля и будут открыты одновременно два окна: половина экрана для одного файла и половина для другого. Если файлов много больше двух, чтобы каждое окно не оказалось

в одну строку, полезно использовать другую форму команды вызова редактора:

```
vim -o2 *.f
```

По этой команде будут вызваны все файлы с расширением `f` (то есть файлы на языке Fortran), но на экране будет организовано только два окна.

Чтобы сделать приемлемый размер активного окна, вы можете установить значение параметра `winheight` в какое-то разумное значение, например, 25.

```
:set winheight=25
```

Когда вы будете переключаться на другое окно, оно автоматически станет размером 25 строк. При этом все остальные окна займут то, что останется на экране.

Далее мы приведем только те команды, которые не связаны с синтаксисом конкретных языков программирования. Для получения подробных сведений можно посмотреть описание, которое вызывается командой редактора `:help index.txt`.

Таблица 3.6. Команды с квадратной скобкой

Команда	Действие
[Ctrl+i	Перейти к началу файла и произвести поиск слова, на который указывает курсор
[(	Перейти n раз к непарной скобке (
[I	Перечислить все включения слова под курсором в текущем файле, начиная с начала файла, и до конца
[i	Показать первое включение слова, на которое указывает курсор, при поиске от начала файла
[[	Переместить курсор назад на n разделов текста или к символу { в первой позиции
[]	Переместить курсор назад на один раздел или к предыдущей скобке } в первой позиции
[{	Переместить курсор назад n раз к незакрытой скобке {
]Ctrl+i	Переместить курсор к первому включению слова, на которое указывает курсор. Начать поиск с текущей позиции
]I	Перечислить все включения слова под курсором в текущем файле, начиная с текущей позиции курсора, до конца файла
][	Переместить курсор вперед n раз на один раздел
]]	Переместить курсор вперед n раз на один раздел
]i	Переместить курсор к первому включению слова, на которое указывает курсор. Начать поиск с текущей позиции
]	Переместить курсор вперед к незакрытой скобке

Метки в тексте

В тексте можно устанавливать метки, которые позже могут быть использованы для быстрого перехода курсора к отмеченному месту. Метки не видны. Это лишь запомненные позиции курсора в тексте. Не следует путать метки с регистрами, это совершенно разные сущности.

Поставить метку 'a можно командой :mark 'a. Или просто — :ma 'a. См. также табл. 3.7.

Имена меток состоят из одной буквы или одной цифры, которой предшествует одиночный апостроф. Различают несколько видов меток:

- 'a — 'z — строчные метки, устанавливаются и действуют в пределах одного файла;
- 'A — 'Z — прописные метки, устанавливаются и действуют во всех редактируемых в данный момент файлах;
- '0 — '9 — нумерованные метки, устанавливаются в файле .viminfo.

Метки нижнего регистра запоминаются и хранятся до тех пор, пока редактируемый файл находится в буфере редактора. Если вы удаляете файл из буфера редактора, то метки теряются. Если вы удалите строку в файле, на которую указывала метка, то эта метка также теряется. Однако метки восстанавливаются при выполнении команды редактора :undo.

Строчные имена меток можно использовать в командах редактора. Например, d'm означает удалить строки от позиции курсора до метки с именем m.

Вы можете, например, установить метки командами :m t для первой строки редактируемого файла (top) и :m b для последней строки редактируемого файла (bottom). Позже можно быстро переходить к концу файла вводом комбинации 'b в обычном режиме редактора. Таким же образом вы сможете перейти к началу текущего файла командой 't. Поскольку строчные метки действуют только в пределах файла, то каждый файл может иметь метки 'b и 't.

Прописные имена меток 'A — 'Z запоминаются редактором вместе с именем редактируемого файла. Таким образом,

переходя по метке, вы можете перескочить из одного файла в другой. В операторах редактирований можно использовать такие метки, если только они указывают на тот же файл. Если параметр `viminfo` не пуст, то метки верхнего регистра хранятся в файле `.viminfo`. Значение метки верхнего регистра будет поддерживаться, даже если вы удалите или добавите в файл какие-то строки, но не удалите строку, на которую указывает метка.

Таблица 3.7. Команды обработки меток

Команда	Действие
m{a-zA-Z}	Установить метку
m' или m`	Установить предыдущую контекстную метку. В эту точку можно будет вернуться командами » или «
:[n]ma[rk] {a-zA-Z}	Установить значение метки с именем {a-zA-Z} в номер строки, который имеет n-я строка от текущей
:[n]a-zA-Z	То же самое, что mark, только можно опустить пробел между командой и именем метки
'{a-z}	Перейти на первый символ, не являющийся пробелом, в строке, помеченной меткой {a-z}
'{A-Z0-9}	Перейти на первый символ, не являющийся пробелом, в строке, помеченной меткой {A-Z0-9}
`{a-z}	Перейти к метке с именем {a-z}
`{A-Z0-9}	Перейти к метке с именем {A-Z0-9}
:marks	Показать список всех текущих меток
:marks arg	Показать список всех меток, перечисленных в arg. Например, marks smh – показать значения меток с именами s, m, h, если таковые метки существуют

Нумерованные метки `'0 – '9` заметно отличаются от остальных. Они не могут быть поставлены непосредственно с помощью команд редактора и устанавливаются в файле `.viminfo` (когда параметр редактора `viminfo` не пуст). Когда записывается файл `.viminfo` (во время выхода из редактора или по команде `:wviminfo`), то метка `'0` устанавливается в текущий файл и текущую позицию курсора. При этом старое значение метки `'0` перемещается в метку `'1`, а старое значение `'1` перемещается в `'2` и так далее. Таким образом, `'0` указывает на позицию курсора, которую он имел в мо-

мент выхода из редактора. Чтобы вернуться, например, на другой день к прежней точке файла, надо выполнить команду оболочки:

```
vim -c "normal '0"
```

С использованием меток связан ряд команд в обычном режиме редактора.

Регистры

Редактор vim имеет специальный вид внутренних буферов, которые называются *регистрами*. Каждый регистр обозначается знаком " и односимвольным именем. Имеется несколько видов регистров:

1. Безымянный регистр "".
2. Десять нумерованных регистров от "0 до "9.
3. Регистр удаления "-.
4. Именованные регистры от "a до "z (или от "A до "Z). Регистр "a и "A — это один и тот же регистр, от того, строчная или прописная буква использована при именовании регистра, зависит только способ его заполнения.
5. Четыре регистра только для чтения: ", ". % "#.
6. Регистр выражений (expression) "=.
7. Регистр выбора (selection) "*.
8. Регистр черная дыра (black hole) "_.

Безымянный регистр

Безымянный регистр заполняется командами удаления текста, такими как: d, r, s, x, c, а также yank. Причем заполнение данного регистра происходит независимо от того, используется или нет в команде именованный регистр. Иными словами, при удалении текст не уничтожается, а перемещается в безымянный регистр. Исключение составляет регистр *черная дыра*. Если этот регистр использован в команде удаления текста, то текст исчезает безвозвратно.

Вы можете использовать содержимое *безымянного регистра*, указав его имя ".

Нумерованные регистры

Регистр номер 0 содержит текст, который был удален наиболее поздней командой удаления текста, если только в самой команде не был явно указан другой регистр. Регистр с номером 1 содержит текст, удаленный предыдущей командой `d`, `r`, `s`, `x`, `c` или `yank`. Регистр с номером 2 содержит текст, удаленный еще более ранней командой и т. д. Иными словами, при очередном удалении текста vim помещает его в регистр 0, сдвинув перед этим содержание остальных регистров, то есть регистр 8 копируется в регистр 9, регистр 7 копируется в регистр 8 и т. д. Естественно, что прежнее содержание регистра 9 теряется.

Если удаляемый текст содержит менее одной строки, то при удалении он помещается в *регистр удаления*.

Регистр удаления

Этот регистр служит для сохранения удаляемого текста, если он занимает меньше одной строки и в команде удаления явно не указан другой регистр.

Именованные регистры

Vim заполняет эти регистры только в том случае, если вы явно попросили это сделать. Если использовано обозначение имени регистра строчной буквой, то содержимое регистра будет замещено новой информацией. Если имя регистра обозначено прописной буквой, то новая информация будет дописана вслед за уже имеющейся в регистре.

Например, команда редактора (в обычном режиме) `2"xdd` приведет к следующим действиям:

1. Из текущего файла будут удалены две строки, начиная с текущей (на которую указывает курсор).
2. Эти две строки будут помещены в регистр с именем `x` заместив предыдущее содержание регистра.
3. Те же две строки будут помещены в *безымянный регистр*, и они же будут помещены в регистр с номером 0.

Продолжим пример. Если позже вы захотите дописать в тот же регистр (с именем `x`) новые пять строк, то это можно сде-

лать следующей командой редактора (в обычном режиме): `5"Xyy`. При этом пять строк просто копируются (дописываются в данном случае) в регистр с именем `x`. Позже содержимое регистра с именем `x` можно записать в тот же самый или другой файл командой `"xp`. Например, вы можете открыть новое окно командой `Ctrl+w` и в нем выполнить команду `"xp`. Далее содержимое окна легко записать в файл на диск командой `:w file_name`, где `file_name` — имя файла.

Регистры только для чтения

Эти регистры содержат информацию, предназначенную только для чтения.

`".` (кавычки и точка) — содержит информацию, которая была введена в режиме ввода, например по команде `i` или `o`.

`"%` (кавычки и знак процента) — содержит имя текущего файла.

`"#` (кавычки и решетка) — содержит имя альтернативного файла, если таковой имеется.

`":` (кавычки и двоеточие) — содержит последнюю командную строку, которая была введена с использованием двоеточия, например:

```
:help registers.
```

Если вы желаете повторить выполнение последней команды, то можно ввести комбинацию `@:`.

Чтобы лучше понять алгоритм изменения в различных ситуациях, полезно поэкспериментировать с содержанием регистров. Содержание всех регистров можно посмотреть командой `:reg`.

Регистр выражений

Этот регистр не является обычным регистром для запоминания информации, а является способом использования выражений в командах редактора там, где обычно используется регистр. *Регистр выражений* является регистром только для чтения, то есть вы не имеете возможности изменить содержимое этого регистра обычными командами, которыми мы изменяли содержимое других регистров в ранее

приведенных примерах. Заполнение регистра производится следующим образом. Вы вводите "=. После ввода знака равенства курсор переместится в командную строку, где вы сможете ввести *выражение*. Ввод *выражения* завершается возвратом каретки (Enter). Чтобы аннулировать ввод *выражения*, введите Esc.

В качестве простого примера использования *Регистра выражений* рассмотрим следующий случай. Пусть нам необходимо определить максимальное количество символов, помещающееся в видимой части окна редактора. Тогда мы могли бы предпринять следующую последовательность действий:

1. Вводим команду "= — курсор перейдет в командную строку.
2. Далее, вводим в командной строке `&winheight*&columns`, и курсор возвращается на прежнее место.
3. Теперь, если сразу ввести команду p, то в текст будет вставлено произведение этих двух параметров, что в моем случае равно 200, то есть 25*80.

Регистр выбора

Этот регистр используется для запоминания и поиска выбранного текста в режиме GUI.

Регистр черная дыра

Когда что-то записывается в этот регистр, то ничего не происходит с остальными регистрами. Если производится попытка чтения из этого регистра, то ничего не считывается.

Примеры использования регистров

Пример переноса текста из одного места (файла) в другое (другой файл).

Для выполнения такого переноса проще всего воспользоваться следующим методом:

- Отметить необходимую часть текста, например, посредством визуального режима: V (построчное выделение текста).
- Удалить выделенный текст, например, командой редактора "ad, которая удалит выделенный текст и скопирует его в регистр с именем a.

- Переключиться на другой файл любым способом, например, командой редактора `:n` или создать новое окно посредством команды `Ctrl+w` и перейти в него.
- В новом месте выполнить команду `"ap`.

Такой перенос можно, конечно, выполнить и без использования регистров, особенно, если перемещение текста производится в пределах одного файла. Для этого можно использовать команды редактора `:copy` и `:move`,

Программы построения таблицы тегов

Программы построения таблицы тегов (индексов) способна анализировать исходные тексты программ на языках: C, C++, EiffelFortran и Java. Наличие таблицы индексов позволяет легко и быстро переходить от одного синтаксического объекта в программе к другому независимо от того, в каком файле располагается искомый объект. Такую таблицу удобно использовать в редакторе текста или просмотрщике. Такие теговские файлы поддерживаются рядом редакторов: vim, emacs, nedit и другими, а также программами просмотра текста, например, less.

Имеется несколько программ формирования таблиц тегов, которые были разработаны для конкретных текстовых редакторов: программа ctags для vim и программа etags для emacs. Здесь мы рассмотрим подробнее лишь первую, так как программа etags в использовании идентична ctags.

Формат вызова программы:

```
ctags [options] [file(s)]
```

Программа ctags способна генерировать теговские файлы с использованием определенных языковых конструкций, которые перечислены ниже для каждого языка отдельно.

C/C++

Программа принимает во внимание следующие конструкции языков C/C++:

- имена макроопределений (взятые из операторов `#define/#undef`);
- перечисления (enum);

- определения функций;
- прототипы;
- декларации классов, перечислений (enum), структур (struct) и объединений (union);
- пространства имен (namespaces);
- определения типов (typedefs);
- переменные (определения и декларации);
- члены классов, структур и объединений.

FORTRAN

Программа принимает во внимание следующие конструкции языка FORTRAN:

- инициализация общего блока (block data);
- общая память (common block);
- точки входа;
- функции;
- интерфейсы;
- метки;
- модули;
- именованные списки ввода/вывода (namelist);
- программы;
- подпрограммы;
- производные типы.

Java

Программа принимает во внимание следующие конструкции языка Java:

- классы;
- поля;
- интерфейсы;
- методы;
- пакеты.

Типы файлов с исходными текстами

Типы файлов с исходными текстами определяются автоматически на основании имеющегося расширения файла, например, `file.c` — текст программы на языке C и т. д., если не указан параметр `--lang`. По умолчанию все файлы с другими расширениями игнорируются. Таким образом, имеется возможность использовать программу ctags следующим образом:

```
ctags *
```

если вас интересует лишь текущий каталог или

```
ctags -R *
```

в случае, когда вам нужен теговский файл по всему ниже лежащему дереву каталогов. При этом в файле будут приняты во внимание все типы исходных текстов, которые известны программе ctags. Обратим также внимание, что в одном теговском файле может быть сгруппирована информация об элементах сотен программ, расположенных в десятках каталогов.

Замечательно, что построенный теговский файл с именем (по умолчанию) `tags` может оказаться полезен также программе less, например, по команде

```
less -t tag-name
```

будет отображен файл, который содержит тег с именем `tag-name`. Подробности можно увидеть на страницах руководства: `man less` или `info less`

Параметры программы ctags

Следует отметить, что существует несколько различных версий этой программы. Здесь рассматривается версия, которая отвечает на команду:

```
ctags --version
```

строкой:

```
Exuberant Ctags 3.4, by Darren Hiebert
<darren@hiebert.com>
```

Параметры, которые воспринимает данная программа, сгруппированы в табл. 3.8.

Таблица 3.8. Параметры программы ctags

Параметр	Значение
-a	Дописать имеющийся теговский файл
-B	Использовать команды поиска в обратном направлении для определения положения тега в файле, то есть поиск вида ?expr?. Игнорируется, если использован параметр -e
-e	Построить теговский файл для использования редактором emacs (он несколько отличается по формату от теговского файла для vim)
-f filename	Использовать имя filename для теговского файла, по умолчанию используется имя tags или TAGS для параметра –e. Для выдачи информации на стандартное устройство вывода следует использовать знак - (минус) на месте имени файла
-F	Использовать команды поиска в прямом направлении для определения положения тега в файле, то есть поиск вида /expr/. Игнорируется, если использован параметр –e
-h list	Определяет список окончаний имен файлов, разделенных точками, который интерпретируется как окончания имен файлов заголовков или файлов типа include. Этот параметр воздействует на область видимости различных тегов. Теговский тип, который не находится в файле типа include, рассматривается как видимый только в том файле, где он определен. Значение параметра по умолчанию равно .h.H.hh.hpp.hxx.h++.inc.def. Следует обратить внимание на параметр --file-scope
-I tokenlist	Определяет список слов, которые будут обрабатываться специальным образом при анализе текста на C/C++. Параметр предназначен для учета особенностей, связанных с использованием препроцессорных макроопределений. Если перечисленные в списке слова являются простыми подстроками (без специальных знаков), то они просто игнорируются во время анализа исходного текста. Если сразу за словом следует знак + (плюс), то ctags будет также игнорировать любые выражения в скобках, которые могут следовать в исходном тексте за обозначенным словом. Если два слова разделены знаком = (равно), тогда первое слово заменяется вторым словом для правильного анализа текста. Список слов может быть набран в командной строке или находиться в файле. Если список слов tokenlist начинается символом . (точка) или слэшем, то это рассматривается как имя файла, в котором хранится список слов, разделенных пробелами. Например, для текущего каталога можно использовать имя ./tokens. Можно использовать несколько параметров -I
-L file	Читать из file список имен файлов, которые должны быть учтены при построении результирующего теговского файла. Если на месте имени файла стоит знак - (минус), то список имен файлов будет считываться со стандартного устройства ввода
-n	То же самое, что --excmd=number

Значение
То же самое, что --excmd=pattern
Использовать каталог с именем path как главный каталог для всех исходных текстов, то есть остальные имена файлов будут относительны к главному каталогу, если не указано абсолютное имя файла
То же что –recurse
То же что --sort=no, то есть теги не сортировать
То же что –verbose, то есть информировать о происходящем подробно
Вывести табличный вид перекрестных ссылок на экран вместо построения теговского файла. Выводимая информация содержит: имя тега; тип тега; номер строки, где найден тег; имя файла, в котором найден тег; исходная строка текста, которая содержит тег
Дописывать (yes) или нет (no) уже существующий теговский файл
Указать типы тегов, которые следует обрабатывать для программ на языке C/C++. Каждый тип обозначается одной буквой. Если букве предшествует знак - (минус), то данный тип исключается из списка по умолчанию. Если типу предшествует знак + (плюс), то тип добавляется к списку по умолчанию. Отсутствие знаков - (минус) или + (плюс) интерпретируется как включение только тех типов, которые указаны явно, то есть не учитывая сисок по умолчанию. Обозначения типов приведено ниже: c – классы; d – макроопределения; e – перечисления; f – определения функций; g – имена перечислений; m – классы, структуры, имена объединений; n – пространство имен; p – прототипы функций и декларации; s – имена структур; t – определения типов (typedefs); u – имена объединений; v – определения имен переменных; x – внешние и (forward) переменные. В дополнение к вышеприведенному используются следующие модификаторы. A –- записать доступ каждого члена в теговский файл. Эта информация записывается в файл с расширением access; C – включить описатель класса для каждого члена в форме class::member. Это позволяет определить положение тега с использованием названия класса, например, :tag class::member. Такая возможность увеличивает размер теговского файла примерно вдвое, поэтому выключена по умолчанию
Определяет список типов тегов для языка Eiffel с использованием тех же соглашений о формате, что приведены выше для языка C/C++ (смотрите описание параметра --c-types). Следующие типы тегов определены для языка Eiffel: c – классы; f – особенности (features); l – локальные сущности (local entities). В дополнение может использоваться модификатор C. Если он используется, то в теговский файл включается дополнительная информация о принадлежности каждого члена к определенному классу, с тем, чтобы можно было позже воспользоваться командой редактора :tag class.member. По умолчанию эта возможность выключена, поскольку приводит к увеличению размера теговского файла

продолжение ⇙

Таблица 3.8 *(продолжение)*

Параметр	Значение
--etags-include=file	Включает ссылку на file в теговский файл. Этот параметр может использоваться много раз. Можно использовать только в редакторе emacs
--excmd=type	Определить тип команд редактора Ex, который является составной частью vim, с помощью которых будет определяться положение тегов в файле. Допустимыми значениями являются: n[umber] – для определения положения тегов использовать только числовые значения (номер строки, символа); p[attern] – для определения положения тегов использовать только поиск регулярных выражений; m[ixed] – используется в основном тип p[attern] с некоторыми исключениями. Детали следует смотреть в info ctags
--file-scope=yes|no	Определяет, как рассматриваются теги: только в пределах файла, в котором определены или являются видимыми извне. По умолчанию, используется значение yes. Смотрите также описание параметра -h
--file-tags=yes|no	Должны ли генерироваться теги для имен файлов с исходными текстами. По умолчанию установлено no
--format=level	Определить формат теговского файла. Сейчас принимаются значения level 1 и 2. Значение level равное 1 означает обычный формат. Значение level равное 2 означает расширенный формат, который содержит дополнительные теги. По умолчанию установлено level=2. Параметр игнорируется, если использован –e
--fortran-types=types	Определить типы тегов для языка Fortran, которые будут фигурировать в результирующем теговском файле. Смотрите описание параметра –c-types для определения формата строки types. Для языка Фортран поддерживаются следующие типы тегов: b – block data; c – common blocks; e – entry points; f – определение функции; i – интерфейсы; l – метки; m – модули; n – списки ввода/вывода (namelists); p – программы; s – подпрограммы (subroutines); t – производные типы
--help	Вывод справочной информации на устройство стандартного вывода
--java-types=types	Определить список типов тегов для языка Java, которые будут фигурировать в результирующем теговском файле. Смотрите описание параметра –c-types для определения формата строки types. Для языка Java поддерживаются следующие типы тегов: c – классы; f – поля; i – интерфейсы; m – методы; p – пакеты. В дополнение используются модификаторы: A – записать доступ каждого поля в теговский файл. Эта информация записывается с использованием дополнительного флага с именем access. C – включить дополнение, то есть тег, который описывает каждый член класса в форме class.member, Это позволяет позже определить положение тега посредством команды редактора :tag class.member. По умолчанию эта возможность выключена, поскольку может привести к удвоению размеров генерируемого теговского файла

Параметр	Значение
--kind-long=yes|no	Определить, как будут описываться дополнительные теги (полными словами или одной буквой)
--lang=c|c++|eiffel|fortran|java	По умолчанию программа ctags автоматически определяет язык исходного текста на основании окончаний имен файлов, игнорируя файлы с нераспознанными окончаниями. Этот параметр заставляет рассматривать все файлы как тексты заданного языка
--langmap=map	Определить связь между окончанием имени файла и языком. Например, строка: --langmap=c:.c.ec.xs,java:+.j определяет, что файлы, имена которых заканчиваются на .c, .ec, .xc, рассматриваются как программы на языке C. В то же время, файлы, имена которых заканчиваются на .j, и определенные по умолчанию окончания являются программами на языке Java
--line-directives=yes или --line-directives=no	Определить, будет ли распознаваться директива #line
--links=yes|no	Определить, будут ли использоваться символические ссылки. По умолчанию символические ссылки игнорируются.
--recurse=yes|no	Просматривать рекурсивно указанные каталоги или текущий каталог, если ничего не указано, а также следовать всем символическим ссылкам. Следует понимать, что вы собираетесь делать, используя данный параметр. Может оказаться, что вы имеете массу каталогов, имена которых начинаются с точки

Использование таблицы тегов

Часто оказывается, что в текстах программ, которые вы редактируете, довольно много одинаковых тегов. В табл. 3.9 приведены команды, которые позволяют перемещаться между одинаковыми тегами.

Таблица 3.9. Команды для перемещения по тегам

Команда	Действие
:ts[elect][!] [ident]	Перечислить теги, удовлетворяющие ident, использовав для этого файл(ы) тегов. Если ident опущен, то используется последнее имя тега из стека тегов
:sts[elect][!] [ident]	Выполнить :ts[elect][!] [ident] и разделить текущее окно для всех вхождений указанного тега. Иными словами, в каждом окне будет свое положение тега, например, в первом – тег из строки 17, а во втором тег с тем же именем из строки 237
:tj[ump][!] [ident]	То же, что :ts[elect][!], но переход к единственному тегу будет выполнен сразу

продолжение

Таблица 3.9 *(продолжение)*

Команда	Действие
:stj[ump][!] [ident]	То же, что :tj[ump][!] [ident], но в дополнение окно разбивается на два окна, в одном из которых курсор указывает на тег с именем ident
:[count]tn[ext][!]	Перейти вперед к count-й встрече тега с текущим именем. По умолчанию к следующей встрече
:[count]tp [revious][!]	Перейти назад к count-й встрече тега с текущим именем По умолчанию к предыдущей встрече
:[count]tN[ext][!]	То же, что и :[count]tp[revious][!]
:[count]tr[ewind][!]	Перейти к count-й встрече тега с текущим именем. По умолчанию к первой встрече
:tl[ast][!];	Перейти к последней встрече тега с текущим именем

Получение информации о положении курсора

Таблица 3.10. Информация о положении курсора

Команда	Действие
Ctrl+g или :f[ile]	Напечатать текущее имя файла и позицию курсора
count Ctrl+g	То же, что Ctrl+g, но дополнительно печатается абсолютное имя файла. Если значение count больше 1, то печатается и имя буфера
g Ctrl+g	Вывести текущую позицию курсора в следующем виде: Колонка, Строка, Символ. Если символ занимает более одной колонки, например, <TAB>, то колонки указываются через дефис

Строчные команды редактора

Редактор имеет список из более чем 300 различных команд

```
:h ex-cmd-index
```

Вряд ли имеет смысл описывать здесь все команды — мно гие команды выполняют похожие функции. В связи с эти мы остановимся на описании нескольких команд и груп команд. Полный список команд и их описание можно на ти с помощью команды:

```
:help index.txt
```

В специальном разделе «Сценарии» мы отметим ряд важ ных сценариев использования возможностей редактора vim.

Таблица 3.11. Строковые команды редактора vim

Команда	Действие
:!	Отфильтровать одну или группу строк, или выполнить внешнюю команду
:!!	Повторить предыдущую команду :!cmd
:!#	То же самое, что команда :n, где n – номер строки, куда следует переместить курсор
:!*	Выполнить команды редактора Ex, которые содержатся в регистре
:!<	Сдвинуть текст влево на величину shiftwidth
:!=	Показать номер строки, где стоит курсор
:!>	Сдвинуть текст вправо на величину shiftwidth
:!@	Выполнить команды редактора Ex, которые содержатся в регистре
:N[ext]	Перейти к следующему файлу в списке аргументов редактора
:P[rint]	Напечатать строки
:X	Запросить ключ для шифрования файла
:a[ppend]	Добавить текст после определенной (или текущей) строки
:ab[breviate]	Показать все имеющиеся символы подстановки и их значения, а также режим, в котором действует подстановка: ввод, командный или для обоих
:ab[breviate] abc	Показать заменяемые подстроки, которые начинаются с символов abc
:ab[breviate] abc new	Добавить подстановку строк abc на строки new в таблицу подстановок. Обычно такая операция выполняется для длинных выражений, например: :ab lo LoadFarm заменить длинное имя скрипта LoadFarm на короткую аббревиатуру lo. Эту аббревиатуру можно использовать как в поле команды, так и при вводе текста. Если вам потребуется отменить действие команды ab, то сделать это можно посредством команды :una
:abc[lear]	Очистить таблицу подстановок, то есть удалить все введенные по команде :ab[breviate] сокращения
:al[l]	Открыть отдельное окно для каждого файла, перечисленного в списке аргументов
:am[enu]	Ввести новое меню для всех режимов (визуального, ввода, обычного и т. д.). Используется в основном в редакторе, где поддерживается GUI
:an[oremenu]	Ввести новое меню, которое не будет переотображаться
:ar[gs]	Показать список аргументов редактора
:argu[ment]	Перейти к файлу, имя которого содержится в списке аргументов
:as[cii]	Вывести ASCII-код символа, который находится под курсором
:au[tocmd]	Добавить новую команду к списку команд, которые vim будет выполнять автоматически при наступлении определенного события
:aug[roup]	Определить имя группы автокоманд, которая будет использоваться в команде :au[tocmd]. Имя группы end или END определяет группу по умолчанию

продолжение ⇩

Таблица 3.11 *(продолжение)*

Команда	Действие
:b[uffer]	Перейти к редактированию указанного буфера из списка активных буферов
:bN[ext]	Перейти к редактированию следующего буфера из списка активных буферов
:ba[ll]	Открыть окно для каждого файла в списке буферов
:bad[d]	Добавить файл к списку буферов (не загружая сам файл)
:bd[elete]	Удалить заданные файлы из списка буферов
:bl[ast]	Перейти к последнему файлу в списке буферов
:bm[odified]	Перейти к следующему файлу в списке буферов, который был модифицирован
:bn[ext]	Перейти к следующему файлу в списке буферов
:bp[evious]	Перейти к предыдущему файлу в списке буферов
:br[ewind]	Перейти к первому буферу из списка буферов
:brea[k]	Когда используется внутри цикла, то следующей исполняемой командой будет первая команда после цикла
:buffers	Показать список буферов
:bun[load]	Освободить заданный буфер (по умолчанию – текущий)
c[hange]	Заменить строку или группу строк
cN[ext]	Перейти к предыдущей ошибке
:ca[bbrev]	То же самое, что :ab[breviate], но в командном строчном режиме
:cal[l]	Вызов функции, определенной пользователем или встроенной в vim Чтобы увидеть список доступных в данный момент функций, следует воспользоваться командой fu[nction]
:cc	Перейти к определенной ошибке
:cd	Изменить текущий каталог
:ce[nter]	Расположить текст в центре строки
:cf[ile]	Прочесть файл с сообщениями об ошибках компиляции программы
:ch[ange]	Вставить строки текста поверх существующих
:che[ckpath]	Перечислить дополнительные файлы, если они имеются
:cl[ist]	Перечислить все ошибки компиляции программы
:cla[st]	Перейти к последней ошибке компиляции программы
:clo[se]	Закрыть текущее окно
:cm[ap]	То же, что map, но только для командного строчного режима
:cnew[er]	Перейти к более свежему списку ошибок
:cnf[ile]	Перейти к первой ошибке компиляции в следующем файле
:co[py]	Копировать строки

Команда	Действие
:col[der]	Перейти к более старому списку ошибок
:com[mand]	Определить команду пользователя (если без параметров – перечислить определенные пользователем команды)
:con[tinue]	Перейти к команде while, то есть в начало цикла
:d[elete]	Удалить строки из текущего файла
:di[splay]	Отобразить содержание регистров
:dig[raphs]	Показать все введенные диграфы
:dj[ump]	Переход курсора к ближайшему оператору #define
:dl[ist]	Показать список всех операторов #define
:do[autocmd]	Применить автокоманды к текущему буферу
:doauto[all]	Применить автокоманды ко всем загруженным буферам
:ds[earch]	Найти оператор #define
:dsp[lit]	Разделить окно vim на два окна и в новом окне перейти к оператору #define
:e[dit]	Начать редактирование файла
:ec[ho]	Вывести результат выражения
:echoh[l]	Установить режим выделения символов для информации, которая выводится последующими командами :ec[ho]
:el[se]	Часть команды :if
:elsei[f]	Часть команды :if
:en[dif]	Завершение команды :if
:endf[unction]	Завершение пользовательской функции
:endw[hile]	Завершение цикла while
:ex	То же, самое, что и команда :edit
:exe[cute]	Выполнить результат выражения
:exi[t]	То же что и xit, то есть записать на диск буфер, если он был изменен, и выйти из редактора
:f[ile]	Установить или показать текущее имя файла, ассоциированное с данным буфером
:files	Перечислить все файлы из списка буферов
:filet[ype]	Включить/выключить определение типа файла
:fin[d]	Найти файл в списке каталогов path
:fix[del]	Установить значение ASCII-кода для Del
:fu[nction]	Определить пользовательскую функцию
:g[lobal]	Выполнять заданные команды над заданным множеством строк текста
:go[to]	Перейти к заданному байту в текущем буфере

продолжение

Таблица 3.11 *(продолжение)*

Команда	Действие
:gr[ep]	Выполнить команду, имя которой содержится в параметре grepprg (по умолчанию – grep –n)
:gu[i]	Запустить графический режим (GUI)
:h[elp]	Разделить текущее окно vim и показать в нем страницу руководства vim. Если окно со страницей руководства было открыто ранее, то используется ранее открытое окно
:helpf[ind]	В режиме GUI открыть диалоговое окно для ввода термина, о котором хотелось бы найти страницу руководства
:hi[ghlight]	Эта команда предназначена для управления выделением текста в различных условиях, например, в зависимости от редактируемого языка программирования
:hid[e]	Закрыть текущее окно, если оно не является единственным окном vim на экране. При этом буфер становится невидимым
:his[tory]	Вывести файл истории введенных команд редактора vim
:i[nsert]	Начать ввод (вставку) текста в текущий буфер
:if	Начало конструкции условий :if expr –commands – endif. Последовательность команд commands будет выполняться всякий раз, как только выражение expr даст в результате не нуль
:ij[ump]	Перейти к месту в буфере, в котором содержится указанная подстрока
:il[ist]	Перечислить все места в буфере, где встречается указанная подстрока (практически то же что [I или]I), например, il /слить/ – показать все места в тексте, где встречается подстрока /слить/
:int[ro]	Вывести начальное сообщение vim
:is[earch]	Перейти к подстроке, которая удовлетворяет выражению, например, /сообщ/
:isp[lit]	Разделить окно vim на два и перейти к заданной подстроке, например, isp /язык/.
:j[oin]	Объединить соседние строки текста в одну строку
:ju[mps]	Вывести список переходов по тексту
:k	Определить метку в данном месте текста, например, :k b определить метку с именем b в месте, где стоит курсор
:l[ist]	Вывести заданные строки (по умолчанию – весь буфер)
:la[st]	Перейти к последнему файлу в списке аргументов
:le[ft]	Выровнять строку по левому краю
:let	Присвоить значение переменной или внутреннему параметру редактора
:ls	Показать список всех буферов
:m[ove]	Переместить строки
:ma[rk]	Определить метку с заданным именем в месте, где стоит курсор

Команда	Действие
:mak[e]	Выполнить внешнюю команду, имя которой содержится в параметре makeprg (смотрите описание параметра makeprg на странице makeprg).
:map	Показать имеющиеся или ввести новые подстановки одних символьных последовательностей в другие. Например, :map! i Институт Высокой Технологии после ввода такой команды, при вводе буквы i она немедленно будет заменена на три слова Институт Высокой Технологии. Но будьте осторожны. Заменяемая буква совпадает с командой начала ввода i. Может оказаться, что когда вы попробуете начать ввод текста, то получите неожиданный результат. Механизм подстановок весьма гибок, подробности следует смотреть на страницах :h map.txt.
:marks	Показать все установленные метки
:mes[sages]	Показать ранее выданные редактором сообщения
:mk[exrc]	Записать значения внутренних переменных, а также введенные отображения в файл (по умолчанию .exrc)
:mks[ession]	Записать все параметры данной сессии в заданный файл
:n[ext]	Перейти к следующему файлу в списке аргументов редактора
:new	Открыть новое окно редактора Vim
:nu[mber]	Вывести строки текста вместе с номерами строк
:on[ly]	Закрыть все окна редактора, кроме текущего окна
:opt[ions]	Открыть новое окно редактора для просмотра и установки значений параметров
:p[rint]	Вывести строки текста (по умолчанию текущую строку)
:pe[rl]	Выполнить команду языка Perl
:pre[serve]	Записать содержание всех буферов в файл подкачки
:prev[ious]	Перейти к предыдущему файлу в списке аргументов
:pu[t]	Поместить текст из регистра после указанной строки (по умолчанию текущей)
:pwd	Вывести имя текущего каталога
:py[thon]	Выполнить команду языка Python
:pyf[ile]	Выполнить скрипт языка Python
:q[uit]	Закрыть текущее окно
:qa[ll]	Выйти из редактора
:r[ead]	Прочесть файл и добавить его в текущий буфер (по умолчанию после текущей строки). Другим вариантом использования команды является чтение вывода внешней программы, например, :r !cmd, что означает: выполнить внешнюю команду с именем cmd и результат поместить в строки ниже курсора
:redi[r]	Перенаправить сообщения редактора в файл или в регистр редактора (смотрите также раздел «Регистры»)

продолжение

Таблица 3.11 *(продолжение)*

Команда	Действие
:reg[isters]	Показать содержимое всех регистров
:retu[rn]	Вернуться из пользовательской функции
:rew[ind]	Вернуться к первому файлу в списке аргументов
:ri[ght]	Выровнять текст по правому краю
:rv[iminfo]	Чтение и установка параметров из файла viminfo
:s[ubstitute]	Для каждой указанной строки найти подстроку и произвести замену
:sN[ext]	Разделить окно на два окна и перейти к предыдущему файлу из списка параметров
:sa[rgument]	Разделить окно на два окна и перейти к указанному файлу из списка параметров
:sal[l]	Открыть окно для каждого файла из списка аргументов
:se[t]	Показать или установить значения параметров редактора
:sf[ind]	Разделить окно на два и в новом окне начать редактировать заданный файл из каталога path. Если файл не найден, то новое окно не создается
:sh[ell]	Запустить оболочку
:sni[ff]	Послать запрос системе SNiFF+. SNiFF+ является системой управления массивами исходных текстов в больших проектах. Система SNiFF+ позволяет использовать vim в качестве средства редактирования и отображения текстов. С другой стороны, vim может обращаться к системе SNiFF+ с разнообразными запросами. Подробности можно почерпнуть на сервере: http://www.takefive.co.at/
:so[urce]	Прочесть и выполнить команды редактора из заданного файла
:star[tinsert]	Перейти к вводу текста с позиции курсора. Практически работает так же, как команда i в обычном режиме
:st[op]	Приостановить редактор и перейти в оболочку
:sw[apname]	Показать имя текущего файла резервной копии (swap-файла)
:sy[ntax]	Включить выделение элементов текста текущего буфера в соответствии с синтаксисом
:syncbind	Заставляет все окна редактора, где выданы команды :syncbind, перемещаться по тексту синхронно. Иными словами, если одно из окон установлено в начало текста, то и все остальные будут установлены в начало своих буферов
:t	То же самое, что команда :copy
:tN[ext]	То же самое, что команда :tp[revious] – перейти к предыдущему тегу
:ta[g]	Перейти к указанному тегу
:tags	Показать содержимое таблицы тегов

Команда	Действие
:tcl	Выполнить команду языка Tcl
:tcld[o]	Выполнить команду языка Tcl для каждой строки в указанном диапазоне (по умолчанию всего буфера)
:u[ndo]	Вернуть буфер к состоянию, которое он имел до последнего изменения
:una[bbreviate]	Удалить подстановку из списка подстановок
:unl[et]	Удалить переменную
:unm[ap]	Удалить подстановку символов из списка
:up[date]	Записать буфер на диск, если буфер был изменен
:v[global]	То же самое, что команда g[lobal]!. Команда выполняется в два шага: на первом шаге производится сканирование текста в указанном диапазоне (по умолчанию весь буфер) и отмечаются строки, где встречается искомая последовательность, а на втором шаге производится требуемая замена в отмеченных строках
:ve[rsion]	Показать версию редактора, а также все установленные при компиляции параметры
:vi[sual]	Если введено в режиме ex, то редактор перейдет в обычный режим. В противном случае, эта команда работает так же, как команда e[dit]
:vie[w]	Просмотреть указанный файл в режиме только чтение
:w[rite]	Записать заданную часть буфера (по умолчанию весь буфер) на диск
:wa[ll]	Записать все измененные буферы на диск. Буферы, которые не ассоциированы с файлом, не записываются
:wh[ile] expr commands endw[hile]	Выполнять последовательность commands (между :wh[ile] и :endw[hile]) до тех пор, пока выражение expr не равно нулю. Если при выполнении команд commands возникла ошибка, то управление перейдет к первой после :endw[hile] команде
:wn[ext]	Записать текущий буфер в файл и перейти к следующему файлу в списке аргументов
:wq	Записать текущий буфер и закрыть текущее окно или выйти из vim
:wqa[ll]	Записать все измененные буферы и выйти из vim
:wv[iminfo]	Записать информацию viminfo в заданный файл (по умолчанию в viminfo).
:x[it]	Записать буфер, если он был изменен, и закрыть текущее окно или выйти из vim, если окно было единственным
:xa[ll]	То же, что :wqa[ll]
:y[ank]	Скопировать заданные строки в указанный регистр
:z	Вывести на экран заданные строки из буфера
:~	Повторить последнюю команду :substitute

Пример замены подстроки

Например, команда

```
:g/section/s//chapter/g
```

заменит `section` на `chapter` во всем буфере. То же самое может быть реализовано командой:

```
%s/section/chapter/g
```

Сценарии использования редактора vim

Поскольку многие возможности редактора не являются тривиальными, то здесь мы перечислим несколько сценариев, где использование данного редактора будет достаточно эффективным. Начнем с простых примеров.

Общие замечания

Редактор vim довольно быстро работает с длинными текстами (например, 100 тысяч строк), что позволяет использовать редактор для просмотра различных файлов протоколов (log-файлов), почтовых ящиков и других больших файлов для поиска каких-либо ключевых слов и выражений. Например, для просмотра протокола сообщений Linux можно использовать редактор vim, вызванный командой `view`.

Просмотр протокола системных сообщений /var/log/messages

Для простого просмотра файла vim удобно вызывать командой `view`:

```
view /var/log/messages
```

Использовать vim для просмотра удобно не потому, что другие программы плохи, а просто из соображений экономии усилий, чтобы оставаться в одной и той же среде, в которой вы наработали определенные навыки и освоили (приготовили) удобные для вас инструменты.

Положим, вы прочитали файл `/var/log/messages`. Теперь вы сможете получить о нем массу информации:

- `Ctrl+g` покажет имя файла и количество строк;
- `g-Ctrl+g` покажет положение курсора, а также количество строк и количество символов в файле.

Если вы захотите посмотреть список строк, где встречается определенная подстрока, то укажите на нее курсором, затем введите:

[I — перечислить встречи подстроки во всем файле;

]I — перечислить встречи подстроки, начиная с позиции курсора до конца файла.

Параметры редактора

Перед началом использования редактора vim полезно посмотреть, с какими параметрами вызван редактор. Командой

```
:set
```

вы сможете вывести все параметры, чьи значения отличаются от значений по умолчанию. Значения всех остальных переменных можно также увидеть посредством команды редактора `:set`.

Часть команд зависят от параметров, значения которых устанавливаются во время трансляции редактора vim. Значения таких параметров можно получить командой редактора `:version`.

Редактирование нескольких файлов одновременно

В ряде ситуаций, когда вы отлаживаете программу из нескольких подпрограмм или готовите описание из нескольких отдельных глав при внесении изменений в один модуль требуется вносить согласованные изменения в другие модули. В такой ситуации удобно сразу указать редактору имена всех модулей. Сделать это очень просто:

```
vim mod_1 mod_2 ... mod_N
```

При этом можно указать желаемое количество окон, которые организует vim. Можно открыть хоть дюжину окон, но эффективнее использовать 2-3 окна. Тогда для трех открываемых окон команда будет выглядеть так:

```
vim -o3 mod_1 mod_2 ... mod_N
```

В каждом окне появится файл из списка аргументов. Поскольку были запрошены три окна, то vim поместит в них первые три файла из списка. Когда редактор уже вызван, список аргументов можно посмотреть командой:

```
:ar[gs]
```

С помощью этой же команды вы сможете перейти к редактированию других файлов, например:

```
:ar[gs] mod_12 mod_19
```

Переместиться по списку аргументов вперед можно командой:

```
:n[ext]
:N[ext]
```

а назад — командой:

```
:prev[ious]
```

Перейти к любому файлу в списке аргументов можно командой:

```
:argu[ment] n
```

здесь n означает номер файла в списке аргументов слева направо.

Переход к первому файлу в списке выполняется командой:

```
:rew[ind]
```

а к последнему —командой:

```
:la[st]
```

Часто при внесении изменений в текст программ требуется редактировать сразу в нескольких описаниях функций или структур, которые могут находиться в других местах файла или в разных файлах. Быстрый переход от одной точки в одном файле к другой точке в другом файле проще всего реализовать с использованием механизма меток.

Например, командой редактора:

```
:ma[rk] W
```

вы можете отметить вашу рабочую точку меткой W (вообще можно использовать любую прописную букву). То же самое вы можете реализовать посредством команд `:k W` или `m-W`. Тогда переход из любого файла к рабочей точке, где вы модифицируете текст, мог бы быть выполнен посредством команды: `'W` или `` `W ``. Описание механизма меток можно посмотреть в разделе «Метки в тексте». Если вы при этом отметили интересующую вас таблицу командой `m-T` или `:ma[rk] T`, то после этого вы можете переходить от таблицы к точке ввода сколько угодно раз. Кстати, точку ввода

можно и не отмечать, если вы только хотите взглянуть на таблицу и сразу вернуться. Вы можете перейти к таблице по команде 'T, а возврат произвести по команде ».

Другой способ быстрого перемещения курсора по файлам проекта можно реализовать с помощью команд редактора, которые используют специальную таблицу синтаксических элементов текста — *тегов*. Как мы помним, таблица для программ на языках C и C++ строится программой ctags, которая обсуждается в разделе «Программы построения таблицы тегов». Например, по команде:

```
:ta[g] function
```

будет произведено перемещение курсора в тот файл и ту точку, где находится тег с именем `function`. Если вам потребуется получить полный список точек, где встречается в тексте тег с именем `name`, вам следует использовать команду:

```
:ts[elect] name
```

Можно использовать альтернативный вариант тех же команд. Если вы введете в обычном режиме `Ctrl+]`, то это будет эквивалентно команде `:ta[g] word`, где `word` есть слово, на которое указывает курсор. С другой стороны, ввод комбинации `g]` в обычном режиме полностью эквивалентен команде `:ts[elect] word`, где `word` есть слово, на которое указывает курсор.

Вызвать программу формирования таблицы тегов и построить таблицу тегов можно перед вызовом редактора или во время редактирования. Большая часть доступных программ построений таблицы тегов является ориентированными на популярные языки C, C++, Java, FORTRAN и т. д.

Допустим, вы желаете быстро узнать, в какой программе (файле) встречается определенное регулярное выражение и перейти к редактированию этого файла. Это можно сделать командой:

```
:grep "популярные" *.tex
```

Команда передаст все параметры программе grep, которая просмотрит указанные файлы (как видно из примера, файлы не обязаны быть только программами) и загрузит первый файл, в котором найдено слово `популярные`. Далее можно

перемещаться по списку файлов с помощью команд просмотра `:cnext`, `:clist` и т. д. Подробности можно почерпнуть со страниц руководства, которое встроено в vim, например:

```
:h clist
```

Другие способы перемещения по текстам

Пусть вы редактируете (или просматриваете) множество файлов, состоящее, например, из нескольких десятков файлов. Тогда можно быстро перемещаться по файлам и отдельным строкам с помощью таблицы скачков и с помощью таблицы буферов.

Все перемещения курсора по файлу (файлам) vim записывает в специальную таблицу скачков, содержание которой можно посмотреть командой редактора `:ju[mps]`. Далее по таблице можно перемещаться (то есть перемещаться по файлу или файлам), используя команды редактора `[n]Ctrl+i` — перейти по таблице к более ранним положению курсора и `[n]Ctrl+o` — перейти к более поздним положениям курсора. Здесь `n` — есть число строк таблицы скачков, на которое следует переместиться по таблице скачков (по умолчанию на 1). Этот способ перемещения действует как в пределах одного файла, так и между файлами.

Аналогично таблице скачков vim ведет таблицу буферов. Если вы желаете переместиться из одного редактируемого файла в другой, ранее редактируемый файл, то вы можете ей воспользоваться. Таблица может быть отображена командой редактора:

```
:files
```

После этого вы сможете ввести номер буфера в таблице буферов и нажать `Ctrl+^` — редактор немедленно перейдет к указанному файлу.

Пример использования механизма вызова программ по команде редактора :make

По умолчанию vim по команде `:make` выполнит командную строку, которая содержится в параметре `makeprg`. Вы можете использовать в качестве такой строки две команды, соединенные программным каналом, как показано ниже:

```
:set makeprg=gmake\ \\\|\ myfilter
```

Таким образом можно использовать программу latex вместо программы make. При этом резервирование места для параметров (комбинация **$***) производится так:

```
:set makeprg=latex\ \\\\nonstopmode\ \\\{$*}
```

После ввода команды `:make` (в данном случае будет запущена программа latex) вы сможете просматривать диагностические сообщения программы latex посредством команд редактора.

Пример более сложной команды

Пусть нам требуется время от времени заменять одно слово в тексте на какое-то другое в соответствии с определенным алгоритмом. Реализовать такое можно посредством следующих шагов.

1. Отметить слово с помощью визуального режима.
2. Выполнить команду редактора K с предварительно установленным внутренним параметром редактора `keywordprg`.
3. Наконец, заменить отмеченное слово в тексте на результат, полученный с помощью команды редактора K.

Чтобы выполнить перечисленные шаги, выдадим прежде следующие команды установки:

```
:set keywordprg=ESymb
:map F K<Enter>:source /tmp/subst11<Enter>
```

В первой установке внутреннему параметру `keywordprg` присваивается имя сценария, который будет вызываться по команде редактора K. В данном случае сценарий производит требуемую перекодировку, а результат в виде команды редактора выводит в файл с именем `/tmp/subst11`. Вторая строка обеспечивает подстановку имени новой команды F в командную последовательность редактора, в которой сначала выполняется команда K, а затем выполняются команды редактора, сформированные в файле `/tmp/subst11` командой K.

В результате, при вводе в обычном режиме команды F будет выполнена команда K, а затем команды редактора сформированные в файле `/tmp/subst11` командой K. Ниже приведены тексты соответствующих сценариев.

Сценарий 1

```
#!/bin/bash
#-----------------------------------------------+
# Usage:   ESymb word    |
# Creation date:  Thu Feb 3 09:00:03 MSK 2000 |
#-----------------------------------------------+
WORD="$1"
echo s/"$WORD"/`symb $WORD`@"$WORD"/ > /tmp/subst11
exit
```

Сценарий 2

```
#!/bin/sh
#  Перекодировать слово, представленное в коде koi-8,
#  в число, которое
#  есть конкатенация номеров позиций первых четырех
#  букв слова в алфавите. #
ALPH='АаБбВвГгДдЕеЖжЗзИиЙйКкЛлМмНнОоПп'
ALPH=$ALPH'РрСсТтУуФфХхЦцЧчШшЩщЪъЫыЬьЭэЮюЯя'
echo "$1" | \     awk --assign Line=$ALPH --file ~/bin/
ProgN.awk
exit
# Awk программа ProgN
Len=length($1)
if (Len < 4) Len = 4 if (Len > 4) Len = 4
i=1 while (i <= Len) {
printf("%02d",index(Line,substr($1,i,1))) i=i+1
printf("\n")
```

Перенаправление сообщений редактора

Рассмотрим теперь пример перенаправления сообщений из vim в файл или в регистр.

```
redi > Messages
```

По данной команде все сообщения редактора будут перенаправлены в файл с именем `Messages`. Чтобы дописать сообщения в конец файла, можно использовать команду:

```
redi >> Messages
```

Чтобы перенаправить все сообщения редактора в регистр, следует использовать команду:

```
redi @R
```

где R может быть любым регистром (смотрите также раздел «Регистры»).

Чтобы завершить вывод сообщений в файл, следует использовать команду:

```
redi END
```

Примеры использования элементов внутреннего языка vim

Пусть нам требуется вывести на экран последовательные значения переменной `i`. Это можно выполнить командой редактора:

```
:execute 'let i=0 | while i < 5 | echo i | let i = i + 1
| endwhile'
```

Другой пример: требуется вывести выделенное сообщение

```
:echohl WarningMsg | echo "Don't panic!" | echohl None
```

Здесь первая команда `:echohl WarningMsg` устанавливает тип выделения текста `WarningMsg` (смотрите описание команды `:echohl`). Далее следует команда собственно вывода текста на экран:

```
echo "Don't panic!"
```

Последняя команда `echohl None` убирает выделение текста, которому подвергается весь текст, выводимый посредством команды редактора `echo`.

Если произошла ошибка

Человеку свойственно ошибаться, и при редактировании текста в частности. Например, вы по ошибке удалили не тот кусок текста или вставили не тот кусок текста и не туда. Такое может произойти, если ошибочно двинуть мышку не в том направлении. В результате у вас появился неверный текущий буфер. Однако ошибки редактирования, в отличие от многих ошибок в жизни, почти всегда можно исправить. Что надо предпринять, чтобы ликвидировать или уменьшить последствия таких ошибочных действий?

Во первых, следует перевести редактор vim в обычный режим, нажав `Ctrl+c` или `Ctrl+[`, или `Esc`.

После перехода в обычный режим следует один раз ввести команду `u`.

По этой команде происходит возврат к состоянию текста перед последним изменением. Если необходимо, можно повторить команду `u`.

По умолчанию вы можете повторять команду u 100 раз, то есть вернуться на 100 действий назад. Обычно этого вполне достаточно для ликвидации последствий ошибочных действий.

emacs, xemacs

Введение

Emacs представляет собой мощный экранный редактор текста с массой встроенных функций по изменению текста, созданию командных последовательностей редактирования текста и взаимодействия с операционной системой. emacs можно считать настраиваемой средой для программирования, в которой имеются развитые средства редактирования текста.

По редактору emacs имеется обширная литература на многих языках, включая русский. Основной сайт в Интернете, где имеется последняя версия редактора, — http://www.gnu.org.

Другой редактор, xemacs, является, с точки зрения пользователя, модернизацией emacs — главным образом в использовании графического интерфейса. Несмотря на близость целей разработчиков и свойств упомянутых редакторов, emacs и xemacs — два разных редактора. Хорошим источником информации является сайт http://www.xemacs.org/. Оба редактора обычно являются частью большинства комплектов Linux.

Ниже мы обсудим коротко только emacs.

Технические особенности редактирования в emacs

Emacs написан на диалекте языка Lisp, который именуется Elisp. Хотя Emacs написан на высокоуровневом языке обработки списков, однако не требует знания Lisp для конфигурирования редактора. Тем не менее, если вы планируете создавать нетривиальные алгоритмы преобразования текстов или реализовывать сложные интерфейсы к другим программным системам, то вам стоит ознакомиться с языком Lisp.

Emacs не имеет различных режимов для ввода текста и команд как, например, редактор vim. Если вы вводите с клавиатуры любой изображаемый символ, например, букву «а», то она будет помещена в текстовый документ, который вы видите на экране. Если вы вводите символ `Esc` или `Ctrl`, то это рассматривается редактором как начало какой-то команды, например, `Ctrl+h` (вызов справки).

Редактор сам различает три основных типа ввода:

- обычный ввод текста;
- команда редактора, которая начинается с символа `Ctrl+x`;
- команда редактора, которая начинается с символа `Alt+x`.

Обозначение `Ctrl+x` означает, что одновременно должны быть нажаты две клавиши: клавиша `Ctrl` и клавиша с изображением буквы `x`. Символ `Alt+x` — означает, что одновременно должны быть нажаты две клавиши: клавиша `Alt` и клавиша с изображением буквы `x`. Более подробно ввод команд обсуждается в связи с понятием минибуфера.

Например, для немедленного выхода из редактора следует ввести две специальных комбинации:

```
Ctrl+x Ctrl+c
```

Иными словами, следует одной рукой нажать и держать клавишу `Ctrl` и в то же время другой рукой нажать клавишу `x`, а затем клавишу `c`.

Обсуждаемый редактор весьма интенсивно использует различные комбинации клавиш для обозначения тех или иных действий. Все команды emacs имеют полное имя, то есть имя соответствующей Lisp-функции, которая интерпретирует команду редактирования. Большинство или все общеупотребительные команды редактора имеют соответствующие им сокращения до 1-2 символов с использованием управляющих клавиш `Ctrl` или `Alt`.

Emacs имеет развитую систему описаний. В любой момент вы можете посмотреть любую часть руководства по редактору или ЧАВО (FAQ) командой `Ctrl+h`, после которой вы можете ввести символ «?» (вопросительный знак), чтобы узнать, какие имеются возможности. Например, если вы нажмете `Ctrl+h t`, то на экране появится начальное руковод-

ство по emacs. ЧАВО вызываются командой Ctrl+h F. Можно пользоваться традиционным способом получения текста руководства, например, info emacs. Если вам известен раздел руководства emacs, то вы можете использовать более детальный запрос, например, info emacs abbrev. В табл. 3.12 приведены возможные параметры для команды Ctrl+h.

Таблица 3.12. Параметры команды Ctrl+h

Команда с параметром	Действие
Ctrl+h a	Выполняется команда apropos, то есть производится поиск команды, функции и т. д. по данной подстроке
Ctrl+h b	Выдать таблицу соответствия управляющих кодов клавиш (key binding)
Ctrl+h c	Показать имя функции, которая вызывается по вводу конкретной последовательности с клавиатуры
Ctrl+h C	Показать систему кодирования символов, например, iso-latin-1. Если вы вводите Enter, то будет показана текущая система кодирования
Ctrl+h f	Показать краткое описание функции, имя которой вы введете после Ctrl+h f
Ctrl+h Ctrl+f	Показать описание функции, имя которой необходимо ввести после Ctrl+h Ctrl+f
Ctrl+h i	Вы попадете в систему руководств info
Ctrl+h I	Показать описание системы ввода символов. Вы можете ввести имя системы ввода или Enter. В последнем случае будет выдано описание текущей системы ввода
Ctrl+h Ctrl+i	Показать определение определенного символа (имени), который используется в языке, на котором написан текст в текущем буфере
Ctrl+h k	Показать значение комбинации клавиш, которую необходимо нажать после ввода команды
Ctrl+h l	Показать последние 100 символов, которые вы ввели с клавиатуры
Ctrl+h L	Показать описание языкового окружения определенного типа, если вы его введете после Ctrl+h L, или тип текущего языкового окружения
Ctrl+h m	Показать описание режимов буфера (главные и дополнительные)
Ctrl+h n	Показать новости emacs (историю изменений и/или добавлений в редакторе)
Ctrl+h p	Выдать список программных пакетов, которыми пользуется emacs. Выбрав пакет, можно получить его описание
Ctrl+h s	Получить описание синтаксической таблицы, используемой в настоящее время

Команда с параметром	Действие
Ctrl+h t	Выдать базовое руководство (tutorial) по редактору
Ctrl+h v	Выдать краткое описание переменной, которую вы введете после Ctrl+h v, и ее значение
Ctrl+h w	Вывести имена команд, соответствующие подстроке, введенной после Ctrl+h –w, и показать соответствующие комбинации клавиш (key binding)
Ctrl+h F	Выдать файл ЧАВО
Ctrl+h h	Выдать демонстрационный файл с приветствием на разных языках и использованием разных шрифтов
Ctrl+h Ctrl+c	Вывести правила передачи редактора другим пользователям
Ctrl+h Ctrl+d	Вывести информацию о том, как заказать редактор emacs
Ctrl+h Ctrl+n	Вывести информацию о последних изменениях и/или дополнениях в редакторе
Ctrl+h Ctrl+p	Вывести информацию о проекте GNU
Ctrl+h Ctrl+w	Вывести текст предупреждения об отсутствии всяких гарантий на редактор

Любой вид редактирования в emacs производится в буфере — так называется область компьютерной памяти, которая содержит редактируемый или вновь вводимый текст. Обычный способ использования буфера таков. Текст из какого-то файла считывается в буфер, в нем редактируется, затем содержимое буфера сохраняется в файле по окончании редактирования или во время записи. Emacs может использовать массу буферов для различных целей. Если вы просматриваете и/или редактируете несколько файлов, то каждому файлу будет отведен свой буфер. Отдельный буфер используется для вызова описания редактора.

С каждым буфером связан так называемый *основной* режим, который определяет вид и алгоритм обработки содержащейся в буфере информации. Имеется несколько основных режимов, которые можно разделить на три группы:

1. Режим документов (простой текст, текст в форматах: TeX, LaTeX, nroff и т. д.).
2. Режим программирования (Lisp, Fortran, C и т. д.).
3. Внутренние режимы для специализированных буферов (описания команд редактора и проч.).

Только один основной режим может быть связан с буфером в любой момент времени.

Имеются дополнительные режимы, например, режим *Filling*. Любому буферу могут быть приписаны несколько дополнительных режимов одновременно. К дополнительным режимам относятся:

- *Abbrevs* — режим обработки сокращений (аббревиатур). Этот режим позволяет вам определять сокращения для слов или целых фраз (что позволяет вам, среди прочего, ускорить процесс стенографирования). Этот дополнительный режим включается командой `Alt+ abbrev mode`. Дополнительная информация в `info emacs abbrevs`.
- *Filling* — будучи присвоен буферу, он позволяет вам вводить текст, не задумываясь о размере экрана, там где надо, будет автоматически введен перевод строки, чтобы перейти на следующую строку экрана. Таким образом вы можете избежать слишком длинных строк текста в файле. Дополнительная информация в `info emacs filling`.
- *Auto Save* — если этот режим включен, то периодически производится автоматическое сохранение измененных буферов в специальных файлах редактора. Это позволяет минимизировать потери труда при сбоях. Дополнительная информация в `info emacs "auto save"`.
- *Formatted text* — режим редактирования текста, который имеет команды форматирования, в стиле WYSIWYG, то есть отображая на экране уже форматированный текст и, одновременно, предоставляя средства для изменения текста прямо на экране. В настоящее время (версия 20.5.1) такой режим реализован лишь для формата `text/enriched`, который определен протоколом MIME (Multipurpose Internet Mail Extension — многоцелевое интернетовское почтовое расширение). Этот протокол описывает способы передачи файлов произвольной при-

роды посредством электронной почты. Дополнительная информация в `info emacs "formatted text"`.

- *Spelling* — позволяет отмечать неверно написанные слова и производить коррекции написания. Например, команда редактора `Alt+x flyspell mode` включает проверку написания слов и отметку неверно написанных слов. Смотрите также `info emacs spelling`.
- Имеется еще несколько дополнительных режимов, информацию о которых можно получить посредством `info emacs "minor mode"`.

Emacs можно использовать как на самых простых алфавитно-цифровых терминалах, так и в окне X/Window. На любых видах терминалов emacs позволяет организовать на экране несколько логически независимых окон для редактирования и отображения текста. Окно — это выделенное место на экране, где производится ввод текста и его изменение.

Если вы используете алфавитно-цифровой терминал, то два окна на терминале будут делить между собой один экран, то есть каждое окно будет меньшего размера, чем одно окно на полном экране. Если используется система X/Window, то вы имеете возможность запустить дополнительное окно системы, то есть вы будете иметь два или более окон с такими же свойствами и размерами, как и единственное окно emacs.

Наконец, emacs имеет встроенные интерфейсы к массе разнообразных внешних программ и систем, таких как `info`, CVS, различных трансляторов и т. д. Добавление дополнительных интерфейсов к новым системам и программам не является непреодолимой проблемой, однако, требует знания языка Lisp.

Примеры общеупотребительных команд emacs

Основные команды перемещения курсора по тексту приведены в табл. 3.13. Команды удаления символов приведены в табл. 3.14.

Таблица 3.13. Основные команды перемещения курсора в emacs

Команда	Действие
Ctrl+v	Переместить курсор вперед на размер одного экрана
Alt+m	Переместить курсор назад на один экран
Ctrl+p	Переместить курсор назад на одну строку
Ctrl+n	Переместить курсор вперед на одну строку
Ctrl+b	Переместить курсор назад на один символ
Ctrl+b	Переместить курсор назад на один символ
Ctrl+l	Очистить экран и отобразить текст заново
Alt+f	Переместить курсор на одно слово вправо
Alt+b	Переместить курсор на одно слово влево
Ctrl+a	Переместить курсор на самую левую позицию в строке
Ctrl+e	Переместить курсор на самую правую позицию в строке
Alt+a	Переместить курсор на начало текущего предложения
Alt+e	Переместить курсор в конец текущего предложения
Alt+<	Переместить курсор на начало текущего текстового файла
Alt+>	Переместить курсор в конец текущего текстового файла

Таблица 3.14. Основные команды удаления символов в emacs

Команда	Действие
BS	Удалить один символ слева от курсора
Ctrl+d	Удалить один символ справа от курсора
Alt+BS	Удалить одно слово слева от курсора
Alt+d	Удалить одно слово справа от курсора
Ctrl+k	Удалить все символы справа от курсора до конца строки
Alt+k	Удалить все символы справа от курсора до конца текущей фразы

Если вы хотите отменить последние изменения в тексте, то можно использовать команду `Ctrl+x u` (ввести `Ctrl+x` затем символ `u`). Для тех же целей можно использовать команду `Ctrl+_`. Повторяя ввод команды `Ctrl+x-u`, вы можете вернуться назад более, чем на один шаг редактирования.

Вспомогательные команды редактора

Emacs имеет массу вспомогательных команд. К ним можно отнести:

- Отображение команд редактора в управляющие комбинации клавиш (key binding), свойство аналогичное группе команд `map` в редакторе vim.

- Введение сокращений, которые позволяют значительно сократить количество печатаемого текста в случае повторений длинных фраз или выражений. Вы можете обозначить эти повторяющиеся части текста коротким сокращением. Простейшим способом ввести сокращение может быть следующий:
  - вы вводите длинное название «объектно-ориентированное программирование» с начала строки;
  - далее вводите команды `Ctrl+u 5 Ctrl+x a g`;
  - затем саму аббревиатуру — `ор`. Кстати, в предыдущей команде цифра 5 означает число слов — так `emacs` считает слова в символах кириллицы. Итак, определение аббревиатуры введено, теперь надо включить режим подстановки (если он не был включен) командой `Alt+abbrev mode`.

  Теперь, если вы введете `ор`, после которой появится пробел или возврат каретки, то немедленно вместо `ор` появится:

  `объектно-ориентированное программирование`
- Управление автоматическим смещением строк текста в различных языках программирования и строк комментариев.
- А также множество других мелких полезных элементов, позволяющих уменьшить количество ручного труда при вводе и редактировании текста.

Основные структурные элементы emacs

Перечислим основные структурные объекты, с которыми имеет дело emacs и которые видит и/или имеет дело пользователь. К ним мы отнесем следующее:

- **Окна.** Редактор имеет как минимум одно окна на экране, но может использоваться несколько окон. Количество окон на X-терминале может ограничиваться только вашим вкусом. Каждое новое окно emacs логически независимо от остальных. Если вам требуется закрыть все окна, кроме текущего (на которое указывает курсор), то это выполняется командой `Ctrl+x 1` (ввести цифру 1

после Ctrl+x). Переход к следующему окну осуществляется командой Ctrl+x o (ввести букву o после Ctrl+x). Наконец, закрыть текущее окно можно командой Ctrl+x0 (ввести цифру 0 после Ctrl+x).

- **Каталоги.** Для работы с каталогами имеется специальная часть emacs под названием Dired (directory editor), который предоставляет набор полезных операций для работы с каталогогами. Такая команда может быть введена последовательностью Alt x dired. См. также info emacs dired.
- **Файлы.** Emacs читает текстовые файлы в рабочие области памяти, то есть любые изменения файлов могут последовать только после того, как вы сохранили новое состояние файла.

  Чтобы прочесть файл и отобразить его в текущем окне, следует использовать команду Ctrl+x Ctrl+f. После ввода команды редактор попросит ввести имя файла. Если введенное имя является относительным, то редактор попытается найти его в каталоге по умолчанию. Имя каталога по умолчанию можно установить командой Alt+x cd и посмотреть посредством команды Alt/x pwd.

  Чтобы прочесть файл и отобразить его в другом окне редактора, следует использовать команды Ctrl+x 4 Ctrl+f или Ctrl+x 4 f. Наконец, чтобы открытый файл оставался с правами доступа *только чтение*, следует использовать команду Ctrl+x-Ctrl+r.

  Редактор позволяет скопировать содержимое одного файла в другой файл — Alt x copy file, удалить файл — Alt/x delete file, переименовать файл — Alt/x rename file, создать дополнительное имя файла — Alt/x add name to file, а также создать символическую ссылку к файлу — Alt/x make symbolic link.
- **Буферы.** Буфер — это часть оперативной памяти, которую можно просматривать через окно редактора. Текстовые файлы записываются редактором в буферы. После окончания редактирования следует сохранить содержимое буфера. Это выполняется командой Ctrl+x Ctrl+s.

Если у вас было несколько файлов в процессе редактирования, то полезно использовать команду `Ctrl+x s`. Она сканирует все буферы и спрашивает о каждом измененном, следует ли его сохранить. В обеих командах сохранения используются имена файлов, откуда данные были прочитаны. Если вы собираетесь поменять имя файла, куда планируется записать буфер, то можно воспользоваться командой `Alt/x Ctrl+w`.

- **Мини-буфер.** Emacs использует так называемые мини-буферы для ввода команды, если вводимая команда длиннее одного управляющего символа. Содержимое мини-буфера отображается внизу окна редактора. Ввод длинной команды с клавиатуры в мини-буфер обычно начинается с комбинации `Alt+x`. Если вы желаете аннулировать ввод команды, то следует использовать команду `Ctrl+g`. Повторный ввод этой же команды используется, чтобы остановить уже начавшееся выполнение команды emacs. Наконец, чтобы остановить рекурсивное выполнение редактирования, следует использовать комбинацию `Ctrl+]`.

  Вам может показаться, что имя команды чересчур длинно, например, 10 или 15 символов. Чтобы сократить количество вводимых символов, вы можете назначить какой-то комбинации управляющих символов значение, равное имени этой команды. Можно использовать альтернативное решение: воспользуйтесь клавишей `Tab`, которая, как правило, используется для вывода вариантов завершения команды (или имени файла и т. д.).

- **Указатель.** Указатель (point) — это позиция идеальной точки в выбранном окне редактора между соседними символами. Указатель может быть в самом начале или в самом конце буфера. Каждое окно имеет один указатель. Курсор всегда находится справа от указателя в выбранном окне редактора. Позицию указателя можно посмотреть командой `Ctrl+x =`, а номер текущей строки — командой `Alt/x what line`.

- **Маркер.** Маркер (mark) — это позиция идеальной точки в буфере редактора между соседними символами или в конце буфера. Каждый буфер имеет точно один маркер, но может иметь несколько указателей, поскольку буфер может отображаться в нескольких окнах одновременно. Для каждого буфера редактор поддерживает стек маркеров (mark ring). Когда вы устанавливаете новый маркер, то он помещается в вершину стека маркеров. По умолчанию глубина стека равна 16. Команды установки маркера можно посмотреть в табл. 3.15.
- **Регион.** Регион (region) выбранного окна — это область текста между указателем и маркером. Каждое окно может иметь регион, хотя часть региона может оказаться невидимой в окне из-за своих размеров. Содержимое региона можно записать в файл командой `Alt/x write region` или дописать в конец файла командой `Alt/x append to file`.
- **Регистры.** Регистры представляют собой специальные области памяти, которые позволяют вам хранить данные различной природы: фрагменты текста, значения конфигурационных переменных, имена файлов и прочее. Например, вы можете запомнить текущую позицию в тексте командой`Ctrl+x r R`, то есть запомнить позицию в регистре R. Позже вы сможете перейти к ранее запомненной позиции командой `Ctrl+x r j R`.

  Следует заметить, что содержимое регистров будет утеряно, если вы выйдете из редактора. Чтобы сохранить информацию на длительный срок, превышающий время одной сессии с emacs, то следует пользоваться специальными сохраняемыми регистрами — bookmarks.

  Дополнительная информация может быть найдена командами:

  `Ctrl+h a register`

  `Ctrl+h a bookmark`
- **Переменные.** Emacs имеет набор внутренних переменных (параметров), значения которых редактор использует во время работы. Переменные являются Lisp-пере-

менными. Посмотреть описание любой из них можно командой `Ctrl+h v`. Командой `Alt+x list options` можно посмотреть все параметры редактора. Установка значений переменной выполняется командой `Alt/ x set variable`.

- **Интерфейсы.** Редактор имеет значительное число интерфейсов к разным внешним системам и отдельным программам. Наиболее простые из них — это:
  - система руководств `info`;
  - система многопоточной поддержки версий текстов RCS, CVS;
  - система электронной почты;
  - система компиляции для разных языков программирования;
  - проверка написания слов Ispell;
  - и другие.
- **Теги.** Теги представляют собой элементы текста, например, имена переменных в каком-то языке программирования. Особый интерес представляет случай, когда вы имеете дело с сотнями исходных текстов программ, расположенных в нескольких каталогах. Тогда редактор позволяет вам быстро найти текст, в котором встречается то или другое имя. Emacs находит текст по таблице тегов, которая до этого должна быть построена программой etags (см. страницу `man etags`). Теги имеют такое же назначение, как и в редакторе vim.

  Построение теговской таблицы для emacs выполняется программой etags, которая сходна, но не совпадает с программой ctags. Для того чтобы редактор принял теговскую таблицу к рассмотрению, следует выполнить команду `Alt+x visit tags table`. После того как теговская таблица прочитана, вы можете найти конкретные теги командами `Alt+.` (клавиша `Alt` и точка нажаты одновременно) или `Ctrl+x 4.` (клавиши `Ctrl` и `x` нажимаются одновременно, затем последовательно вводятся цифра 4 и точка). Дополнительная информация может быть получена командой `info emacs tags`.

- **Прямоугольные элементы текста.** Кроме линейного текста, редактор распознает и поддерживает операции с текстами, обладающими геометрической структурой в виде прямоугольника, что помогает при вводе и редактировании различных таблиц. Прямоугольник определяется положением двух своих углов, один из которых определяется указателем, а другой позицией маркера. Вы можете удалить прямоугольник командой `Alt/x delete rectangle` или вставить пустой прямоугольник в текст командой `Ctrl+x r o`, а также многие другие операции (см. `info emacs rectangles`).

Командами редактора все перечисленные выше объекты (кроме интерфейсов) могут быть созданы, удалены, очищены, перемещены, скопированы и т. д.

Таблица 3.15. Команды установки маркеров

Команда	Действие
Ctrl+SPC или Ctrl+@	Установить маркер на месте, на которое указывает указатель. Если вы используете эту команду с предшествующей командой Ctrl+u, маркер будет перемещен к своему предыдущему значению
Ctrl+x Ctrl+x	Произвести обмен позициями указателя и маркера. Таким образом, повторив команду дважды, можно увидеть границы региона
Alt+@	Поместить маркер в конце следующего слова
Alt+h	Поместить указатель в начале параграфа, который окружает или следует за указателем
Ctrl+x Ctrl+p	Поместить указатель в начале страницы, которая окружает или следует за указателем
Ctrl+x h	Поместить указатель в начале буфера и поместить маркер в конце буфера. Таким образом, регион будет в точности соответствовать буферу

Запуск и инициализация emacs

Обычным способом вызова редактора является ввод команды `emacs` или `emacs &`. Поведение редактора после старта находится в зависимости от используемых параметров. В табл. 3.16 приведена краткая характеристика ряда параметров, которые

могут быть использованы при вызове emacs. Подробнее можно посмотреть посредством `info emacs initial`.

Таблица 3.16. Параметры при запуске emacs

Параметр	Действие
file	Прочесть файл с именем file в буфер
-t device | --terminal= device	Определить устройство device в качестве терминала, с которого будет производиться ввод, и на который будет производиться вывод
-d display | --display= display	Использовать в системе X Window терминал с именем display, то есть чаще всего то значение, которое присвоено переменной окружения DISPLAY
-nw | --no-windows	Предполагать, что терминал является алфавитно-цифровым терминалом (не использовать значение переменной окружения DISPLAY)
-batch | --batch	Запустить emacs в так называемом пакетном режиме, который означает, что редактируемый текст не выводится на экран вовсе. Часть сообщений редактора будут выводиться на устройство стандартного вывода диагностики stderr. Такой режим запуска обычно используется для того, чтобы выполнить программы на Lisp, которые надо запустить из сценария или из Makefile. Обычно параметр –batch сопровождается параметром –q
-q | --no-init-file	НЕ использовать файлы .emacs.el и .emacs во время запуска редактора
-u user | --user=user	Во время запуска редактора использовать инициализационный файл пользователя user
--debug-init	Включить отладчик редактора для отладки инициализационного файла

Инициализационный файл /.emacs

Когда редактор emacs стартует, то обычно он загружает Lisp-программу из файла `.emacs` или из файла `.emacs.el` в вашем домашнем каталоге. Инициализационный файл `.emacs` содержит одно или несколько выражений, содержащих вызов функций Lisp, например:

```
(setq fill-column 60)
```

Это выражение означает, что следует вызвать функцию `setq`, чтобы присвоить переменной с именем `fill-column` значение 60. Более подробное обсуждение инициализационного файла можно найти на страницах руководства `info emacs "init file"`.

Потоковый редактор текста sed

Введение

Программа sed не является интерактивным потоковым редактором текста (sequential editor — последовательный редактор). Ее основное назначение — выполнение относительно простых операций преобразования текста, которые можно выполнить за один проход. Вводом для программы sed является файл или устройство стандартного ввода. Выводом является устройство стандартного вывода, то есть sed — фильтр. Команды редактирования могут быть как в командной строке вызова sed, так и в файле, имя которого указано в командной строке. Из команд редактирования могут быть составлены программы.

В данной книге обсуждается GNU-версия sed 3.02.

Вызов sed

Вызов sed производится следующим образом:

```
sed параметры... [сценарий] [input-file...]
```

Следует заметить, что среди передаваемых sed параметров некоторые могут задавать сценарий для исполнения или файл, содержащий такой сценарий. В том и только в том случае, если такие параметры не указаны (то есть не указаны параметры `-e`, `-f`, `--expression`, `--file`), первый непосредственно следующий за параметрами аргумент командной строки рассматривается как сценарий для редактирования входного потока. Оставшиеся после распознавания параметров (и, возможно, сценария) аргументы командной строки трактуются как имена входных файлов. Если в качестве имени входного файла указан знак - (дефис) или же имя файла не указано вовсе, то в качестве входного потока используется стандартное устройство ввода.

Таблица 3.17. Параметры командной строки sed

Параметр	Значение
-V --version	Вывести версию программы sed
-h --help	Вывести краткие сведения о параметрах sed

Параметр	Значение
-n --quiet --silent	По умолчанию sed выводит шаблон в конце цикла прохождения вводного файла. Данный параметр запрещает этот вывод, если в командах редактирования не использован флаг p
-e script --expression= script	Добавить команды редактирования в script к командам, которые будут выполняться для обработки входного потока
-f script-file --file= script-file	Добавить команды, содержащиеся в файле script-file, к командам, которые будут выполняться для обработки вводного потока

Команды редактирования sed

Сценарий редактирования, который интерпретирует sed, состоит из одной или более команд редактирования, которые могут находиться как в командной строке, так и в файле, который указан в командной строке (сценарий указывается при помощи одного из параметров `-e`, `-f`, `--expression`, `--file`). В данном разделе мы будем иметь в виду одну программу или сценарий для редактирования, который образуется конкатенацией всех отдельных команд и сценариев редактирования.

Каждая команда редактирования состоит из указания адреса или адресного интервала в тексте, за которым следуют однобуквенное имя команды и, возможно, дополнительные коды, детализирующие условия выполнения команды.

Адрес в тексте

Для адресации элементов текста используются следующие формы:

- `number` — номер вводимой строки. Данный параметр указывает только одну строку с номером `number`. Заметим, что sed считает номера строк подряд, начиная с первой введенной строки, вне зависимости от количества файлов.
- `first~n` — этот способ задания адреса отсутствует в стандартном sed, но применяется в GNU-расширении. Параметр означает, что данному выражению удовлетворяет каждая `n`-ная строка, начиная со строки `first`. То есть, будут выбраны все строки с номерами вида `first + n*k`, где k — неотрицательное целое число. Например,

чтобы выбрать все нечетные строки, следует использовать выражение `1~2`, а если вы хотите выбрать каждую третью строку, начиная со 2-й строки, то следует использовать адрес `2~3`. Наконец, выражение `45~0` будет означать просто 45-ю строку.

- `$` — знак доллара используется для указания последней из введенных строк.
- `/regexp/` — этот адрес задает все строки, соответствующие регулярному выражению `regexp` (регулярные выражения разбираются в разделе «Поиск по шаблону и регулярные выражения». Если внутри регулярного выражения должен использоваться слэш (`/`), то следует использовать комбинацию `\/`.
- `\%regexp%` — такое выражение также соответствует всем строкам, которые удовлетворяет регулярному выражению `regexp`. Вместо знака `%` можно использовать любой символ. Такая запись оказывается удобнее, если, например, `regexp` содержит много символов `/`.
- `/regexp/I` или `\%regexp%I`. Модификатор `I` (присутствует только в GNU-расширении sed) указывает, что поиск по регулярному выражению `regexp` должен быть независимым от регистра символов (например, строка `qWeRtY` будет соответствовать регулярному выражению `erty`).

Если не указано никакого адреса, то действию команд(ы) редактирования подвергаются все строки входного потока. Если указан адрес, то действию команд редактирования подвергается лишь строка, которая удовлетворяют адресу. Адресный интервал может быть указан как пара адресов, разделенных запятой. Адресный интервал включает все строки, начиная со строки, соответствующей первому адресу (включительно), до строки, соответствующей второму адресу (тоже включительно).

Если второй адрес является регулярным выражением `regexp`, тогда проверка на соответствие начнется со строки, следующей за первым адресом. Если второй адрес есть целое `n`, которое меньше или равно первому адресу, то лишь одна строка будет считаться соответствующей адресному интервалу.

Добавление символа ! (восклицательный знак) в конце адресной спецификации означает отрицание значения соответствия. Таким образом, если знак ! следует после адресного интервала, то такое выражение будет означать все строки, не попадающие в адресный интервал.

Буферы данных sed

sed использует для обработки вводного потока два буфера: основной буфер обработки (pattern buffer) и дополнительный буфер (hold buffer). В обычном режиме sed читает строки вводного потока и помещает в основной буфер; там же производятся операции редактирования. Дополнительный буфер изначально пуст, но ряд команд позволяет перемещать данные из одного буфера в другой.

Часто используемые команды

Комментарий. Знак # используется для обозначения комментария, который продолжается до конца строки. Если вам необходимо заботиться о переносе ваших сценариев на другие операционные платформы, имейте в виду, что в ряде реализаций sed интерпретация строк комментариев может отличаться от описанной здесь. В частности, могут допускаться лишь строки комментариев, содержащие знак # только в крайней левой позиции.

В первой строке сценария sed два символа `#n`, идущие подряд, имеют специальное значение — будет включен режим `no-autoprint`.

Замена. `s/regexp/replacement/flags`

После такой команды sed будет искать части строк входного потока, которые соответствуют регулярному выражению `regexp`, и производить замены этих частей на значение `replacement` в соответствии со значением `flags`.

Значение `replacement` может содержать выражения вида `\n`, где `n` есть целое от 0 до 9. Выражение `\0` обозначает часть строки, соответствующую образцу `regexp`. Если `n` от 1 до 9, то `\n` указывает на `n`-ю по порядку часть соответствия, заключенную в специальные скобки `\(` и `\)`. Также внутри выражения `replacement` может использоваться символ &,

который ссылается на полную строку, соответствующую `regexp` в основном буфере. Если вы хотите использовать в `replacement` символы \ или \ в своем непосредственном значении, то следует экранировать их при помощи знака \.

Приведем несколько примеров использования команды замены.

```
$ echo "Жужжали" | sed -n 's/Жуж/Гуж<&>/p'
Гуж<Жуж>жали
```

Еще пример:

```
$ echo "Жужжали Бабочки" | sed -n 's/\(жж\)\(али\)/
Гуж<&>\2\1/p'
ЖуГуж<жжали>алижж Бабочки
```

Несколько замечаний:

- Символ / может быть заменен на любой другой символ в пределах одной команды `s`.
- Символ / может использоваться внутри выражения `regexp`, если ему предшествует знак \. Для указания внутри `regexp` символа конца строки <NL> используется последовательность \n.

За командой поиска и замены `s` могут следовать (или не следовать) флаги, к обсуждению которых мы и переходим:

- `g` — произвести замену во всех подстроках обрабатываемой строки, соответствующих образцу, а не только в первой найденной;
- `n` — произвести замену только в n-й подстроке обрабатываемой строки, соответствующей `regexp`;
- `w file-name` — если подстановка имела место, то результат записать в файл с именем `file-name`;
- `I` — проверять соответствие образцу вне зависимости от регистра символов (флаг доступен только в GNU-расширении sed);
- `q` — выйти из sed без обработки остальных команд редактирования или входного потока;
- `d` — очистить основной буфер и немедленно начать новый цикл обработки (чтение входного потока, редактирование и т. д.);

- `p` — вывести основной буфер (на устройство стандартного вывода). Эта команда используется обычно в сочетании с параметром `-n` в командной строке.

  Некоторые реализации sed (в том числе описываемая автором GNU-версия) будут печатать все строки входного файла, в которых произведена замена, дважды, если режим `auto-print` не отключен и использован флаг `p`. Другие варианты sed могут вести себя иначе. Оба способа поведения `sed` не являются ошибкой.
- `n` — если режим `auto-print` не выключен, вывести основной буфер на устройство стандартного вывода, затем ввести в основной буфер следующую строку из входного потока. Если входной поток завершен , тогда sed завершает свое выполнение без обработки остальных команд, даже если они есть.
- `{flags}` — группа флагов может быть заключена в фигурные скобки.

Трансляция символов

```
y/source-chars/dest-chars/
```

Эта команда позволяет заменить символы во входном потоке, указанные в наборе `source-chars`, на соответствующие им символы из набора `dest-chars` (то есть, например, третий символ из `source-chars` будет заменен на третий символ из `dest-chars`). Строки `source-chars` и `dest-chars` должны содержать одинаковое число символов.

Приведем пример использования команды `y`:

```
$ echo жужжал | sed y/жуп/gul/ guggal
```

Прочие команды sed

Здесь обсуждаются команды sed, которые не так часто используются в командной строке, но очень полезны при составлении сценариев.

Вставка строк

Существует два варианта команды вставки строк: команды `a` и `i`. Разберем их по очереди.

Команда a разрешает использовать только один адрес. Строки, следующие за этой командой, будут помещены за обрабатываемой строкой. Все строки, кроме последней, должны заканчиваться знаком \. Например:

```
$ cat t
3 a\
Да, вы переполнили \
все \
файловые системы

$ df | awkprint $5, $6' | sed -f t
Use% Mounted 12% /
33% /boot
Да, вы переполнили
все
файловые системы
85% /data03
53% /data04
74% /data05
86% /usr
56% /var
99% /data02
```

Как видим, строки, указанные в файле t после команды a, были выведены после третьей строки входного потока.

Команда i также разрешает использовать лишь один адрес. Ее назначение — немедленно вывести строки текста, следующие за этой командой. Все строки, кроме последней, должны заканчиваться знаком \. Пример использования:

```
 $ cat t
1 i\
Да, вы переполнили \
все файловые системы

$ df | awkprint $5, $6' | sed -f t
Да, вы переполнили
все файловые системы
Use% Mounted
12% /
33% /boot
85% /data03
53% /data04
74% /data05 86%
/usr 56%
/var 99% /data02
```

Как видим, текст, следующий за командой i, выведен до указанной строки. Если адрес не задан, то текст будет выводиться перед каждой строкой.

Замена строк

Команда c требует указания интервала адресов. Она позволяет удалить из входного потока строки, соответствующие адресному интервалу, а вместо них вывести строки, следующие после команды c. Все строки кроме последней должны заканчиваться знаком \. Пример использования:

```
 $ cat t
1,5 c\
Это будет новый заголовок

$ df  | awkprint $5, $6' | sed -f t
Это будет новый заголовок
74% /data05
86% /usr
56% /var
99% /data02
```

Строки с 1-й по 5-ю включительно были заменены на новую строку. Посмотрим, что будет, если не задать адресный интервал:

```
 $ cat t
c\
Это будет новый заголовок

$ df  | awkprint $5, $6' | sed -f t
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
```

Здесь адрес не был указан, и sed заменил каждую строку из входного потока на строку, следующую после команды c.

Вывод номера строки

Для вывода номеров строк используется команда =. Например:

```
 $ df  | awkprint $6' | sed =
1
Mounted
2
/
3
/boot
4
/data03
5
/data04
6
/data05
7
/usr
8
/var
9
/data02
```

Вывод в стандартной форме

Команда l позволяет вывести основной буфер в так называемой «стандартной» форме: вместо неотображаемых символов выводятся их восьмеричные коды, которым предшествует обратный слэш, длинные строки разбиваются на более мелкие (места разбиения будут отмечены обратными слэшами); конец строки помечается символом $. Например:

```
 $ echo "ЖужжалиБабочкиЖужжалиБабочки" | sed -n l
\366\325\326\326\301\314\311\342\301\302\317\336\313\
311\366\325\326\
\326\301\314\311\342\301\302\317\336\313\311$
```

Как можно видеть, sed считает символы кириллицы неотображаемыми.

Вставка файла

Команда r используется для вставки в выходной поток содержимого файла с указанным именем. Если по какой-либо причине файл не может быть прочитан, то не выдается никакого сообщения об ошибке, а файл рассматривается как файл с нулевой длиной. Формат команды: `r file`.

Запись в файл

Записать основной буфер в файл с указанным именем позволяет команда `w`. Формат: `w file`.

Удаление текста

Команда `D` используется для удаления текста из основного буфера (до первого конца строки). Если после этого в основном буфере остался какой-то текст, то sed переходит к его обработке. В противном случае производится переход к новому циклу чтение-обработка.

Разбиение строки

Команда `N` добавляет в основной буфер символ конца строки `<NL>`, а затем помещает туда очередную строку из входного потока. Если входной поток исчерпан, то производится завершение работы sed без обработки оставшихся команд.

Вывод основного буфера

Команда `P` указывает sed вывести порцию основного буфера до первого символа `<NL>`.

Команды для работы с дополнительным буфером

Как мы уже упоминали, в sed имеется ряд команд для работы с дополнительным буфером. Они перечислены ниже:

- `h` — заместить текущее содержание дополнительного буфера содержанием основного буфера;
- `H` — добавить символ `<NL>` к содержимому дополнительного буфера. Затем добавить содержимое основного буфера к содержимому дополнительного;
- `g` — заместить содержимое основного буфера содержимым дополнительного буфера;
- `G` — добавить символ `<NL>` к содержимому основного буфера. Затем добавить содержимое дополнительного буфера к содержимому основного;
- `x` — обменять содержимое основного буфера и содержимое дополнительного буфера.

Команды управления потоком исполнения

Одним из важнейших достоинств sed является наличие команд управления потоком исполнения. Это позволяет использовать sed в специальных ситуациях, когда требуется наличие «языка программирования».

Эти команды перечислены ниже.

`: label` — определить положение метки с именем `label`. (Эту команду не разрешается использовать после адреса.)

`b [label]` — выполнить безусловный переход к метке с именем `label`. Если имя метки опущено, то производится переход к следующему циклу обработки входных строк.

`t [label]` — выполнить условный переход к метке с именем `label`. Переход производится в случае успешного выполнения последней команды `s` (поиск и замена) с момента последнего ввода очередной строки текста и с момента последнего выполнения команды условного перехода. Если имя метки опущено, то производится переход к следующему циклу обработки вводного потока.

Приведем пример простой программы, которая заменяет первый встреченный знак / на букву Щ и завершает работу.

```
 $ cat t
s/\//Щ/p
t Mylabel
b
: Mylabel
q

 $ df | awkprint $6' | sed -n -f t
Щdev/sda6          497636   55293   416643   12% /
```

Поскольку в команде `s` не указан флаг `g`, то в строке будет заменен только первый встреченный слэш. При успешном выполнении замены (если строка содержала слэш) произошел условный переход к метке с именем `Mylabel`, и выполнение sed завершилось. До тех пор пока слэш не встретился, выполнение после команды условного перехода `t Mylabel` не происходит и интерпретируется следующая команда `b`, то есть снова вводится очередная строка вводного потока и над ней производится команда `s/\//Щ/p`.

Другие редакторы текста

pico

pico является очень простым редактором текста с небольшим набором команд редактирования (всего дюжина команд). Все команды видны на экране при запуске редактора. Этим редактором легко пользоваться на терминалах типа vt100. Этот редактор является частью клиентской подсистемы электронной почты pine. Исходные тексты редактора могут быть найдены по адресу: ftp://ftp.cac.washington.edu/mail/pine.tar.Z.

xedit

`xedit` — простой редактор текста с использованием возможностей X Window. Подробная информация может быть найдена на страницах `info xedit`. Редактор входит практически в каждую поставку Linux.

joe

joe — довольно мощный текстовый редактор. Фактически, joe является некоторым «сборным» редактором, который включает в себя привлекательные модели работы с текстом, заимствованные из различных редакторов: pico, WordStar, emacs.

Редактор joe имеется почти в каждом варианте Linux. Его также можно найти по адресу ftp://ftp.std.com/src/editors/joe*.tar.Z.

nedit

`nedit` — весьма развитый редактор текста для среды X Window. Позволяет работать с использованием схемы клиент/сервер. Список рассылки по обсуждению вопросов по редактору — mailserv@fnal.gov. nedit является свободно распространяемым продуктом с открытым кодом. Он доступен по адресу ftp://ftp.fnal.gov/pub/nedit/.

4. Программы анализа, преобразования и печати текста

Текст

Под текстом понимается последовательность символов, которые могут перемежаться управляющими символами: символ табуляции `\t` (напомним, что управляющие символы даются в нотации С), символ перехода на новую строку `\n` и т. п. Текст принято разделять на параграфы, предложения и слова. Обычно пробел разделяет слова между собой, так же как точка — предложения. Во многих редакторах в среде Linux абзацы отделяются один от другого пустой строкой.

С развитием компьютеров все больше и больше текстов готовится с их помощью: книги и брошюры, программы и описания к ним, стенограммы и руководства, даже поговорить можно посредством текста.

Текст можно:

- ввести и отредактировать при помощи текстового редактора;
- записать в виде обычного файла и передать по компьютерной сети;
- проанализировать или преобразовать программой, предназначенной для преобразования или анализа текста.

Поиск по шаблону и регулярные выражения

Целый ряд программ обработки текста в Linux позволяет вам производить поиск, а в некоторых случаях и замену текста по шаблону, построенному с использованием специаль-

ных правил. К таким программам относятся, например, редакторы текстов: ex, vi, ed, sed, а также утилиты семейства grep и многие другие программы-интерпретаторы. Текстовые шаблоны (patterns), которые часто называют регулярными выражениями (regular expressions), содержат обычные текстовые символы вместе со специальными (часто их называют метасимволами).

Толковый словарь Collins Cobuild English Language Dictionary дает следующее толкование слова regular: простое, обычное, повторяющееся без изменения, удовлетворяющее простым правилам. Представляется, что такое толкование исчерпывающим образом описывает смысл термина regular expressions — регулярные выражения.

В этой главе вы найдете следующую информацию:

- список метасимволов;
- описание метасимволов;
- имена файлов и шаблоны;
- примеры.

Метасимволы

В регулярных выражениях используются специальные символы, называемые метасимволами. Эти символы не используются для обозначения самих себя, а имеют специальный смысл. Следующие символы имеют специальное значение только в шаблонах поиска.

Таблица 4.1. Метасимволы в шаблонах поиска

Метасимвол	Значение
. (точка)	Соответствует любому символу, за исключением NULL
* (звездочка)	Соответствует любому количеству (в том числе и нулю) символов, которые предшествует данному. Предшествующий одиночный символ может также быть метасимволом, например, так как точка обозначает любой символ, то выражение .* обозначает любое количество любых символов
^ («крышка»)	Указывает на начало строки

продолжение

Таблица 4.1 *(продолжение)*

Метасимвол	Значение
$	Указывает на конец строки
[...]	Обозначает любой символ из тех, что заключены в квадратные скобки. Дефис (-) внутри скобок может употребляться для обозначения ряда последовательных символов, например, [a-z] обозначает любую строчную латинскую букву. Знак крышки (^) в качестве первого символа в скобках играет роль отрицания: результирующее выражение обозначает любой символ, не входящий в список. Например, [^0-9] – любой символ, кроме цифры. Дефис или правая (закрывающая) скобка в качестве первого символа рассматривается просто как элементы списка
\{n,m\}	Обозначает ряд вхождений символа, который непосредственно предшествует этому выражению. (Предшествующий символ тоже может быть регулярным выражением, заключенным в скобки ( ).) Выражение \n\ обозначает ровно n повторений, выражение \n,\ означает как минимум n повторений, выражение \n,m\ обозначает, что число повторений не меньше n и не больше m. При этом n и m могут быть от 0 до 256
\	Следующий символ обрабатывается специальным образом. Если это метасимвол, то он трактуется как обычный символ
\(...\)	Сохранить подстроку, заключенную между \(и \) в памяти для последующего использования. При разборе одной строки может быть сохранено до девяти подстрок (нумерация происходит слева направо). Сохраненные подстроки могут быть вызваны и использованы в последующих подстановках посредством выражений \цифра
\< \>	Указывает на начало (\<) или конец (\>) слова
+ (плюс)	Обозначает одно или больше появлений предшествующего регулярного выражения
?	Обозначает одно или нуль появлений предшествующего регулярного выражения
|	Операция логического «или» для регулярных выражений. Например, a|(^[bcd]) будет обозначать a, либо один из символов b, c, d в начале строки
(...)	Группировка регулярных выражений. См. пример для предыдущего символа

Последующие символы имеют специальное значение только в заменяющих подстроках.

Таблица 4.2. Метасимволы в шаблонах поиска

Метасимвол	Значение
\	Следующий символ обрабатывается специальным образом. Если это метасимвол, то он трактуется как обычный символ
\цифра	Взять подстроку, сохраненную ранее с помощью выражения \(и \). Цифра задает номер подстроки (от 1 до 9). Самое левое выражение в строке имеет номер 1
& (амперсант)	Использовать искомый текст в текущей заменяющей подстроке
~	Использовать предыдущую заменяющую подстроку вместо текущей
\u	Преобразовать первый символ заменяющей подстроки в верхний регистр
\U	Преобразовать всю заменяющую подстроку в верхний регистр
\l	Преобразовать первый символ заменяющей подстроки в нижний регистр
\L	Преобразовать всю заменяющую подстроку в нижний регистр
\E	Отменить предыдущую команду \U или \L
\e	Отменить предыдущую команду \u или \l

Список метасимволов

Некоторые метасимволы могут иметь одно значение в одних программах и совершенно иное — в других. Программы, в которых используется конкретный метасимвол, отмечены звездочкой в соответствующей графе табл. 4.3.

Таблица 4.3. Список метасимволов для шаблона поиска

Символ	ed	ex	vi	sed	awk	grep	egrep
. (точка)	*	*	*	*	*	*	*
* (звездочка)	*	*	*	*	*	*	*
^ (символ крышки)	*	*	*	*	*	*	*
$ (доллар)	*	*	*	*	*	*	*
\ (обратный слэш)	*	*	*	*	*	*	*
[набор символов]	*	*	*	*	*	*	*
\(шаблон \)	*	*		*			
\{ шаблон \}	*			*		*	
\< шаблон \>		*	*				
+					*		*
?					*		*
|					*		*
( )					*		*

Заметим, что в ed, ex и sed во время выполнения операций редактирования (подстановки подстрок) вы определяете оба параметра: поисковый шаблон (слева) и заменяющая подстрока (справа). Метасимволы в приведенной таблице имеют значение лишь для шаблона поиска.

В ed, ex и sed могут использоваться метасимволы и в заменяющей подстроке (табл. 4.4).

Таблица 4.4. Список метасимволов для заменяющей подстроки

Символ	ex	sed	ed
\ (обратный слэш)	*	*	*
\цифра	*	*	*
& (амперсанд)	*	*	
~ (тильда)	*		
\u или \U	*		
\l или \L	*		
\E	*		
\e	*		

Примеры поиска

Примеры использования регулярных выражений для поиска приведены в табл. 4.5.

Таблица 4.5. Метасимволы в примерах

Шаблон	Значение
big	Последовательность символов big
^big	Последовательность символов big в начале строки
big$	Последовательность символов big в конце строки
^big$	Строка, состоящая только из последовательности символов big
[Bb]ig	Обозначает последовательности big и Big
b[aeiou]g	Обозначает любую из следующих последовательностей: bag, beg, big, bog, bug
b[^aeiou]g	Обозначает все слова или части слов из трех букв, которые между буквами b и g не содержат букв, перечисленных в квадратных скобках: a, e, i, o, u
^...$	Обозначает любую строку, состоящую ровно из трех символов
b.g	Обозначает все строки, в которых буквы b и g разделены любым одиночным символом, например, b2g, big, bug, b?g и т. п.

Шаблон	Значение
^.	Обозначает любую строку, которая начинается с точки
^.]	Обозначает любую строку, которая начинается не с точки. Заметим попутно, что если один и тот же текстовыйфайл просмотреть с использованием программы egrep с только что описанными шаблонами поиска ^\., а затем ^[^.], то мы обнаружим, что суммарное количество найденных строк может оказаться не равно общему числу строк в файле. Такое происходит из-за того, что часть строк файла могут содержать единственный символ NULL
^$	Обозначает пустую строку, которая состоит только из символа NULL
^[a-z][a-z]	Обозначает строку, начинающуюся с точки, за которой следуют две любые буквы в нижнем регистре
[a-z]{2}	То же самое, но только для grep или sed
bg*	Обозначает последовательность символов bo, за которой может идти (а может и нет) несколько букв g
a{3,\}	Только для sed и egrep: три или более нулей подряд
[1-9]\{2\}-[0-9]\{3\}-[0-9]\{1\}	Только для sed и egrep: число, которое имеет вид nn-nnnnn-n

Примеры поиска и замены

Табл. 4.6 показывает, как метасимволы могут использоваться в редакторе sed. Символ подчеркивания (_) обозначает в данном примере пробел. Символ табуляции обозначен как <TAB>.

Таблица 4.6. Метасимволы в примерах

Команда	Действие
s/.*/(&)/	Заменить всю строку на нее же, но в скобках
/^$/d	Удалить пустые строки, то есть строки, содержащие один символ NULL
/^[<TAB>]*$/d	Удалить пустые строки, а также строки, содержащие только пробелы или табуляторы
s/__*/_/g	Превратить два и более пробелов в один пробел
s/Yes/No/	Заменить слова Yes на No

Программы работы с текстом

Люди, уже умеющие программировать, взглянув на описания программ для преобразования текстов могли бы заметить, что алгоритмы преобразования достаточно просты.

Многие смогли бы быстро набросать короткую программу на одном из популярных языков для реализации описанного или своего алгоритма, который им более подходил бы, если им понадобилось преобразовывать тексты. На первый взгляд, написать свою программу преобразования текста, когда это потребуется, — эффективный путь.

Однако если принять во внимание стоимость написания и отладки программы, учитывая реальное время на такую работу, то есть от момента осознания необходимости, до момента, когда программу можно будет уверенно использовать хотя бы небольшим коллективом пользователей, то во многих случаях правильнее будет использовать уже готовые средства преобразования текста. А если учесть возможные проблемы с переносом средств преобразования текста на другие машины или платформы, то очевидность использования уже готовых и многократно проверенных средств станет неоспоримой.

Ниже мы рассмотрим несколько общеупотребительных программ, которые помогают преобразовывать или анализировать текст.

Программа cat

Синтаксис:

```
cat [options] [file]...
```

Программа cat читает каждый файл `file` или стандартный ввод и выводит введенные строки на устройство стандартного вывода. Если перечислено несколько файлов, то все они будут обработаны выведены на устройство стандартного вывода в той же последовательности, в которой были перечислены.

Таблица 4.7. Параметры команды cat

Параметры	Описание
-A --show-all	Эквивалентно комбинации параметров –vET
-b --number-nonblank	Пронумеровать все непустые строки
-e	Эквивалентно комбинации параметров –vE
-E --show-ends	Показать концы строк, поместив знак $ сразу после конца каждой строки

Параметры	Описание
-n --number	Пронумеровать все выводимые строки
-s --squeeze-blank	Заменять несколько идущих подряд пустых строк одной пустой строкой
-t	Эквивалентно комбинации параметров –vT
-T --show-tabs	Показать символы табуляции, отобразив их как ^I
-u	Игнорируется (для совместимости с UNIX)
-v --show-nonprinting	Отобразить управляющие символы, за исключением \n и \tc использованием нотации ^символ. Если в байте установлен старший бит, то символу будет предшествовать M-. Как читатель уже догадался, всем символам русского (кириллицы) алфавита будет предшествовать последовательность M-

Программа tac

Использование:

```
tac [option]... [file]...
```

Программа tac копирует записи (по умолчанию — строки) каждого файла `file` со стандартного устройства ввода на стандартное устройство вывода в обратном порядке. Иными словами, первая запись на вводе будет последней на выводе, а последняя запись на вводе будет первой на выводе. Записи разделяются специальными последовательностями (по умолчанию символом `\n` — конец строки).

Таблица 4.8. Параметры команды tac

Параметр	Действие
-b– --before	Поместить разделитель записей в начале записи
-r --regex	Обработать разделитель строк как регулярное выражение
-s separator --separator =separator	Использовать значение separator как разделитель записей вместо используемого по умолчанию символа \n

Программа nl

Синтаксис:

```
l [option]... [file]...
```

Программа nl записывает строки файлов `file` или строки со стандартного устройства ввода на стандартное устройство вывода. При этом происходит нумерация всех или час-

ти строк на устройстве стандартного вывода. Во время работы программа разделяет ввод на логические страницы. По умолчанию номера строк начинаются с 1 на каждой логической странице. В то же время, если на ввод подаются несколько файлов, то nl рассматривает их все как единый поток ввода (единый документ).

Логическая страница состоит из трех частей: заголовок, тело страницы, подстрочные замечания. Любая из перечисленных частей может быть пустой. Начало каждой из частей страницы отмечается во входном файле с помощью отдельной строки, содержащей один из нижеследующих разделителей:

- `\:\:\:` — начало заголовка страницы;
- `\:\:` — начало тела страницы;
- `\:` — начало подстрочных замечаний.

Вместо `\` и `:` при помощи описанных ниже параметров могут использоваться другие символы. Однако вид и длина разделителей не могут быть изменены.

Сами строки с разделителями частей страницы при выводе заменяются пустыми строками. Любой вводимый текст, который следует до первого разделителя частей, рассматривается как часть тела страницы. Таким образом, nl рассматривает текст, который не содержит никаких описанных здесь разделителей, как одно тело одной логической страницы.

Программа nl воспринимает следующие параметры:

- `-b style` или `--body-numbering=style` — выбрать стиль нумерации для строк в теле страницы для каждой логической страницы. Когда строка не нумеруется при выводе, то номер строки не увеличивается, однако в ненумерованных строках в начале строки помещается разделитель, который отделяет обычно номер от строки. Стили могут быть следующими:
  - `a` — нумеровать все строки, включая заголовки и строки подстрочных замечаний (которые по умолчанию не нумеруются);
  - `t` — нумеровать только непустые строки (значение по умолчанию для тела страницы);

- `n` — не нумеровать строки (значение по умолчанию для заголовка и подстрочных замечаний);
- `pregexp` — нумеровать только те строки, которые содержат подстроки, удовлетворяющие регулярному выражению `regexp`.

- `-d cd` или `--section-delimiter=cd` — использовать для разделителя частей символы `cd`]; по умолчанию используются символы `\:`. Если в параметре дан только один символ `c`, то вторым символом остается `:`. Обратите внимание, что когда вы передаете обратный слэш или другие специальные знаки в команде из оболочки, то необходимо предусмотреть заключение их в кавычки во избежание интерпретации данных специальных знаков самой оболочкой.
- `-f style` или `--footer-numbering=style` — работает так же, как и параметр `--body-numbering`, но по отношению к подстрочным замечаниям.
- `-h style` или `--header-numbering=style` — работает так же, как и параметр --body-numbering, но по отношению к заголовку.
- `-i number` или `--page-increment=number` — увеличивать номер строки на величину `number`. По умолчанию `number=1`.
- `-l number` или `--join-blank-lines=number` — выбрать формат нумерации строк:
  - `ln` — номера строк выровнены по левому краю, ведущих нулей в написании номеров нет;
  - `rn` — номера строк выровнены по правому краю, ведущих нулей в написании номеров нет (значение по умолчанию);
  - `rz` — номера строк выровнены по правому краю, используются ведущие нули в написании номеров строк.
- `-p` или `--no-renumber` — не сбрасывать номер строки с началом каждой логической страницы (по умолчанию происходит сброс номера строки).

- `-s string` или `--starting-line-number=string` — разделить при выводе номер строки и саму строку последовательностью символов `string` (по умолчанию используется символ табуляции).
- `-v number` или `--starting-line-number=number` — установить начальный номер нумерации на каждой логической странице равным `number`. По умолчанию `number=1`.
- `-w number` или `--number-width=number` — использовать номера строк из `number` цифр. По умолчанию `number=6`.

Программа od

Синтаксис:

```
od [option]... [file]...
od -C [file] [[+]offset [[+]label]]
```

Вывод версии программы:

```
od --version od (GNU textutils) 1.22
```

Программа od выводит на устройство стандартного вывода явное (бинарное) представление каждого файла `file` (или информации, полученной из стандартного потока ввода, если вместо имени файла указан знак `-`).

Каждая строка вывода состоит из относительного адреса во вводном файле, за которым следуют группы данных из файла. По умолчанию od выводит адрес в восьмеричном виде, а каждая группа данных из файла представляет собой два байта вводного файла, выведенных как одно восьмеричное число.

Ниже перечислены параметры программы:

- `-A radix` или `--address-radix=radix` — выбрать систему счисления, в которой будет выводиться адрес. Параметр `radix` может принимать следующие значения:
  - `d` — десятичное основание системы счисления;
  - `o` — восьмеричное основание системы счисления (значение по умолчанию);
  - `x` — шестнадцатеричное основание системы счисления;
  - `n` — адреса не выводятся.

- `-j bytes` или `--skip-bytes=bytes` — пропустить `bytes` байтов на вводе до начала вывода. Если `bytes` начинается с `0x` или `0X`, то число `bytes` интерпретируется как шестнадцатеричное. Если `bytes` начинается с `0`, то число интерпретируется как восьмеричное. В остальных случаях оно рассматривается как десятичное число. Если `bytes` заканчивается буквой `b`, то `bytes` умножается на 512; буквой `k` — на 1024; буквой `m` — на 1048576.
- `-N bytes` или `--read-bytes=bytes` — вывести `bytes` байтов максимум. Префиксы и суффиксы `bytes` интерпретируются так же, как при использовании параметра `-j`.
- `-s[n]` или `--strings[=n]` — вместо обычного вывода напечатать лишь строковые константы, далее как минимум `n` последовательных символов (по умолчанию 3).
- `-t type` или `--format=type` — выбрать формат, в котором производится вывод данных. Строка `type` может содержать один или более символов, каждый из которых определяет отдельный вид вывода. Если строка состоит из более, чем одного символа, то каждая строка вводного файла будет выводиться столько раз, сколько заказано видов вывода и в том порядке, который был указан в строке `type`. В качестве параметра `type` могут использоваться следующие символы:
  - `a` — выводятся именованные символы, то есть все специальные знаки выводятся с помощью их имен, например, пробел выводится как `<SP>`;
  - `c` — выводятся символы ASCII (если есть соответствующий отображаемый символ) или их коды;
  - `d` — представление в виде знакового десятичного числа;
  - `f` — представление в виде десятичных чисел в формате с плавающей точкой;
  - `o` — восьмеричное представление;
  - `u` — десятичное представление без знака;
  - `x` — шестнадцатеричное представление.

Исключая типы `a` и `c`, вы можете определить число байтов, которые вы хотели бы использовать для интерпретации данных каждого типа. Число байтов определяется десятичным целым, которое следует за символом типа данных. Поскольку допускаемые значения целых должны соответствовать размерам типов данных для языка C (`C` — char, `S` — short, `I` — int, `L` — long; для чисел с плавающей точкой: `F` — float, `D` — double, `L` — long double), то разумнее их и употреблять. Например, `date | od -t cxC`. Будет выведено нечто похожее на нижеследующее:

```
0000000  Fri Jul  2 17: 05
         46 72 69 20 4A 75 6C 20 20 32 20 31 37 3A 30 35
0000020     :50 MSD 1999\n
         3A 35 30 20 4D 53 44 20 31 39 39 39 0A
0000035
```

- `-v` или `--output-duplicates` — включить вывод последовательных одинаковых строк. По умолчанию od выведет только первую строку, а вместо остальных — только звездочку.

Семейство программ grep

Семейство программ grep позволяет вам находить в одном или в группе файлов все строки, которые содержат цепочки символов, удовлетворяющие данному регулярному выражению или шаблону (подробнее о регулярных выражениях см. раздел «Поиск по шаблону и регулярные выражения»). Само имя grep означает Global Regular Expresion Print — печать регулярных выражений общего вида. Программы этого семейства можно использовать как фильтры, то есть извлекать подходящие под регулярное выражение строки из стандартного ввода и посылать их на устройство стандартного вывода.

Семейство содержит три программы, которые отличаются друг от друга скоростью работы, а также набором допустимых регулярных выражений.

- Программа grep производит поиск символьных цепочек, соответствующих представленному шаблону. Имеется три варианта работы grep, которые можно выбрать, используя параметры:

- `-G` или `--basic-regexp` — интерпретировать шаблон как базовое регулярное выражение. Этот режим используется по умолчанию.
- `-E` или `--extended-regexp` — интерпретировать шаблон как расширенное регулярное выражение.
- `-F` или `--fixed-strings` — интерпретировать шаблон как список фиксированных символьных цепочек, разделенных символами `NULL`. Если любая цепочка из списка встречена в файле, то соответствующая строка файла отправляется в стандартный поток вывода.

- Программа egrep весьма похожа на программу grep, вызванную с параметром `-E`, но не идентична ей полностью.
- Программа fgrep есть то же, что и `grep -F`.

Все программы семейства grep воспринимают параметры, сведенные в табл. 4.9.

Таблица 4.9. Параметры программ семейства grep

Параметр	Значение
-num	Удовлетворяющие шаблону строки будут напечатаны с num предшествующими строками и num последующими. В этом случае grep не будет печатать никакую строку более, чем один раз
-A num --after-context=num	Вывести на устройство стандартного вывода num строк после каждой найденной
-B num --before-context=num	Вывести на устройство стандартного вывода num строк перед каждой найденной строкой
-C --context	То же самое, что и –2
-V --version	Вывести информацию о версии программы
-b --byte-offset	Вывести относительный адрес байта в файле для каждой выводимой строки
-c --count	Запретить обычный вывод программы. Вместо него вывести число строк, которые удовлетворяют шаблону. Если этот параметр используется совместно с параметром -v (см. ниже), то будет выводиться число строк, не удовлетворяющих шаблону
-e pattern --regexp=pattern	Использовать pattern в качестве шаблона. Эта запись используется, когда шаблон начинается со знака -
-f file --file=file	Взять шаблон из файла с именем file
-h --no-filename	Запретить указание имен файлов на выводе, если производится поиск по шаблону в нескольких файлах. По умолчанию будут указаны имена файлов, в которых найдены строки, соответствующие шаблону

продолжение

Таблица 4.9 *(продолжение)*

Параметр	Значение
-i --ignore-case	Игнорировать различие в регистре символов как в шаблоне, так и во вводных файлах
-L --files-without-match	Запретить обычный вывод. Вместо этого напечатать имя каждого последовательно проверяемого файла, в котором не найдено строк, соответствующих шаблону. Операция будет остановлена, как только будет найдена первая строка, соответствующая шаблону, или когда будет исчерпан список файлов
-l --files-with-matches	Запретить обычный вывод. Вместо него вывести имя каждого файла, в котором имеются строки, удовлетворяющие шаблону
-n --line-number	Совместно с самими строками, удовлетворяющими шаблону, вывести их номера в файле
-q --quiet	Запретить любой вывод. Поиск строк, соответствующих шаблону, останавливается на первой же найденной строке. Если строка найдена, устанавливается код завершения 0, в противном случае – 1
-s --silent	Запретить сообщения об ошибках (если встретились несуществующие или нечитаемые файлы, например, нет доступа для чтения)
-v --revert-match	Вывести строки, которые не соответствуют шаблону
-w --word-regexp	Вывести лишь те соответствия шаблону, которые совпадают с границами слова. Слово может содержать буквы, цифры и символ подчеркивания
-x --line-regexp	Вывести лишь те соответствия, которые представляют собой целую строку
-y	Устаревший синоним для параметра -i

Несколько примеров использования grep

Для начала выведем версию программы grep:

```
$ grep -V
grep (GNU grep) 2.1
Copyright (C) 1988, 92, 93, 94, 95, 96, 97 Free Software
Foundation, Inc. This is free software; see the source
for copying conditions. There is NO warranty; not even
for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Найдем в файле `Grep.tex` все строки, которые содержат последовательность `-i`:

```
$ grep -e -i Grep.tex
  [-i]
[-y] устаревший синоним для параметра -i.
```

Найти в файле все строки, которые содержат в первой слева позиции обратный слэш:

```
grep ^\\\\ Grep.tex
```

Команда:

```
grep -v ^\\\\ Grep.tex
```

выведет на экран все строки, которые не содержат знака обратный слэш в первой слева позиции. Того же результата можно достичь и другим способом:

```
cat > Pattern
\\
trl+D
grep -vf Pattern Grep.tex
```

В последнем примере шаблон для поиска помещен в файл с именем Pattern. При этом вы могли обратить внимание, что использовано разное количество обратных слэшей. Дело в том, что в первом примере обратные слэши сначала были обработаны оболочкой bash (при этом каждые два слэша были заменены на один), а затем — программой grep.

Программа fmt

Программа fmt производит простое форматирование текста, которое в основном заключается в наиболее полном заполнении строк.

Синтаксис:

```
fmt [option]... [file]...
```

Программа читает текст либо из файла, либо со стандартного устройства ввода, а выводит сформированный текст на стандартное устройство вывода. По умолчанию пустые строки, пробелы между словами и пустые места в начале строк сохраняются и на выводе. Вводимые строки с пробелами в начале строк не объединяются. Символы табуляции расширяются обычным образом при вводе и используются затем при выводе.

Программа fmt предпочитает разбивать строки в конце предложения. Она пытается избежать разрыва строки после первого слова предложения и перед последним словом

предложения. Конец предложения определяется при помощи одного из двух условий:

- Достигнут конец параграфа, то есть встречена пустая строка или символ `NULL`.
- Слово закончилось одним из знаков `.?!` (точка, вопросительный знак, восклицательный знак), за которыми следует два пробела или конец строки. Алгоритм разбиения параграфа на строки является вариантом алгоритма, описанного в статье «Breaking Paragraphs Into Lines» (авторы: Donald E. Knuth и Michael F. Plass, журнал «Software — Practice and Experience», 11 (1981), 1119-1184).

Программа воспринимает следующие параметры:

- `-c` или `--crown-margin` — сохранить без изменения сдвиги первых двух строк параграфа и выровнять левый край всех последующих строк параграфа по левому краю второй строки.
- `-t` или `--tagged-paragraph` — этим параметром устанавливается режим форматирования tagged-paragraph. Этот режим похож на режим crown-margin за исключением того, что если сдвиг первой строки параграфа тот же самый, что и сдвиг второй строки параграфа, то первая строка обрабатывается как однострочный параграф.
- `-s` или `--split-only` — только разделять строки. Не объединять короткие строки, чтобы сформировать более длинные.
- `-u` или `--uniform-spacing` — унифицировать пробелы, то есть уменьшить число пробелов между словами до одного пробела, а число пробелов между предложениями — до двух пробелов.
- `-width` или `-w width` или `--width=width` — Заполнить выводные строки до ширины `width` (по умолчанию — 75 символов).
- `-p prefix` или `--prefix=prefix` — форматированию будут подвергнуты только строки, которые начинаются с `prefix` (перед ним могут быть пробелы). Сам `prefix` и

предшествующие ему пробелы будут удалены из строки перед форматированием. После выполнения форматирования, при выводе результата в начало строк будет помещен `prefix`. Одно из полезных применений данного параметра — форматирование программных комментариев. Следует лишь иметь в виду, что если вы планируете отформатировать комментарии сценариев, то знак комментария полезно задать в кавычках, например,

```
$ fmt -p "#" script;
```

Программа pr

Эта программа позволяет выполнить печать текста с разделением его на страницы и колонки.

Синтаксис:

```
pr [option]... [file]...
```

По умолчанию на каждой странице печатается 5-строчный заголовок: две пустые строки, строка, содержащая дату, имя файла, число страниц, затем две пустые строки. Внизу страницы также выводится пять пустых строк. При использовании параметра `-f` печатается только трехстрочный заголовок: две начальные пустые строки опускаются, вывод пустых строк внизу страницы также опускается. По умолчанию высота страницы в обоих случаях составляет 66 строк. Текстовая строка заголовка занимает всю ширину страницы и имеет следующий вид:

```
yy-mm-dd HH:MM file_name Page nnnn
```

Строка центрирована. Знаки перевода формата на вводе (form feed) — переход на следующую страницу.

Колонки имеют одинаковую ширину и отделяются друг от друга последовательностью символов (по умолчанию пробелами). Длина выводных строк ограничивается (по умолчанию 72 символа), если не использован параметр `-j`. По умолчанию вывод в одну колонку не ограничивается по длине строк; чтобы ввести в действие ограничение по длине строк можно использовать параметр `-w`.

Параметры программы сведены в табл. 4.10:

Таблица 4.10. Параметры программы pr

Параметр	Значение
+first_page[:last_page]	Начать печать со страницы first_page и закончить на странице last_page. Имеются в виду реальные страницы, а не их номера. Отсутствие параметра last_page означает, что на печать будет выводиться вся вводимая информация до конца вводного файла. Символ <FF> \f на вводе рассматривается как переход к новой странице. Несколько таких символов подряд вызовут выдачу на печать пустых страниц, что засчитывается при определении страницы first_page
-column	С каждым файлом file, который читается программой pr, производится следующее: ▪ Производится определение ширины колонок исходя из величин page_width и column (все колонки одинаковой ширины) ▪ При считывании строк файла, если они оказались больше ширины колонки, строки усекаются по длине до величины ширины колонки (правая часть строки отбрасывается) и помещаются в колонку до заполнения колонки (сверху вниз). Затем начинается новая колонка ▪ Если страница заполнена, происходит переход к следующей странице. Если встречен конец вводного файла или символ смены /f, а страница не заполнена, то производится балансировка колонок, чтобы все колонки были заполнены примерно одинаково
-a	При многоколонном выводе заполнять колонки по очереди, то есть заполнять вначале первую строку первой колонки, затем первую строку второй колонки и т. д.
-c	При выводе печатать управляющие символы с использованием специальных обозначений, например, ^C. Непечатаемые символы отображаются в восьмеричном виде (каждому символу предшествует обратный слэш). Нетрудно заметить, что таким образом можно выдать на печать файлы произвольного вида, а не только текстовые
-d	При выводе заменять одиночный \n на двойной
-e[in-tabchar [in-tabwidth]]	При выводе заменить знаки табуляции на соответствующее число пробелов. Возможный аргумент in-tabchar представляет собой символ табуляции (по умолчанию \t). Возможный аргумент in-tabwidth – целое число, расстояние между соседними позициями табуляции (по умолчанию 8)
-f -F	Использовать символ \f вместо символа \n для разделения строк в исходном файле. При этом размер страницы в строках остается неизменным (по умолчанию 66 строк), но число строк текста на странице увеличивается с 56 до 63

Параметр	Значение
-h header	Заменить имя файла в заголовке строкой header. Программа может обрезать строку header, если она окажется слишком длинной. При использовании параметра -h "" будет выводиться пустой заголовок. Пробел между параметром -h и аргументом весьма важен, его опускать нельзя
-i[out-tabchar [out-tabwidth]]	Заменить знаки табуляции на выводе соответствующим числом пробелов. Возможный аргумент out-tabchar указывает символ табуляции (по умолчанию используется \t). Возможный аргумент out-tabwidth указывает расстояние между соседними позициями табуляции (по умолчанию 8)
-j	Печатать строки полной длины. Не производится укорачивания строк, даже если они выходят за пределы колонок; при этом не предпринимается никаких мер по выравниванию колонок. Этот параметр может быть использован совместно с параметрами -column, -a, -m или -s[separator]
-l page_legth	Установить длину страницы в строках равной значению page_legth (по умолчанию 66 строк). Если величина page_legth меньше или равна 10 (меньше или равно 3 с использованием параметра -f), то заголовки и подстрочные заголовки будут опущены, а все символы /f на вводе будут игнорироваться (как при использовании параметра -T)
-m	Объединить и напечатать все файлы представленные в виде аргументов программы pr параллельно, по одному файлу в каждой колонке. Если строка слишком длинна, чтобы поместиться в колонке, то она усекается. Отсеченная часть отбрасывается. Этот параметр может использоваться совместно с параметрами -j, -s[separator]
-n[number-separator [digits]]	В каждой колонке добавить в начале номер строки. При использовании параметра -m в каждой строке будет добавлен лишь один номер для всех колонок. Возможный параметр number-separator задает символ, который помещается между номером строки и самой строкой (по умолчанию \t). Возможный параметр digits определяет количество цифр в номере строки (по умолчанию 5). Счет начинается с первой введенной строки, а не с первой напечатанной строки (см. также -N)
-N line_number	Начать счет строк с номера line_number для первой строки первой выведенной страницы
-o n	Сдвинуть каждую строку на n позиций вправо (по умолчанию – не сдвигать). Общая ширина вывода составит n плюс ширина, установленная параметром -w
-r	Не печатать сообщений об ошибках, если файл file не может быть открыт
-s[separator]	Разделить колонки в выводном тексте строкой separator. Пробела между параметром -s и значением аргумента быть не должно

продолжение

Таблица 4.10 *(продолжение)*

Параметр	Значение
-t	Прекратить печать заголовков и номеров страниц и не печатать нижние пустые строки. Таким образом, структура страниц на печати не формируется, однако символы \f остаются неизменными. Параметр -t отменяет значение параметра -h, если последний был использован
-T	Не печатать верхние и нижние заголовки страницы, а также удалить все символы \f в выводном файле
-v	Выводить непечатаемые символы в виде их восьмеричного представления
-w page_width	Установить ширину страницы в значение page_width (по умолчанию 72). Строка заголовка всегда усекается до значения page_width (без параметра –w до значения по умолчанию), а строки текста усекаются только с использованием -w и без параметра –j.– Если не используются параметры управления колонками и не используется параметр -w, то это эквивалентно использованию параметров –w 72 -j

Программа fold

Программа fold предназначена для разделения длинных строк.

Синтаксис:

```
fold [option]... file]...
```

Программа fold читает каждый файл `file` (или устройство стандартного ввода) и производит разбиение длинных строк так, чтобы они укладывались в допустимую ширину строки вывода. По умолчанию `fold` разбивает строки, которые оказались длиннее 80 символов. Вводимая строка разбивается на столько строк, на сколько это необходимо.

Используемый по умолчанию алгоритм определения размера выводной колонки таков:

- `\t` (табулятор) означает более, чем одну колонку;
- `\b` (возврат на символ) означает, что счетчик размера выводной колонки уменьшается на 1;
- `\r` (возврат каретки) обнуляет счетчик ширины выводимой строки.

Программа fold воспринимает следующие параметры:

- `-b` или `--bytes` — подсчитывать байты, а не позиции символов. Таким образом, символы `\t`, `\b`, `\r` рассматриваются точно так же, как и остальные символы.
- `-s` или `--spaces` — разбивать вводимые строки на границе слов, то есть строка будет разбиваться после пробела, но так, чтобы не превысить максимальную допустимую ширину строки вывода. Если строка не содержит пробелов, то она разбивается в точке максимально допустимого размера выводной строки. По умолчанию, при разбиении строки границы слов (символы-разделители) не принимаются в расчет.
- `-w width` или `--width=width` — установить максимальную допустимую ширину выводной строки в `width` символов (по умолчанию — 80).

Подсистема печати текста a2ps

Введение

Программа a2ps предназначена для подготовки текстовых файлов (иначе говоря, документов) для печати в формате PostScript: вывод производится либо на устройство печати, либо в файл.

Используемый по умолчанию план страниц довольно компактен и информативен: возможно учесть особенности печати на обоих сторонах листа и т. д.; каждая страница снабжена специальной рамкой; заголовки содержат имя файла, дату преобразования и другую информацию.

Поскольку программа имеет большое число параметров, которые позволяют производить весьма тонкую настройку вида печати в зависимости от конкретных особенностей вводимого текста, то a2ps следует рассматривать как гибкую настраиваемую подсистему преобразования произвольных текстов (программ на разных языках программирования, описаний с использованием национальных языков) в формат PostScript.

Конфигурационные возможности a2ps таковы, что эту программу можно использовать с учетом конкретных особенностей документов почти любого вида, а также предусмотреть подключение любых других программ форматирования или проверки документов.

Программа a2ps постоянно дополняется и совершенствуется. При подготовке данного описания автор ориентировался на версию 4.12. Последнюю версию программы можно найти по вдресу http://www.inf.enst.fr/~demaille/a2ps/. Вместе с программой поставляется довольно обширное описание, которое легко посмотреть посредством утилиты `info`.

Также полезно знать, что при помощи команды `a2ps --help` можно вывести отличную справочную информацию о программе.

Вопросы по программе, а также комментарии могут быть направлены авторам: Miguel Santana,Miguel.Santana@st.com и Akim Demaille demaille@inf.enst.fr. Имеется список рассылки a2ps-request@inf.enst.fr, а также адрес, по которому следует сообщать об обнаруженных в программе ошибках: a2ps-bugs@inf.enst.fr.

Параметры командной строки программы a2ps

Таблица 4.11. Параметры командной строки программы a2ps

Параметр	Описание
-a[page-range] --pages=[page-range]	По умолчанию печатаются все страницы, однако имеется возможность явно указать страницы, которые должны быть напечатаны. Например, -a1 означает, что следует напечатать только первую страницу; -a-5,7,13,43,179- означает, что печатаются первые 5 страниц, затем страницы с номерами 7, 13 и 43, а далее все страницы до конца документа, начиная со страницы с номером 179. Имейте в виду, что речь идет о логических, а не о физических страницах. Если вы используете, например, комбинацию параметров –2 -a1, то вы получите наполовину заполненную физическую страницу. Заметим, что явное указание номеров печатаемых страниц работает так же и в режиме делегирования (см. параметр –Z и --delegate=boolean и раздел «Делегирование»)

Параметр	Описание
-c --truncate-lines= boolean	Усечь строки, которые оказались слишком длинны, чтобы быть помещенными внутрь рамки. Максимальный размер строки зависит от используемого размера шрифта, формата символов, а также наличия или отсутствия нумерации строк
-i --interpret=boolean	Интерпретировать символы \t и \f. Это означает, что \f приводит к переходу на следующую логическую страницу, \t приводит к переходу на следующую колонку табуляции
--end-of-line=type	Определить, какая последовательность символов будет восприниматься как конец строки, type может принимать следующие значения: ■ n или unix, что эквивалентно \n; ■ rc или rn, что означает \r\n, то есть обозначение конца строки, принятое в MS DOS; ■ r или mac, что указывает обозначение конца строки, принятое в Mac OS – \r ■ nr, для использования в качестве конца строки последовательности символов \n\r; ■ any или auto позволяет указать программе, что требуется автоматически выбрать последовательность (одну из указанных выше), используемую в качестве символа конца строки
-X key --encoding=key	Парамер задает кодировку входного потока при помощи аргумента key, который может принимать такие значения, как ASCII, latin1 и т. п. Полный список поддерживаемых кодировок (и доступных значений key) можно получить при помощи команды a2ps--list=encodings
--stdin=filename	Указать в качестве имени файла для входного потока с устройства стандартного ввода имя filename (имя файла будет использоваться в колонтитулах напечатанных страниц)
-t name--title=name	Дать документу имя name. (Это имя документа, а не имя выводного устройства.)
--prologue=prologue	Использовать пролог с именем prologue в качестве PostScript-пролога для программы a2ps. Файл пролога должен иметь имя prologue.pro и находиться в библиотечном каталоге программы a2ps. Аргумент prologue может принимать следующие значения: bold – пролог для черно-белой печати, но все символы будут полужирными; bw – пролог для черно-белой печати, используются шрифты по умолчанию; color – пролог для печати с использованием для выделения цветом ключевых слов; gray – пролог для печати с использованием для выделения комментариев и меток полутоновых символов; gray2 – вариант предыдущего пролога с использованием несколько более темных символов; matrix – пролог для черно-белой печати, однако текст будут располагаться на чередующихся серых и белых горизонтальных полосах, которые удобны при просмотре больших таблиц

продолжение

Таблица 4.11 *(продолжение)*

Параметр	Описание
--print-anyway=boolean	Если аргумент boolean равен 1, то производится печать двоичных файлов. По умолчанию если встречен двоичный файл, то печать прекращается. Программа a2ps определяет тип файла по количеству неотображаемых символов (если файл содержит более 40% неотображаемых символов, то он считается двоичным). Если команда file показывает, что файл имеет тип data, то файл также полагается двоичным и, как следствие, не печатается. Параметр --print-anyway=boolean позволяет преодолеть это ограничение и файл будет напечатан
-Z --delegate=boolean	Разрешить или запретить процесс делегирования. Если делегирование разрешено (--delegate=1), то программа a2ps произведет вызов подходящей специализированной программы, делегируя ей полномочия по преобразованию вводного файла. Если делегирование запрещено (--delegate=0), то программа a2ps будет обрабатывать сама каждый вводной файл. Разницу в поведении программы легко понять, если вы попытаетесь напечатать с помощью программы a2ps файл в формате PostScript. Если процесс делегирования разрешен, то вы получите обычную печать файла. А если процесс делегирования запрещен, то вы получите символьное изображение внутреннего формата PostScript вашего файла, то есть весьма длинную и многостраничную печать, которая окажется малоинформативной для вас, если вы не являетесь экспертом по языку PostScript. Список программ, используемых a2ps при делегировании, можно получить при помощи команды a2ps --list=delegations (см. раздел «Делегирование»)
--toc[=format]	Генерировать оглавление. Аргумент format обрабатывается как файл типа PreScript (см. раздел «PreScript»). Если аргумент format опущен, то оглавление не генерируется.

Простые примеры использования a2ps

Для того чтобы оценить, как будет выглядеть на печати текст из файла `text.file`, используйте следующую команду:

```
$ a2ps -P display text.file
```

Параметр `-P` указывает тип принтера, который должна использовать программа. Тип `display` означает, что вывод будет передан программе просмотра PostScript-файлов `ghostview`, а не отправлен на принтер.

Если вы хотите просто узнать, сколько страниц займет при печати ваш документ, но не хотите печатать его, то это можно сделать так:

```
$ a2ps -P void text.file
```

Наконец, если требуется сохранить PostScript-файл, приготовленный для печати, то поможет тип принтера `file`:

```
$ a2ps -P file text.file
```

В ответ программа напечатает примерно следующее:

```
future76:shevel /usr/home/shevel> a2ps -P file Text.tex
[Text.tex (TeX): 2 pages on 1 sheet]
[Total: 2 pages on 1 sheet] saved into the file Text.ps'
```

Тот же результат можно получить, используя команду `a2ps Text.tex -o My.text.ps`.

Возвращаясь к виду принтера, можно заметить, что имена принтеров, за исключением уже перечисленных `display`, `void` и `file`, обозначают имя принтера в операционной системе.

Если вы имеете достаточно узкий текст, например, текст программы, то можно разместить текст в две колонки при помощи команды `a2ps -3 Text.tex`. Параметр `-3` указывает, что на одной физической странице следует разместить три страницы текста.

Программа a2ps имеет множество параметров. Чтобы узнать, какие значения параметров используются по умолчанию, используйте команду `a2ps --list=defaults`.

Делегирование

Если программа a2ps полагает, что определенное преобразование текста успешнее выполнит другая программа, то a2ps производит обращение к ней. Этот процесс в терминологии программы a2ps называется делегированием.

Приведем примеры делегирования.

Предположим, что вам нужно напечатать уже готовый файл в формате PostScript. Вы можете сделать это при помощи команды `a2ps -4 -P display T.ps`.

В данном случае программе a2ps указано вывести файл с именем `T.ps` на экран терминала в четыре колонки.

Поскольку исходный файл уже имеет формат PostScript, то программа a2ps производит делегирование:

```
future76:shevel /usr/home/shevel> a2ps -4 -P display T.ps
[T.ps (ps, delegated to PsNup): 1 page on 1 sheet]
[Total: 4 pages on 1 sheet] sent to the printer Display'
```

В данном случае делегирование произведено к известному программному фильтру PsNup (который выполнил преобразование файла в формате PostScript таким образом, чтобы разместить текст в четыре колонки).

Процесс делегирования описывается в конфигурационном файле. Чтобы узнать, какие конкретно делегирования настроены в вашей системе, используйте команду `a2ps --list=delegations`.

Вывод этой команды может выглядеть, например, так:

```
future76:shevel /usr/home/shevel> a2ps --list=delegations
Applications configured for delegation
Delegation Groff', from roff to ps
          eval Grog -Tps $f | #{psselect} | #{psnup}
Delegation Gzip-a2ps', from compressed to ps
          gzip -cd $f | #{a2ps} --stdin=$N
Delegation Netscape', from html to ps
          netscape -noraise -remote 'openurl($f)' -remote
'saveas(#f0,postscript)' &&     #{psselect} #f0 | #{psnup}
Delegation PsNup', from ps to ps
          fixps $f | #{psselect} | #{psnup}
Delegation Dvips', from dvi to ps
          #{dvips} $f -o #f0 && #{psnup} #f0
Delegation Pdf2ps', from pdf to ps
          pdf2ps $f #f0 && #{psselect} #f0 | #{psnup}
Delegation Texi2dvi', from texinfo to ps
          #{texi2dvi} $f && mv $N.dvi #f0 && #{dvips}
          -f #f0 | #{psnup}
```

Из приведенного текста видно, что a2ps может распечатать даже файл, уплотненный архиватором gzip, выполнив соответствующее делегирование. Вот пример:

```
future76:shevel /usr/home/shevel> a2ps T.ps.gz -P display
[T.ps.gz (compressed, delegated to Gzip-a2ps):
3 pages on 2 sheets]
[Total: 4 pages on 2 sheets] sent to the printer Display'
```

Нетрудно видеть, что используя механизм делегирования, из среды a2ps можно вызвать любую программу.

Стили печати

Для повышения выразительности печати подсистема a2ps дает возможность определять и использовать различные стили. Для большей выразительности следует использовать особенности печатаемого текста, например, стили печати текста программ на языке C и программ на языке FORTRAN-90 должны быть разными. Программа a2ps имеет набор предопределенных стилей печати, к описанию которых мы приступим в данном разделе.

Параметры выразительной печати

Таблица 4.12. Параметры выразительной печати a2ps

Параметр	Описание
--highlight-level=level	Аргумент level определяет степень форматирования текста (при помощи шрифтов, раскраски, и др.) Возможные значения аргумента level: • none – обычная печать, без форматирования; • normal – печать с выполнением обычного форматирования; • heavy – более детальное форматирование, нежели обычно. Дополнительную информацию о выполняемом в этом режиме форматировании можно получить при помощи команды a2ps: `--list=style-sheets`
-g	Сокращенный вариант параметра --highlight-level=heavy
-E language --pretty-print[=language]	Если аргумент language не указан, то производится автоматический выбора стиля печати. В противном случае используется стиль печати, заданный аргументом language. Заметим, что если значением language является plain, то форматирование при печати будет отключено. Для вывода списка возможных значений аргумента language используйте команду a2ps: `--list=style-sheets`. Некоторые из доступных стилей мы рассмотрим ниже. Если значение language имеет вид KEY.ssh, то программа не производит поиск стиля в библиотеке, а берет стиль печати из файла с данным именем (ssh – сокращение от stylesheet)
--strip-level=num	Аргумент num позволяет управлять печатью комментариев. Эта возможность удобна при печати программ на языке Java, а также при использовании спецификаций, которые созданы графическими редакторами. Аргумент num может принимать следующие значения: • 0 – нормальная печать; • 1 – обычные комментарии не печатаются; • 2 – серьезные комментарии не печатаются; • 3 – никакие комментарии не печатаются

Готовые стили печати

Стили печати описываются в специальных конфигурационных файлах программы a2ps. Имена файлов имеют вид `стиль.ssh`, например, `ada.ssh`. Конфигурационные файлы с описанием стилей находятся в подкаталоге `sheets` основного каталога `a2ps`.

В табл. 4.13 перечислены некоторые из стилей, входящих в комплект поставки `a2ps`.

Таблица 4.13. Стили печати a2ps

Обозначение стиля печати	Его назначение
68000	Предназначен для печати программ на ассемблере 68K. Предполагается, что такой стиль подходит и для печати программ на других видах ассемблера
ada	Стиль печати программ на языке Ada
sh	Стиль печати текстов сценариев на языке оболочек sh и bash
c gnuc	Стили печати программ на языке C
csh tcsh	Стили печати текстов сценариев на языке оболочек csh и tcsh
cpp objc	Стили печати программ на языках C++ и Objective C
caml	Стиль печати программ на языке ML
claire	Стиль печати программ на языке Claire
clisp	Стиль печати программ на языке Common Lisp
coqv	Стиль печати программ на языке Coq Vernacular
dc_shell	Стиль печати программ на языке описания электронных схем DesignCompiler
eiffel	Стиль печати программ на языке Eiffel
elisp	Стиль печати программ на языке Emacs Lisp
eps	Стиль печати программ на языке Encapsulated PostScript. Неверные операторы выделяются в тексте другим начертанием
tclx tk vtcl	Стили печати программ на языках Extended Tcl, Tk, Visual Tcl
fortran for-fixed for-free for-free	Стили печати программ на языке Fortran
for77-fixed for77 -free for77kwds	Стили печати программ на языке Fortran 77
for90-fixed for90-free for90kwds	Стили печати программ на языке Fortran 90
java	Стили печати программ на языке Java
modula2 modula3	Стили печати программ на различных вариантах языка Modula

Обозначение стиля печати	Его назначение
oberon	Стиль печати программ на языке Oberon
pascal	Стиль печати программ на языке Pascal
perl	Стиль печати программ на языке Perl
python	Стиль печати программ на языке Python
rexx	Стиль печати программ на языке REXX
sather	Стиль печати программ на языке Sather
scheme	Стиль печати программ на языке Scheme
zsh	Стиль печати текстов сценариев на языке оболочки zsh

В табл. 4.14 перечислены стили печати для вывода файлов специального назначения.

Таблица 4.14. Дополнительные стили печати a2ps

Обозначение	Его назначение стиля печати
card	Этот стиль помогает подготовить выразительную карту с кратким описанием параметров любой программы в Linux. Пример такой карты можно получить с помощью команды wget --help |a2ps -Ecard -1 --stdin=wget. Параметр -Ecard означает, что установлен стиль card. Параметр -1 указывает, что следует размещать одну страницу документа на одной физической странице. Наконец, параметр --stdin=wget означает, что входному потоку следует присвоить имя wget. Это имя будет использовано в колонтитулах напечатанных страниц
chlog	Стиль служит для печати файлов, которые содержат список изменений, внесенных в программу при переходе от одной версии к другой (такие файлы часто носят название ChangeLog).
gmakemake	Стили для печати make-файлов
html|vrml	Стиль для печати html-файлов и vrml-файлов
lace	Стиль для печати эквивалента make-файлов для языка Eiffel
mail	Стиль для печати электронных сообщений. Удобно использовать этот стиль совместно с параметром -g
initora	Стиль печати инициализационного файла Oracle init.ora
ps	Стиль печати файла в формате PostScript
pre	Стиль печати PreScript. Это специальный стиль, который позволяет использовать во входном потоке ряд операторов форматирования (см. раздел «Prescript»).
pretex	Стиль печати PreTeX. Это специальный стиль, который позволяет использовать во входном потоке ряд операторов форматирования (подмножество операторов LaTeX). Подробное описание вы найдете на страницах руководства info a2ps

продолжение

Таблица 4.14 *(продолжение)*

Обозначение	Его назначение стиля печати
texscript	Стиль печати TextScript. Это специальный стиль, который позволяет использовать во входном потоке операторы форматирования – как PreTeX, так и PreScript. Подробное описание вы найдете на страницах руководства info a2ps
a2psrc	Стиль печати инициализационных конфигурационных файлов программы a2ps: a2ps.cfg или .a2ps/a2psrc
ssh	Стиль печати конфигурационных файлов a2ps, описывающих стили печати, например, a2psrc.ssh или ada.ssh

PreScript

Поскольку в программе a2ps реализованы заглавные последовательности, специальные символы и другие управляющие команды, то было бы неплохо иметь какой-либо механизм для доступа к этим возможностям. Таким механизмом является разработанный совместно с a2ps язык описания входного потока данных, который получил имя PreScript. С помощью этого языка, используя синтаксис ssh (Style Sheets – таблицы стилей), можно управлять использованием шрифтов.

К основным достоинствам PreScript можно отнести простоту и доступность на любой аппаратной платформе.

Синтаксис

Каждая команда в языке PreScript начинается с обратного слэша (\).

Если команда использует аргумент, то он обязательно заключается в фигурные скобки. Не допускается никаких пробелов между командой и аргументом.

Внутри команд PreScript не должно использоваться никаких других команд PreScript, то есть суперпозиция команд запрещена. Например, следующая строка будет неверно интерпретироваться подсистемой a2ps:

```
\Keyword{Problems using \keyword{recursive \copyright}
calls}
```

Следует писать так:

```
\Keyword{Problems using} \keyword{recursive} \copyright
\Keyword{calls}
```

Для обозначения комментариев используется знак процента (**%**) в начале строки.

Команды PreScript

Таблица 4.15. Команды PreScript

Команда	Действие
Keyword{text} Keyword{text}	Выделение (сильное выделение) текста text. Может использоваться лишь для нескольких расположенных рядом слов (не для большого фрагмента текста)
comment{text} Comment{text}	Придать тексту text специальное начертание. Указать, что text может быть удален при использовании параметра --strip
label{text} Label{text}	Текст text должен рассматриваться как определение или как важный пункт в структуре документа
string{text}	Вывести текст text как строковую константу, обычно шрифтом Times
error{text}	Вывести текст text как сообщение об ошибке, то есть с помощью специального бросающегося в глаза шрифта
symbol{text}	Текст text написан с использованием символьного шрифта PostScript. Эта возможность несовместима с LaTeX. При наличии возможности рекомендуется использовать для обозначения символов специальные ключевые слова (совместимые с LaTeX). Их полный список можно найти в файле ./a2ps/sheets/symbols.ssh
header{text} footer{text}	Использовать текст text как заголовок или подстрочное примечание на текущей странице. Если указано несколько таких команд, то в последующих аргумент text может быть опущен. В этом случае используется текст из последней команды, в которой он был задан
encoding{key}	Изменить текущую кодировку входного потока. После этой команды текст будет печататься с использованием кодировки key

Примеры использования PreScript

Предположим, что мы хотим напечатать список пользователей, выделив часть выводимой информации:

```
 cat /etc/passwd |  \
awk -F: \
'{print "\\Keyword{" $5 "} (" $1 ")
\\rightarrow\\keyword{" $7 "}"}'\
| a2ps -Epre -P display -1
```

Обратите внимание на два обратных слэша, которые необходимо использовать, чтобы учесть особенности интерпретации команды оболочкой bash. В то же время, если запрос для программы awk будет находиться в файле (тогда команда имела бы вид `awk -F: -f input.awk ...`), то вывод на экран не потребовался бы, и следовало бы написать только один обратный слэш.

Итак, разберем пример подробнее. Выделенным шрифтом печатается поле комментария, затем, в скобках, имя пользователя в системе (login name), далее стрелка вправо, затем выделенным шрифтом печатается имя оболочки, установленной для данного пользователя по умолчанию.

Программе a2ps передается несколько параметров. Параметр `-Epre` определяет стиль печати PreScript. Значением параметра `-P` является `display`, то есть вывод программы будет направлен программе просмотра Postscript-файлов `gv`. Наконец, параметр `-1` означает, что на одной физической странице будет размещаться одна страница документа.

Инициализационные файлы a2ps

a2ps обрабатывает несколько файлов до того, как начинает интерпретировать параметры в командной строке. Файлы прочитываются в следующем порядке:

1. Системный инициализационный файл a2ps, который обычно располагается в каталоге `/usr/local/etc` или в каталоге `/etc`. Расположение конфигурационного файла может быть изменено при помощи переменной окружения `A2PS\_CONFIG`.
2. Пользовательский инициализационный файл `\$HOME/.a2ps/a2psrc`.
3. Локальный инициализационный файл текущего каталога `./.a2psrc`.

В перечисленных файлах пустые строки и строки, которые начинаются знаком # (решетка) игнорируются. Все другие строки имеют следующий формат:

```
TOPIC: arguments
```

где TOPIC есть параметр, значение которого устанавливается этой строкой, а arguments определяет само значение. Аргумент arguments может занимать несколько строк. В этом случае каждая строка, за которой следует продолжение, должна заканчиваться обратным слэшем.

Конфигурационные возможности a2ps придают дополнительную гибкость этой программе. Для получения дополнительной информации по этой теме используйте команду info a2ps.

Кодировка входного потока

Текстовые файлы могут иметь весьма различную кодировку, которая отражает языковые особенности использования тех или иных букв. Как следствие, почти каждый национальный язык имеет особенности в начертании букв. Для вывода на печать этих особенностей a2ps имеет специальный механизм — encoding — который позволяет не только указать нужную кодировку, но и добавить новую, если имеется такая необходимость. В последних версиях a2ps поддерживается несколько десятков кодировок, включая koi8-R (используемую в Linux в качестве кодировки кириллицы).

О процессе создания новой кодировки вы можете прочитать в руководстве, выводимом командой info a2ps.

Параметры a2ps

Чтобы прочесть и проанализировать параметры в командной строке, a2ps использует систему GNU getopt (см. man getopt, сравните со встроенной в оболочку bash командой getopts). Использование GNU getopt предопределяет следующие особенности представления параметров в командной строке:

- параметры любого вида должны быть разделены пробелами;
- порядок файлов и параметров не имеет значения, например, a2ps -1 -d document означает то же, что a2ps -d document -1;
- порядок параметров, имеющих сходные по смыслу аргументы, важен;

- параметры в краткой форме записи могут быть сгруппированы, например, можно записать `a2ps -4mg main.c` вместо `a2ps -4 -g -m main.c`;
- если нет двусмысленности, то длинные имена параметров могут быть сокращены, например, `--pro` будет интерпретировано как `--prologue`.
- двумя дефисами (`--`) можно закончить перечисление параметров в командной строке; далее могут быть только имена файлов (что удобно, если имена файлов начинаются с дефиса).

Если аргумент представлен в квадратных скобках, то он не является обязательным. Необязательный аргумент должен записываться слитно с параметром, если последний представлен в краткой форме.

Параметры для выдачи справки программой a2ps

Данная группа параметров отвечает за вывод справочной информации программой a2ps. При этом a2ps лишь выдает требуемую информацию на экран и завершает свою работу.

Таблица 4.16. Информационные параметры a2ps

Параметр	Назначение
-V --version	Вывести версию программы
-h --help	Вывести на экран краткое перечисление параметров a2ps
--copyright	Вывести на экран сведения о правах копирования для a2ps
--guess	Действовать, как программа file, то есть в этом случае a2ps должна самостоятельно определить тип файла
--list=topic	Вывести информацию о программе a2ps по теме topic. Допустимые значения аргумента topic: defaults или options – вывести подробный отчет об установках по умолчанию; features – вывести список всех переменных, значения вы можете определить или изменить; delegations – вывести список установленных делегирований; encodings – вывести список распознаваемых кодировок; variables – вывести подробный список переменных; media – вывести список известных программе типов носителей для печати; prologues – вывести список прологов для PostScript; printers – вывести список известных программе принтеров и устройств для вывода; style-sheets – вывести список известных программе стилей печати; user-options – вывести список пользовательских параметров

Глобальные параметры a2ps

Таблица 4.17. Глобальные параметры a2ps

Параметр	Назначение
-q --quiet --silent	Программа должна выполняться без выдачи сообщений («молча»)
-vlevel --verbose=level	Аргумент level определяет уровень диагностики: 0 – запретить диагностику; 1 – a2ps будет сообщать только общее количество напечатанных страниц; 2 – будет выведено число отпечатанных страниц для каждого файла (значение по умолчанию)
level=configuration	Действие определяется аргументом configuration: options – прочесть конфигурационные файлы и параметры; files – вывести имена всех файлов, которые использовались в данной команде (как для чтения, так и для записи); fonts – вывести список использованных шрифтов; meta-sequences – перечислить расширения макрокоманд; parsers – вывести протокол подстановки макроопределений и протокол анализа командной строки; pathwalk – вывести протокол поиска различных файлов (шрифтов, конфигурационных файлов и т. д.); ppd – обработать таблицу стилей PostScript Printer Description (этот файл содержит информацию о принтере); sheets – вывести список использованных таблиц стилей; all – вывести всю отладочную информацию о выполнении программы При запуске программы a2ps производится проверка значения переменной окружения A2PS_VERBOSITY. Если переменная установлена, то ее значение определяет уровень детальности и характер сообщений программы a2ps, то есть значения, установленные в командной строке, не будут оказывать влияния. Допустимыми значениями для A2PS_VERBOSITY являются те же, что и для --verbose в командной строке. То, что значения переменной A2PS_VERBOSITY являются более приоритетными, позволяет проверять и отлаживать содержание конфигурационных файлов, которые программа прочитывает до анализа параметров в командной строке
-=shortcut --user-option=shortcut	Использовать сокращения, которые определил пользователь. Такие сокращения могут перемежаться в командной строке с обычными параметрами программы a2ps. Имеются три встроенных пользовательских сокращения: lp – эмулировать простое устройство печати, то есть удалить все элементы, повышающие выразительность печатаемого текста, например, выделение полужирным шрифтом или курсивом; mail или longmail – установить предпочтительные значения для печати электронной почты или сообщений из групп новостей; manual – приготовить задание для печати файла на принтере с ручной подачей бумаги
--debug	Включить отладочный режим (этот параметр может помочь понять, почему принтер не печатает ваш файл)

продолжение ⇙

Таблица 4.17 *(продолжение)*

Параметр	Назначение
-D key[=value] --define:key[=value]	Если аргумент value не задан, то переменная key будет деинициализирована. Заметим, что -Dtest= присвоит переменной test пустое значение, а –Dtest удалит переменную test (будет выполнена операция unset). Напомним, что речь идет о механизме внутренних переменных системы печати a2ps, а не о переменных окружения

Общий стиль выводимых страниц

Рассматриваемые в этом разделе параметры программы a2ps определяют общий план размещения текста на печатаемых страницах.

Таблица 4.18. Параметры описания плана вывода a2ps

Параметр	Описание
-M medium --medium=medium	Использовать формат носителя medium. Полный список форматов можно получить при помощи команды a2ps --list=media. Типичными значениями medium являются A3, A4, Letter и т. п. Имеется специальное значение libpaper, которое приводит к тому, что a2ps будет использовать библиотеку libpaper для определения формата. Однако такое значение может быть использовано только в том случае, если поддержка libpaper была включена при компиляции программы a2ps
-r --landscape	Печатать с альбомной ориентацией (горизонтально)
-R --portrait	Печатать с книжной ориентацией (вертикально)
--columns=num	Определить число колонок текста на физической странице
--rows=num	Определить число строк для размещения текста на физической странице
--major=direction	Определить, в каком порядке будут располагаться страницы текста на физической странице: по рядам – значением direction является rows, или по колонкам – значением direction является columns
-1	Вертикальное расположение 1x1 (книжная ориентация), 80 символов в строке. Параметр аналогичен набору параметров --columns=1 --rows=1 --portrait –chars-per-line=80 --major=rows
-2	2 x 1 (альбомная ориентация), 80 символов в строке, расположение по рядам. Параметр аналогичен набору параметров --columns=1 --rows=1 --portrait -chars-per-line=80 --major=rows
-3	3 x 1 (альбомная ориентация), 80 символов в строке, расположение по рядам
-4	2 x 2 (книжная ориентация), 80 символов в строке, расположение по рядам

Параметр	Описание
-5	5 x 1 (альбомная ориентация), 80 символов в строке, расположение по рядам
-6	3 x 2 (альбомная ориентация), 80 символов в строке, расположение по рядам
-7	7 x 1 (альбомная ориентация), 80 символов в строке, расположение по рядам
-8	4 x 2 (альбомная ориентация), 80 символов в строке, расположение по рядам
-9	3 x 3 (книжная ориентация), 80 символов в строке, расположение по рядам
-j --borders=boolean	Напечатать рамку вокруг каждой страницы, если boolean – 1. Если boolean равно 0, то не печатать рамку
-A mode --file-align=mode	Выровнять отдельные файлы при печати в соответствии с аргументом mode. Аргумент mode может принимать следующие значения: virtual – печать каждого следующего файла начинается с новой логической страницы; rank – печать каждого следующего файла начинается в новом ряду или в новой колонке, в зависимости от значения –major; page – печать каждого следующего файла начинается на новой физической странице; sheet – печать каждого следующего файла начинается на новом листе (в случае односторонней печати этот параметр делает то же самое, что и предыдущий); num – печать каждого следующего файла начинается на странице с номером num+1. Это, в частности, означает, что если потребуется пропустить несколько страниц, то они будут присутствовать в документе, но останутся пустыми
-margin=[num]	Установить размер левого поля на странице в num пунктов (12, если аргумент num не задан). Имеется в виду отступ на физическом листе слева для удобства подшивки в скоросшиватель

Определение плана страницы документа

Эта группа параметров программы a2ps определяет план страницы документа.

Таблица 4.19. Параметры описания плана страницы

Параметр	Описание
-line-numbers[=number]	Включить нумерацию строк. Значение number задает интервал нумерации. По умолчанию интервал нумерации равен 1, то есть будут пронумерованы все строки подряд (номер печатается слева от строки). Например, если использован параметр --line-numbers=5, то номера будут проставлены у каждой пятой строки, то есть у строк 5, 10, 15 и т. д.

продолжение

Таблица 4.19 *(продолжение)*

Параметр	Описание
-C	Этот параметр означает то же самое, что и --line-numbers=5
-f size[unit] --font-size=size[unit]	Установить размер шрифта равным size. Аргумент size может являться десятичной дробью. Аргумент unit может принимать одно из следующих значений: cm – размер задан в сантиметрах; points – размер задан в пунктах; in – размер задан в дюймах
-l num --chars-per-line=num	Установить размер шрифта таким, чтобы в одной строке помещалось бы num символов
-L num --lines-per-page=num	Установить размер шрифта таким, чтобы на одной странице помещалось бы num строк. Это очень полезно, если ваш документ уже обработан какой-то программой формирования текста и имеет фиксированное число строк на странице. Аргумент num может принимать значения от 40 до 160
-m --catman	Это параметр включает режим печати страниц описаний (man-страниц). Например: `$ man genscript | a2ps --catman`
-T num --tabsize=num	Установить размер табуляции num. Это значение будет, естественно, игнорироваться, если одновременно использован параметр --interpret=no
--non-printable-format=format	Аргумент format позволяет задать, как будут отмечаться при печати неотображаемые символы: caret – использовать представление, обычное для Unix-систем: ^A, ^B, и т. п.; space – использовать вместо неотображаемых символов пробел; question-mark – использовать вместо неотображаемых символов знак вопроса; octal – использовать восьмеричное представление: \001, \123 и т. п. hexa – использовать шестнадцатеричное представление: \x02, \xfd и т. п. emacs – использовать обозначения типа C-h, M-C-a и т. п.

Параметры определения колонтитулов

С помощью этих параметров вы можете описать, что вы хотели бы видеть на ваших страницах.

Таблица 4.20. Параметры определения колонтитулов

Параметр	Описание
-B --no-header	Не выводить колонтитулы
-btext --header[=text]	Определить заголовок страницы
--center-title[=text] --left-title[=text] --right-title[=text]	Определить верхний колонтитул, печатаемый в центре страницы документа, у левого края страницы документа, у правого края страницы документа, соответственно

Параметр	Описание
-u*text* --underlay[=text]	Текст, заданный аргументом text, будет выведен в качестве фона страницы (как «водяной знак»), то есть светло-серый текст text поместить по диагонали каждой страницы. Отметим, что если программа a2ps выполняет делегирование, то этот параметр может не оказать никакого действия
--footer[=text] --left -footer[=text] --right -footer[=text]	Определить нижний колонтитул, печатаемый в центре страницы документа, у левого края страницы документа, у правого края страницы документа, соответственно

Параметры задания устройства вывода

Эти параметры позволяют вам описать, куда вы хотите направить ваш вывод из программы a2ps. Вывод может быть направлен лишь на одно из нескольких возможных устройств (или в файл).

Таблица 4.21. Параметры, определяющие файл или устройство для вывода

Параметр	Описание
-o file --output[=file]	Сгенерированный PostScript-документ сохраняется в файл с именем file, а не направляется на принтер. Если имя файла указано как --, то PostScript-документ будет направлен на устройство стандартного вывода
--version-control=type	Чтобы избежать случайной потери уже существующего файла, программа a2ps предлагает схемы сохранения файлов в том случае, если имя, использованное в параметре --output, совпадает с именем уже существующего файла. Тип резервной копии файла может быть указан с помощью переменной окружения VERSION_CONTROL. Если переменная не установлена и параметр --version-control не использован, то предполагается простое резервное копирование (выполняемое также, если задан параметр --version-control=existing). Аргумент type может принимать следующие значения: none – резервное копирование не выполняется, то есть если файл с указанным именем существует, то он просто удаляется; existing – создаются нумерованные резервные копии файла simple – создается одна резервная копия файла
--suffix=suffix	Этот параметр позволяет установить суффикс (расширение), используемый в имени файла резервной копии. Суффикс также может быть определен при помощи переменной окружения SIMPLE_BACKUP_SUFFIX, однако параметр командной строки имеет более высокий приоритет. По умолчанию в качестве суффикса используется символ -

продолжение ☞

Таблица 4.21 *(продолжение)*

Параметр	Описание
-P name --printer=name	Произвести печать на принтере с именем name. Обратите внимание на специальные значения параметра --printer, описанные в разделе «Простые примеры использования a2ps». Дополнительную информацию позволит получить команда a2ps --list=defaults
-d	Произвести печать на принтер по умолчанию

Параметры для управления PostScript-выводом

Следующие параметры позволяют управлять процессом генерации PostScript-вывода программой a2ps.

Таблица 4.22. Параметры для управления PostScript-выводом

Параметр	Описание
--ppd[=key]	Если аргумент key не задан, то будет произведен автоматический выбор файла описания принтера (PPD); в противном случае в качестве PPD будет использоваться key
-n num --copies=num	Вывести num копий каждой страницы
-s simplex-mode --sides=simplex-mode -s duplex-mode --sides=duplex-mode	Этот параметр позволяет задать одностороннюю (значение аргумента – 1 или simplex) либо двустороннюю (2 или duplex) печать. При двусторонней печати предполагается, что она будет осуществляться средствами принтера. Если вы хотите осуществить двустороннюю печать средствами a2ps (то есть изменить расположение полей на четных и нечетных страницах), то используйте в качестве значения аргумента tumble
-S KEY[:value] --setpagedevice= KEY[:value]	Передать определение страничного устройства в генерируемый PostScript-документ. Если аргумент value не указан, то ключевое слово KEY удаляется из документа
--statusdict=KEY[:value]	Передать определение «statusdict» в генерируемый PostScript-
--statusdict=KEY[:value]	документ. Если аргумент value не задан, то ключевое слово KEY удаляется из определения в документе. Например, если принтер позволяет выбирать лоток с бумагой, то команда $ a2ps --statusdict=setpapertray:1 quicksort.c печать файла quicksort.c с использованием бумаги из подающего лотка номер 1
-k --page-prefeed	Разрешить предварительную подачу листа бумаги во время печати. Этот режим позволяет принтеру подавать бумагу одновременно с интерпретацией кода PostScript (вместо ожидания конца интерпретации кода). Такой режим может привести к заметному ускорению печати документа, состоящего из большого числа страниц (около сотни и более)
-K --no-page-prefeed	Запретить предварительную (одновременно с интерпретацией кода PostScript) подачу бумаги

Примеры использования a2ps

Классическим примером может быть использование программы a2ps для распечатки страниц описаний:

```
$ man awk | a2ps --stdin=awk -1 -Xkoi8
```

описание программы awk будет напечатано на принтере (около 30 страниц). Каждая страница будет иметь сверху заголовок (колонтитул) awk, на одной физической странице будет располагаться одна страница документа. Дата и время будут напечатаны на русском языке, конечно, если на вашей машине (как на моей) выполнена корректная локализация (см. раздел «На каком языке говорит Linux?»).

Если вы захотите разместить на странице текст, напоминающий водяные знаки, то можно изменить команду так:

```
$ man awk | a2ps --stdin=awk -1 -Xkoi8 \
> --underlay="Официальное Описание"
```

После выполнения этой команды на каждой напечатанной (физической) странице появится светло-серая надпись по диагонали страницы, в данном случае это будет «Официальное описание».

Если вам потребуется иметь на бумаге краткое описание параметров известных компиляторов C и FORTRAN-90, это можно сделать так:

```
$ gcc --help | a2ps --stdin=gcc -1 -Ecard
$ vf90 2>&1 | a2ps --stdin=vf90 -1 -Xkoi8 -Ecard
```

Во второй команде пришлось выполнить перенаправление ввода-вывода с устройства 2 на устройство 1 (перенаправление стандартного потока ошибок на стандартный выход), поскольку компилятор vf90 выводит сообщения в стандартный поток ошибок.

Если таким образом вывести несколько полезных кратких описаний, то можно быстро, за несколько минут, приготовить себе подшивку на манер справочника. Аналогичным образом можно печатать тексты программ на любом языке. Поскольку по умолчанию a2ps пытается «догадаться», какой стиль печати следует использовать, то вы получите весьма выразительные страницы, которые также можно будет ис-

пользовать как справочный материал или, например, как часть отчета.

Для предварительного просмотра можно задать несколько вариантов сравнения, добавив параметр `-P display`. При этом результат будет выведен на дисплей с помощью программы просмотра PostScrip-файлов gv, а не на принтер. Вы можете использовать параметр `-o`, чтобы записать полученный результат в файл, например:

```
$ man od | a2ps --stdin='Utility od' -1 -Xkoi8 -o od.ps
```

PostScript-файл с описанием утилиты `od` будет сохранен под именем `od.ps`.

Если вам надо напечатать список или таблицу, то для удобства чтения полезно разместить на листах какие-то горизонтальные линии. Это легко сделать так:

```
 $ man bash | a2ps --stdin='bash' -1 -Xkoi8 \
> --prologue=matrix -P display
```

Результат будет выведен на экран дисплея.

Предположим, что у вас имеется довольно объемный документ в формате PostScript, например, 150 страниц. Вам хотелось бы напечатать этот документ компактнее, то есть по несколько страниц документа на одной физической странице. Это легко сделать таким образом:

```
$ a2ps Book.ps -9 -Xkoi8
```

На одной физической странице будет помещено 9 страниц документа. Если вы выведете результат на принтере с разрешением 1200 dpi, то все страницы будут выглядеть весьма мелко, но все будет ясно видно, например, с соответствующими очками.

Имейте в виду, что ряд интересных возможностей программы a2ps можеть не работать при делегировании обработки файлов другим программам. Например, если вы попробуете использовать параметр `--prologue=matrix` при печати PostScript-файлов, то не обнаружите никаких полос на напечатанных страницах. Иными словами, в данном случае определяющими являются свойства той программы, которой производится делегирование (передача) обработки файла.

Программа head

Синтаксис:

```
head [option]... [file]...
head -number [option]... [file]...
```

Программа выводит на устройство стандартного вывода головную часть каждого файла `file` (по умолчанию 10 строк). Если указано более одного файла, программа печатает при переходе к каждому следующему файлу однострочный заголовок следующего вида:

```
==> FILE NAME <==
```

Программа head воспринимает несколько параметров, которые могут быть представлены в двух форматах: старом или новом. В новом формате числовые значения аргументов представляются в виде `-q -n 1`, в то же время старый формат то же значение аргумента представляет так: `-1q`.

- `-countoptions` — этот параметр распознается программой только в том случае, если он определен раньше всех остальных параметров. `count` — это десятичное целое, за которым может следовать буква, указывающая размер: `b` (блок — 512 байт), `k` (1024 байта) или `m` (1048576 байта), как в параметре `-c`, или буква `l` (для указания количества строк) или другие параметры: `c`, `q`, `v`.

  Например, следующая команда выведет один блок (512 байтов) из файла с именем FileName.

  ```
  $ head -1b FileName
  ```

  Команда

  ```
  $ head -1bl FileName
  ```

  выведет одну строку из файла с именем `FileName`. Ту же операцию можно выполнить при помощи команды

  ```
  $ head -1l FileName
  ```

- `-c bytes` — напечатать первые `bytes` байтов вместо начальных строк. За десятичным целым может следовать буква `b` (умножить число на 512), `k` (умножить число на 1024), `m` (умножить число на 1048576).
- `-n n` или `--lines=n` — вывести первые `n` строк.

- `-q` или `--quiet` или `--silent` — не печатать заголовок с именем файла.
- `-v` или `--verbose` — всегда печатать заголовок, содержащий имя файла.

Программа tail

Синтаксис:

```
tail [option]... [file]...
tail -number [option]... [file]...
tail +number [option]... [file]...
```

Программа tail выводит на устройство стандартного вывода последние (по умолчанию 10) строки каждого файла, указанного в командной строке. Если указано более одного файла, программа печатает при переходе к каждому следующему файлу однострочный заголовок следующего вида:

```
==> FILE NAME <==
```

GNU-версия tail может выводить любое количество данных. Эта версия программы в отличие от других не имеет параметра `-r` (вывод строк файла в обратном порядке). Для вывода строк файла в обратном порядке можно использовать программу tac (см. раздел «Программа tac»).

Программа распознает несколько параметров:

- `-count` или `+count` — этот параметр распознается только в том случае, если он указан первым. Если за десятичным целым `count` не следует никаких букв, то оно задает количество строк, которые будут выданы на печать. (Если использован знак - (минус) или нет никакого знака. Если использован знак + (плюс), то на устройство стандартного вывода выводится информация из файла, начиная со строки с номером `count` от начала файла.) За номером может следовать буква `l` (означает, что счет идет в строках) или буквы `b`, `k`, `m` с теми же значениями, как в параметре `-c`.
- `-c bytes` или `--bytes=bytes` — вывести последние `bytes` байтов вместо последних строк. Добавление буквы `b` после десятичного целого `bytes` означает умножение на 512, `k` — на 1024, `m` — на 1048576.

- `-f` или `--follow` — войти в бесконечный цикл, который пытается читать больше данных в конце файла. Предположительно файл постоянно растет, поэтому вы будете видеть поступающую в файл информацию. Параметр игнорируется, если чтение производится с устройства стандартного ввода (в том числе из программного канала). Если определено несколько файлов, то tail выводит заголовки с именами файлов.
- `-n n` или `--lines=n` — вывести последние `n` строк.
- `-q` или `-quiet` или `--silent` — не выводить заголовки с именами файлов (по умолчанию, если указано несколько файлов, заголовки выводятся).
- `-v` или `--verbose` — всегда выводить заголовки с именами файлов.

Программа split

Синтаксис:

```
split [option] [input [prefix]]
```

По умолчанию программа split помещает каждые очередные 1000 (тысячу) строк ввода в отдельный файл. Имена выводных файлов формируются на основе значения `prefix` (по умолчанию `x`), за которым следует буквенная комбинация вида `aa`, `ab` и так далее. Таким образом, конкатенация всех выводных файлов составит исходный вводной файл. Если количество выводных файлов превышает число 676, то split добавляет количество букв в именах вновь создаваемых файлов. Этот факт индицируется дополнительной буквой `z`, которая следует за префиксом.

Программа воспринимает следующие параметры.

- `-lines` или `-l lines`, или `--lines=lines` — разбить вводной файл на части по `lines` строк и поместить результат в отдельные файлы.
- `-b bytes` или `--bytes=bytes` — поместить в каждый создаваемый файл очередные `bytes` байтов. Добавление буквы `b` после `bytes` умножает число `bytes` на 512, `k` — на 1024, `m` — на 1048576.

- `-C bytes` или `--line-bytes=bytes` — поместить в каждый создаваемый файл столько полных строк ввода, сколько возможно без превышения `bytes` байтов в каждом создаваемом файле. Для строк, размер которых превышает `bytes` байтов, записывать очередные `bytes` байтов до исчерпания строки, затем действовать как обычно.
- `--verbose` — выдавать диагностические сообщения перед открытием очередного выводного файла.

Программа csplit

Программа csplit предназначена для разделения файла на контекстно обусловленные части.

Синтаксис:

```
csplit [option]... input pattern...
```

`input` представляет собой имя файла или устройство стандартного ввода. Содержание выводных файлов определяется значениями аргументов `pattern` в соответствии с алгоритмом, который описывается ниже.

Если регулярное выражение `pattern` не соответствует никакой строке во вводном файле, то возникает состояние ошибки. После того как все выражения `pattern` найдены во вводном файле, остаток вводного файла просто копируется в последний выводной файл.

По умолчанию программа csplit печатает число байтов, записанных в каждый выводной файл.

Программа допускает следующие виды регулярных выражений `pattern`:

- `n` — создать выводной файл, содержащий строки вводного файла до строки `n` (не включая саму строку `n`). Оставшаяся часть файла будет записана во второй выводной файл. Например, последовательность команд

  ```
  $ csplit crawler 10
  $ wc *
  ```

  даст в результате следующее:

  ```
   $ csplit crawler 10
  790
  21371
  ```

```
$ wc *
      337     1441    22161 crawler
        9       81      790    xx00
      328     1360    21371    xx01
      674     2882    44322   total
```

Можно использовать два и более чисел:

```
csplit crawler 10 14 19 39
wc *
```

Получим следующее:

```
 $ csplit crawler 10 14 19 39
790
948
374
667
19382
$ wc *
      337     144122161 crawler
        9       81  790    xx00
        4       69  948    xx01
        5       31  374    xx02
       20       63  667    xx03
      299     119719382    xx04
      674     288244322   total
```

При этом во второй файл (`xx01`) попадут строки с номерами от 10 до 13, в третий файл (`xx02`) — строки с номерами от 14 до 18, в четвертый файл (`xx03`) — строки с номерами от 19 до 38, а остаток запишется в файл `xx04`.

- `/pattern/[offset]` — создать файл, содержащий часть исходного файла от текущей строки до (не включая) строки, которая удовлетворяет регулярному выражению `pattern`. Если использован аргумент `offset` (десятичное целое со знаком плюс или минус), то выводится часть файла до строки, удовлетворяющей регулярному выражению `pattern` + (плюс) или - (минус) `offset` и одна строка, после которой начинается очередная часть ввода.

Например, пусть имеется текстовый файл `t`:

```
$ cat t
111111
222222
333333
444444
```

```
555555
666666
```

Тогда команда

```
$ csplit t 2 /333/+1
7
14
22
$ wc *
       7       6      43 t
       1       1       7 xx00
       2       2      14 xx01
       4       3      22 xx02
      14      12      86 total
```

создаст три файла, и их содержимое будет следующим:

```
$ cat xx00
111111
$ cat xx01
222222
333333
$ cat xx02
444444
555555
666666
```

Если слэши в выражениях заменить на знаки процента, то никаких выводных файлов создаваться не будет. Такое выражение позволяет пропустить определенную часть исходного файла. Например,

```
$ csplit t 2 %333%+1
```

В этом случае будет создано всего два файла.

- `\repeat-count\` — повторить предыдущее регулярное выражение `repeat-count` раз. Число `repeat-count` может быть десятичным положительным целым или звездочкой. Звездочка означает, что выражение повторяется до тех пор, пока не закончится ввод.

Имена выводных файлов состоят из префикса (по умолчанию `xx`), за которым следует суффикс. По умолчанию суффикс состоит из двух цифр (от 00 до 99).

По умолчанию в случае ненормального завершения программы csplit она удаляет все созданные файлы.

Программа воспринимает следующие параметры.

- `-f prefix` или `--prefix=prefix` — использовать строку `prefix` в качестве префикса в именах создаваемых выводных файлов.
- `-b suffix` или `--suffix=suffix` — использовать строку `suffix` как суффикс в именах выводных файлов. Когда этот параметр указан, строка `suffix` должна включать ровно один символ для определения типа преобразования (как в функции `printf(3)`). В строке `suffix` могут быть также использованы флаги определения формата, количество знаков для числа, точность преобразования. Тип преобразования должен задавать перевод двоичного числа в читаемую форму, иными словами могут быть заданы типы: `d`, `i`, `u`, `o`, `x` и `X`. Если использован данный параметр, то параметр `--digits` игнорируется.
- `-n digits` или `--digits=digits` — использовать суффикс размером `digits` цифр в именах выводных файлов. По умолчанию используется две цифры.
- `-k` или `--keep-files` — не удалять созданные файлы в том случае, если работа программы завершилась ненормально (встретилась ошибка).
- `-z` или `--elide-empty-files` — запретить создание файлов нулевой длины.
- `-s` или `-q` или `--silent` или `--quiet` — не печатать размеры создаваемых файлов. По умолчанию выводятся размеры в байтах всех создаваемых файлов.

Программа sort

Синтаксис:

```
sort [option]... [file]...
```

Программа sort сравнивает, объединяет, сортирует вводной поток и выдает результат на устройство стандартного вывода. В качестве вводного потока может использоваться один или несколько файлов или устройство стандартного ввода. Программа имеет три режима работы: режим сортировки (умолчание), режим слияния, режим проверки —

отсортирован файл или нет. Переключение в режим слияния и проверки производится следующими параметрами:

- `-m` — объединить данные файлы, сортируя их как одну группу. Каждый вводной файл должен быть уже отсортирован индивидуально. Можно использовать вместо слияния режим сортировки, однако в режиме слияния (если его возможно использовать) программа работает быстрее.
- `-c` — проверить, отсортированы ли уже данные файлы. Есяи файлы отсортированы, то устанавливается код завершения 0. Если файлы не отсортированы, то выдается сообщение об ошибке и устанавливается код завершения 1.

Несколько замечаний о характере сортировки

Полем считается непустая последовательность символов, не содержащая пробелов или символов табуляции.

Если определены ключевые поля, по которым производится сортировка, то программа сравнивает каждую пару полей в соседних строках в порядке, определеном в командной строке, с учетом параметров упорядочивания, до того момента, когда будет встречено различие либо строки вводного потока будут исчерпаны.

Если определены любые глобальные параметры сортировки из списка `Mbdfinr`, но не определены ключевые поля, то программа будет сравнивать полные строки.

GNU-версия программы sort не имеет ограничений на длину вводной строки или на значения байтов, которые составляют вводную строку. Если возникла ошибка, программа завершается с кодом 2. Если установлена переменная окружения `$TMPDIR`, то sort будет использовать это значение как имя каталога, в котором будут располагаться временные файлы. По умолчанию используется каталог `/tmp`. Параметр `-T tmpdir`, в свою очередь, устанавливает новое значение имени каталога для временных файлов вне зависимости от значения переменной окружения `$TMPDIR`.

Параметры, определяющие способ сортировки

Нижеследующие параметры влияют на упорядочивание выводимых строк. Они могут быть определены глобально или как часть определенного ключевого поля. Если никаких ключевых полей не используется, то глобальные параметры воздействуют на сравнение полных строк. В противном случае глобальные параметры наследуются теми ключевыми полями, которые сами не имеют специальных параметров.

- `-b` — игнорировать ведущие пробелы при поиске ключей сортировки в каждой строке.
- `-d` — сортировать по правилам телефонного справочника: игнорировать все символы, кроме букв, пробелов и цифр.
- `-f` — игнорировать регистр символов. Например, `G` и `g` будут считаться одним и тем же символом.
- `-g` — сортировать по числовым значениям. Для получения числовых значений использовать функцию `strtod`. Это позволяет сравнивать числа, представленные в формате `nnnE+mm`. Это очень медленный режим работы, и его использование рекомендуется только в случае крайней необходимости. Имейте в виду, что числа, выходящие за пределы точности двойного слова (8 байтов), рассматриваются как нулевые значения. Никакой диагностики о переполнениях не предусмотрено.
- `-i` — игнорировать во время сортировки любые символы вне восьмеричных кодов ASCII, то есть символы с кодами вне диапазона 040–0176.
- `-M` — сравнение строк, в которых после любого количества пробелов следуют трехбуквенные обозначения месяцев. Основное правило сравнения таково:

  `JAN<FEB<MAR<...<DEC`

Регистр символов игнорируется. Неверная аббревиатура меньше верной.

- `-n` — сортировать по числовым значениям. Числу в строке может предшествовать сколько угодно пробелов. Числу может предшествовать знак минус. Число может содер-

жать десятичную точку. Число может отсутствовать (состоять из 0 цифр). Не распознаются знак плюс и символ экспоненты (чтобы провести сортировку с их распознаванием, используйте параметр `-g`).

- `-r` — сортировка в обратном порядке.

Прочие параметры

У программы sort имеются и другие параметры, не относящиеся к способу сортировки.

- `-o output-file` — произвести вывод не на устройство стандартного вывода, а в файл с именем `output-file`. Если имя файла `output-file` совпадает с именем исходного файла, то программа `sort` запишет результат сортировки во временный каталог, а потом скопирует результат в исходный файл.
- `-t separator` — использовать символ `separator` как разделитель полей во время поиска сортировочных ключей в каждой строке. По умолчанию поля разделяются пустой строкой между непробельными символами и пробелами. Например, если строка содержит

  ```
  miru mir
  ```

  то программа sort разделит ее на два поля: `miru` и `mir`.
- `-u` — для случаев по умолчанию или при использовании параметра `-m` выводит лишь первую из одинаковых строк. Если использован параметр `-c`, то проверяется, что нет одинаковых подряд идущих строк.
- `-k pos1[,pos2]` — рекомендованная POSIX-форма параметра, определяющего сортировочное поле в строке. Поле состоит из символов, заключенных между `pos1` и `pos2` или концом строки, если `pos2` опущено. Поля и символьные позиции нумеруются, начиная с 1.

Позиция внутри строки определяется в форме `F.C`, где `F` — номер поля, а `C` — номер символа в поле (считая слева направо), с которого начинается сортировочный ключ (если параметр задан как `+pos`). Если параметр задан как `-pos`, то отсчет производится с конца поля. Если `C` опущено, то подразумевается первый символ поля. Если определен пара-

метр `-b`, то часть `C` отсчитывается от первого непробельного символа данного поля (если параметр указан как `+pos`) или от первого непробельного символа следующего за предыдущим полем (если параметр указан как `-pos`). Ключ сортировки может иметь свои параметры сортировки: `Mbdfinr`. В этом случае глобальные параметры сортировки не применяются к данному полю. Параметр `-b` может быть использован отдельно и независимо для случаев `+pos` и `-pos`. Если параметр `-b` наследуется из глобальных параметров, то он используется во всех случаях. Ключи могут включать несколько полей. (См. примеры ниже.)

`-z` — обработать входной поток как набор строк, каждая из которых заканчивается символом `NULL`, а не `\n`. Этот параметр может оказаться полезным в специальных случаях, например, при использовании команд `perl -0` или `find -print0` или `xargs -0`.

Примеры использования sort

- Отсортировать численно в обратном порядке (в сторону уменьшения значения чисел):

  ```
  $ ps | sort -rn
  ```

  В этом примере вы получите список ваших процессов, упорядоченный по номеру процесса, максимальный номер будет вверху.

- Отсортировать в алфавитном порядке, опустив при этом первые 10 полей.

  ```
  $ ps ux | sort -k 11
  ```

  Вы получите алфавитно упорядоченный по 11-му полю (имя программы в RedHat 5.1) список.

- Отсортировать по алфавиту файл `/etc/passwd` по второй букве поля комментариев, а строки с одинаковыми вторыми символами отсортировать по первой букве:

  ```
  $ sort -t : -k 5.2,5.2 -k 5.1,5.1 /etc/passwd
  ```

  Заметим, что если написать не `5.2,5.2`, а просто `5.2`, то в качестве сортировочного ключа будет взято пятое поле, начиная с символа номер 2 (напомним, нумерация начинается с 1).

- Отсортировать по алфавиту файл `/etc/passwd` по пятому полю, игнорируя лидирующие пробелы, а поля с равными значениями сортировать по убыванию по числовому значению третьего поля.

  ```
  $ sort -t : -k 5b,5 -k 3,3rn /etc/passwd
  ```

  Заметим, что запись `-5b,5` означает произвести сортировку по пятому полю. Если написать просто `-5b`, то в качестве ключа сортировки будет использована часть строки, начиная с пятого поля и продолжающаяся до конца строки.

- Наконец, чтобы игнорировать как лидирующие пробелы в поле, так и замыкающие пробелы поля, можно использовать команду

  ```
  $ sort -t : -k 5b,5b -k 3,3rn /etc/passwd
  ```

  или использовать глобальный параметр `-b`:

  ```
  $ sort -t : -b -k 5,5 -k 3,3rn /etc/passwd
  ```

Программа uniq

Эта программа предназначена для удаления повторяющихся строк. Синтаксис:

```
uniq [option] [input [output]]
```

По умолчанию, программа `uniq` читает входной поток, который предположительно является отсортированным. Если встречаются одинаковые идущие подряд строки, то программа удаляет повторения, оставляя лишь по одной строке (первой) из любого количества повторяющихся строк. (Смотрите ниже описание параметров программы `uniq`). В данном разделе обсуждается GNU-версия программы.

Вывод производится в файл с именем `output`, если параметр не определен, то вывод производится на стандартное устройство вывода.

Программа воспринимает следующие параметры.

- `-n` или `-f n` или `--skip-fields=n` — пропустить `n` полей в каждой строке вводного файла до проверки на уникальность. Полем считается последовательность символов, *не*

содержащая пробелов или символов табуляции. Поля разделяются одним или более символами пробела или табуляции.

- `+n` или `-s n` или `--skip-chars=n` — пропустить n символов до проверки на уникальность. Если используется пропуск полей и одновременно пропуск символов, то вначале выполняется пропуск полей, а потом пропуск символов.
- `-c` или `--count` — вывести, сколько раз встретилась каждая строка.
- `-i` или `--ignore-case` — игнорировать регистр, в котором представлены символы во входном файле.
- `-d` или `--repeated` — вывести только повторяющиеся строки.
- `-u` или `--unique` — вывести только неповторяющиеся строки.
- `-w n` или `--check-chars=n` — сравнивать только n символов в каждой строке (после пропуска полей и символов). По умолчанию, после пропуска полей и символов, если таковые имели место, сравнивается остаток строки целиком.

Рассмотрим несколько простых примеров. Пусть у нас имеется тестовый файл с именем `T`:

```
$ cat T
12311 aabcd
45611 dc09
78911 wigs
21311 anka
32211 after
98722 gens
```

Команда `uniq -c T` выдаст следующий результат:

```
1 12311 aabcd
1 45611 dc09
1 78911 wigs
1 21311 anka
1 32211 after
1 98722 gens
```

что совершенно неудивительно, ведь входной файл не отсортирован. Но если применить команду `uniq +3 -w 2 -c T`, то получим

```
5 12311 aabcd
1 98722 gens
```

то есть программа найдет повторяющиеся части строк. Мы можем использовать команду `uniq -1 -w 2 -c T` и увидим, что программа пропустила только первое поле, но не пробел после него:

```
6 12311 aabcd
```

но если учесть это (`uniq -1 +1 -w 1 -c T`), то получим то, что и ожидалось:

```
1 12311 aabcd
1 45611 dc09
1 78911 wigs
2 21311 anka
1 98722 gens
```

Программа comm

Программа comm предназначена для сравнения отсортированных файлов. Синтаксис:

```
comm [option] file1 file2
```

Предполагается, что `file1` и `file2` уже отсортированы. Без параметров `option` программа `comm` производит вывод в три колонки. Самая левая колонка состоит из строк, которые содержатся только в файле с именем `file1`. Колонка 2 (средняя) состоит из строк, которые содержатся только в файле с именем `file2`. Третья колонка состоит из строк, которые являются общими для файлов с именами `file1` и `file2`. Колонки разделяются символами табуляции.

Могут использоваться следующие параметры: `-1`, `-2`, `-3`, которые означают запрет печати соответствующей колонки (нумерация колонок — слева направо).

Один из примеров использования состоит в сравнении содержимого каталогов или библиотек. Например, вы хотите сравнить два похожих на первый взгляд каталога: `/lib` и `/usr/lib`:

```
$ ls -1 /lib       > /tmp/l_lib
$ ls -1 /usr/lib > /tmp/l_usr_lib
$ comm /tmp/l_lib /tmp/l_usr_lib | less
```

В получившемся выводном файле размером 335 строк обнаружился только один файл, находящийся в обоих каталогах. Фрагмент вывода показан ниже:

```
...
libc.a
libc.so
        libc.so.5
        libc.so.5.4.38
        libc.so.6
                libc5-compat
libc_nonshared.a
        libcom_err.so.2
        libcom_err.so.2.0
...
```

В третьей колонке оказался единственный файл с именем `libc5-compat`. Естественно, что это ничего не говорит о содержании файла с данным именем в разных каталогах. (На самомом деле `libc5-compat` — это подкаталог.)

Аналогичным образом можно сравнивать содержимое двух библиотек

```
 $ ar -t /usr/lib/libc.a | sort > Dc
$ ar -t /usr/lib/libm.a | sort > Dm
$ comm Dc Dm
```

Приведем фрагмент вывода:

```
        s_ceill.o
s_chown.o
                s_copysign.o
                s_copysignf.o
                s_copysignl.o
        s_cos.o
        s_cosf.o
        s_cosl.o
```

Как видно, имеется часть программ с именами, которые встречаются в обеих библиотеках. Как и в предыдущем примере, мы ничего не можем сказать о самих программах.

Программа cut

Программа cut предназначена для печати указанных частей строк файла. Синтаксис:

```
cut option ... file ...
```

Программа выводит на устройство стандартного вывода часть каждой строки из входного потока.

Ниже используются следующие обозначения: `byte-list`, `character-list`, `field-list`. Каждое из них представляет собой последовательность чисел — одно или несколько чисел, разделенных запятой,— или же числовой интервал — два числа, разделенных знаком дефис.

Нумерация байтов, символов и полей начинается с 1. Можно использовать открытые интервалы, например, `-7` означает `1-7` (от 1 до 7), `9-` означает от 9 до конца строки.

Программа cut воспринимает следующие параметры:

- `-b byte-list` или `--bytes=byte-list` — вывести только байты в позициях, перечисленных в параметре `byte-list`. Символы `\t` и `\b` рассматриваются как 1 байт.
- `-c character-list` или `--characters=character-list` — вывести только символы в позициях, перечисленных в параметре `character-list`. Практически то же, что и параметр `-b`, но в будущем использование кодировки Unicode изменит это положение (коды символов занимают несколько байтов).
- `-f fiels-list` или `--fields=fiels-list` — вывести только поля, указанные в параметре `fiels-list`. По умолчанию разделителем полей является символ `\t`.
- `-d delim` или `--delimiter=delim` — используется совместно с параметром `-f` для указания символа `delim` для разделения полей. По умолчанию в качестве разделителя полей используется символ табуляции `\t`.
- `-n` — не разделять многобайтные символы.
- `-s` или `--only-delimited` — используется совместно с параметром `-f` для запрета печати строк, которые не содержат символ разделителя полей.

Программы expand и unexpand

Программы expand и unexpand предназначены для преобразования символов табуляции в сиволы пробела и обратно. Синтаксис:

```
expand [option ...] [file ...]
unexpand [option ...] [file ...]
```

Программа expand заменяет знаки табулятора \t на указанное количество пробелов. Программа unexpand производит обратное преобразование, то есть пытается заменить встреченные пробелы знаками \t.

Следует заметить, что такие преобразования могут оказаться весьма не безобидными: несмотря на то, что внешний вид текста практически не изменится — изменится его объем, а некоторые файлы (например, make-файлы) могут оказаться неработоспособными после обработки.

Программа expand воспринимает следующие параметры.

- `-tab1,tab2 ...` или `-t tab1,tab2 ...` или `--tabs=tab1, tab2 ...` — обработать знаки табуляции \t со значениями колонок табуляции tab1, tab2 (нумерация колонок начинается с 0) и т. д. Кроме того, заменить все знаки табуляции за пределами представленного списка знаками пробела. Если колонки табуляции указаны значениями параметров `-t` или `--tabs=`, то их можно разделять как запятыми, так и пробелами.
- `-i --initial` — преобразовать в пробелы только начальные символы табуляции в строках, то есть символы табуляции, которые непосредственно предшествуют символам, не совпадающим с пробелом или символом табуляции.

Программа unexpand по умолчанию заменяет на символы табуляции только начальные пробелы в каждой вводной строке. Символы \b уменьшают счетчик колонок для помещения знака \t. По умолчанию знак табуляции занимает 8 символов. Программа воспринимает следующие параметры:

- `-tab1,tab2` ... или `-t tab1,tab2` ... или `--tabs=tab1,tab2` ... — установить знаки табуляции \t со значениями колонок табуляции tab1, tab2 (нумерация колонок начинается с 0) и т. д. Кроме того, оставить без изменений все знаки пробелов и табуляторов за пределами представленного списка. Если колонки табуляции указаны значениями параметров `-t` или `--tabs=`, то их можно разделять как запятыми, так и пробелами.

- `-a` или `--all` — раменить, где возможно, знаками табуляции все пробелы в каждой строке.

Теперь приведем пример. Предположим, что нам надо сравнить два каталога, и установить, есть ли в них файлы с совпадающими именами.

```
$ ls -1 /lib     > /tmp/Lib
$ ls -1 /usr/lib > /tmp/Ulib
$ comm /tmp/Lib /tmp/Ulib | expand --tabs=10,50 | cut -b
51-70 | uniq -c
```

Выполнив приведенную выше последовательность действий, мы увидим, что имеется лишь одно общее имя в обоих каталогах — `libc5-compat`.

Программа tr

Программа tr предназначена для перекодировки и уплотнения последовательностей символов. Синтаксис:

```
tr [option]... set1 [set2]
```

Программа tr копирует входной поток со стандартного устройства ввода на стандартное устройство вывода, производя при этом одно из преобразований:

- перекодировка символов и, возможно, уплотнение повторяющихся символов;
- уплотнение повторяющихся символов (если встречается подряд два или более одинаковых символа, программа заменяет их одним символом);
- удаление символов;
- удаление символов и уплотнение повторяющихся символов.

Значениями параметра `set1` (и, возможно, `set2`) являются символьные строки.

Программа воспринимает следующие значения поля `option`.

- `-d` или `--delete` — удалить из входного потока те символы, которые содержатся в строке `set1`.
- `-s` или `--squeeze-repeats` — при использовании только этого параметра программа заменяет одним символом те повторяющиеся символы, которые содержатся в

строке set1. Если используется комбинация параметров --delete и --squeeze-repeats, то программа производит удаление тех символов из входного потока, которые содержатся в set1, затем производится удаление тех повторяющихся символов, которые содержатся в set2.

-c или --complement — использовать для операции те символы, которые не входят в строку set1.

- -t или --truncate-set1 — усечь строку set1, сделав ее равной по размеру строке set2 (усекается правый край строки).

Наборы символов

В символьных строках, служащих значениями параметров set1 и set2, используются некоторые специальные комбинации символов. Некоторые комбинации используются только в set1, а некоторые — только в set2. Каждая специальная комбинация начинается обратным слэшем (\). Специальные комбинации перечислены в табл. 4.23. Если за обратным слэшем будет следовать какой-то символ неуказанный в таблице, то возникнет ошибка.

Таблица 4.23. Специальные символьные комбинации, используемые tr

Символ	Значение
\a	Знак ^G
\b	Знак ^H
\f	Знак ^L
\n	Знак ^J
\r	Знак ^M
\t	Знак ^I
\v	Знак ^K
\nnn	Знак с восьмеричным кодовым представлением nnn
\\	Обратный слэш (\)

Правила указания наборов символов для команды tr

- **Интервалы.** Обозначение a-z обозначает все символы с a по z включительно. Первый символ в интервале должен иметь меньший код, иначе возникнет ошибка.

Например, запись `0-9` означает то же, что и `0123456789`. Обратите внимание, что интервалы не должны заключаться в квадратные скобки.

- **Повторяющиеся символы.** Запись `[C*n]` в `set2` означает, что символ `C` должен быть повторен `n` раз. Так, запись `[y*4]` обозначает то же, что `yyyy`. Запись `[C*]` в `set2` означает, что следует столько раз повторить символ `C`, чтобы сделать длину `set2` равной длине `set1`. Если `n` начинается с 0, тогда это число трактуется как восьмеричное, в противном случае — как десятичное.
- **Классы символов.** Запись `[:class:]` обозначает все символы из заранее определенного класса `class`. Принадлежность классу не предполагает какого-либо упорядочивания символов в классе, исключая классы `upper` и `lower`, в которых символы упорядочены по возрастанию значений кодов.

Имена классов, которые понимает программа `tr`, приведены в таблице 4.24.

- **Классы эквивалентности.** Запись `[=C=]` обозначает все символы, которые эквивалентны `C` в каком-то смысле. Появление классов эквивалентности является относительно новой возможностью, ее цель — поддержать не латинские алфавиты. Однако по всей видимости, не существует стандартного способа определить классы или их содержание. Поэтому GNU tr не имеет полной реализации классов эквивалентности. Пока `tr` поддерживает лишь классы эквивалентности, состоящие из одного символа. Так, `[=c=]` означает `c`, а `[=b=]` означает `b` и т. д.

Таблица 4.24. Имена классов, используемые программой tr

Обозначение класса	Значение
alnum	Буквы и цифры
alpha	Буквы
blank	Пробелы
cntrl	Управляющие символы
digit	Цифры
graph	Отображаемые символы (не включая пробелы)

Обозначение класса	Значение
lower	Буквы в нижнем регистре
print	Отображаемые символы (включая пробелы)
punct	Символы знаков пунктуации
space	Горизонтальные и вертикальные пробелы
upper	Буквы в верхнем регистре
xdigit	Шестнадцатеричные цифры

Все рассуждения, которые относятся к способу изображения символов, верны только для знаков латинского алфавита.

Перекодировка (трансляция) символов

Программа `tr` выполняет перекодировку в том случае, когда присутствуют `set1` и `set2`, но не присутствует параметр `--delete` (`-d`). Программа перекодирует только те символы входного потока, которые присутствуют в строке `set1`. Каждый символ входного потока, который присутствует в `set1`, заменяется на соответствующий символ `set2` (первый в первый, второй во второй и т. п.). Если символ входного потока не представлен в строке `set1`, то он передается на устройство стандартного вывода без изменений. Если символ встречается в строке `set1` дважды, то берется последнее значение. Например, нижеследующие две команды эквивалентны:

```
$ tr mmmm 1234
$ tr m 4
```

Приведем пример различных вариантов перекодировки строчных букв в прописные:

```
 $ tr abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
$ tr a-z A-Z
$ tr '[:lower:]' '[:upper:]'
```

При перекодировке строки `set1` и `set2` обычно задаются одинаковой длины. Если длина этих строк отличается, то возникают неоднозначности, которые могут интерпретироваться в разных версиях программы по-разному, то есть о переносимости использования `tr` с параметрами `set1` и `set2` разной длины следует заботиться отдельно.

GNU-версия программы tr ведет себя следующим образом:

- Если `set1` короче `set2`, то остаток `set2` просто игнорируется;
- Если `set1` длиннее `set2`, то по умолчанию `tr` "растягивает" `set2` до длины `set1` путем дублирования последнего символа `set2` столько раз сколько необходимо.
- Если `set1` длиннее `set2` и использован параметр `--truncate-set1` (`-t`), то строка `set1` будет усечена до длины `set2`. Этот параметр действует только для операции перекодировки.

Примеры использования tr

Рассмотрим примеры работы программы tr. Предположим, что файл с именем T содержит строку `\subsection{Простые примеры}`.

```
$ head T | tr -cs A-Za-z0-9 '\012'
subsection
$ head T | tr -tcs A-Za-z0-9 '\012'
\subsection{Простые примеры}
```

В первой строке восьмеричная кодовая комбинация `\n` расширена до размеров `A-Za-z0-9`, все символы не совпадающие с `A-Za-z0-9`, заменены на `\n` (в данном случае все специальные знаки и символы кириллицы), из выводного потока удалены идущие подряд символы `\n`.

Во втором случае первый параметр `set1` усечен до одного символа (до длины `set2`), а первым символом является восьмеричный код `000`. Поскольку в вводном файле не было символа с кодом `000`, то мы и не видим изменений.

Уплотнение и удаление символов

Уплотнение символов подразумевает, что программа tr заменяет любые два и более одинаковых символа на один символ. Например, для замены всех повторяющихся переводов строк на один перевод строки можно воспользоваться командой:

```
$ tr -s '\n'
```

Чтобы удалить все символы с восьмеричным кодом `000`, используйте команду:

```
$ tr -d '\000'
```

Іреобразовать все знаки кроме букв и цифр в знаки \n и плотнить повторяющиеся знаки \n:

```
$ tr -cs '[a-zA-Z0-9]' '[\n*]'
```

Іиагностические сообщения

ї ряде случаев программа tr выдает диагностические сообщения, которые не оговорены стандартом POSIX. Чтобы збежать этих сообщений, следует установить переменную кружения `$POSIXLY_CORRECT`.

Ірограмма paste

Синтаксис:

```
aste [option]... [file]...
```

Ірограмма читает указанные файлы (один или больше) и ыводит на устройство стандартного вывода строки текста, оторые состоят по умолчанию из объединенных строк данх файлов — первая строка составляется из первых строк водных файлов, разделенных символами \t, вторая стро- а — из вторых и т. д. Программа распознает несколько паметров:

`-s` или `--serial` — выводить последовательно строки одного файла, а лишь после второго и т. д., при этом все строки первого файла будут объединены (между строками будет вставлен знак табуляции) в одну длинную строку.

`-d delim-list` или `--delimiters delim-list` — последовательно использовать символы из `delim-list` вместо символа \t, чтобы отделить объединяемые строки файлов. Если список `delim-list` исчерпан, то он начинает использоваться сначала.

Ірограмма join

Ірограмма join позволяет объединить строки по общим оям. Синтаксис:

```
oin [option]... file1 file2
```

Файлы с именами `file1` и `file2` должны содержать уже гсортированные файлы в возрастающем порядке (не чисовом) по объединяемым полям. Если не указан параметр

`-t`, то они должны быть отсортированы, игнорируя пробелы в начале объединяемого поля (как после использования программы sort) с параметром `-b`. Если используется параметр `--ignore-case`, то строки должны быть отсортированы, игнорируя регистр символов в поле, по которому будет производиться объединение (как после использования программы sort с параметром `-f`).

По умолчанию объединяемое поле является первым полем в каждой строке; поля на вводе разделены одним или более пробелами, ведущие пробелы в строках игнорируются; поля на выводе разделяются пробелом; каждая выводимая строка состоит из объединяемого поля, за которым следует остаток строки из файла `file1`, а затем остаток строки из файла `file2`.

Программа join воспринимает значения поля `option`, которые представлены в таблице 4.25.

Таблица 4.25. Значения поля `options` для программы join

Значение	Описание
-a file-number	Вывести строку для каждой непарной строки в файле file1 или file2 в дополнение к обычному выводу
-e string	Заместить отсутствующие поля на вводе подстрокой string
-i --ignore-case	Игнорировать регистр символов. Вводимые строки должны быть отсортированы также вне зависимости от регистра символов. Для такой сортировки можно использовать sort -f
-1 field-j1 field	Объединить по полю field (положительное целое) файла file1
-2 field-j2 field	Объединить по полю field (положительное целое) файла file2
-j field	Эквивалентно -1 field и -2 field
-o field-list...	Сконструировать каждую строку в соответствии с форматом field-list. Каждый элемент field-list может состоять из одиночного символа О или иметь форму M.N, где M – 1 или 2 (номер файла), а N – номер поля. Поле, обозначенное как О, является полем объединения. В большинстве случаев функциональность спецификации с помощью О может быть реализована с помощью явного описания типа M.N, которое соответствует полю объединения. Однако если встречаются непарные строки во вводных файлах, то описание типа M.N не имеет возможности описать поле объединения. Элементы в field-list разделены запятыми или пробелами. При использовании параметра -о можно указать несколько полей field-list – все они будут объединены. Все выводимые строки, включая те, которые сгенерированы параметрами -а и -v, являются объектами описания в field-list

Значение	Описание
t char	Использовать символ char в качестве символа разделителя полей
v file-number	Вывести строку для каждой непарной вводной строки в файле file-number (1 или 2) вместо обычного вывода

одсистема сканирования, анализа обработки текстов awk

Введение

Программа awk является интерпретатором специального зыка awk для контекстного поиска и анализа текста. awk ыла написана в 70-х годах создателями операционной системы Unix. Позднее в рамках проекта GNU была создана GNU-реализация awk, носящая имя gawk. Поскольку практически во всех современных Linux команда `awk` является сылкой на программу gawk, то в данной книге под awk мы удем понимать именно GNU-версию — gawk. Подробное писание программы следует искать на страницах руководств `an awk` и `info awk` или в книгах на эту тему, например, The AWK Programming Language», (A. V. Aho, P. J. Weinerger, B. W. Kernighan; Addison-Wesley, 1988, ISBN 0-201-7981-X), или «Sed and Awk» (Dale Doughberry; O'Railly, 992, ISBN: 0-937175-59-5). Полезная информация может ыть найдена в Интернете по адресу http://mx.nsu.ru/FAQ/F-omputer-lang-awk/.

Напомним еще раз, что мы будем рассматривать GNU-верию awk, а именно — GNU Awk 3.0.3.

Основное назначение программы awk — интерпретация язы-а программирования awk, который благодаря простоте поволяет быстро создать программу-прототип для анализа и реобразования текстовых файлов.

Типичный awk-сценарий построен по следующей схеме:

```
аблон1 \{действие1\} шаблон2 \{действие2\} ... шаблонN {действиеN\}
```

В соответствии с этой схемой программа читает входной айл (или устройство стандартного ввода) и анализирует

каждую строку на предмет соответствия указанному шаблону. Если соответствие имеет место, то производятся действия, описанные в фигурных скобках (быть может, весьма нетривиальные).

Существуют два частных упрощенных случая. Если шаблон отсутствует, то указанное действие применяется к каждой строке. Если отсутствует действие, то все строки, соответствующие шаблону, выводятся на стандартное устройство вывода.

Простые примеры использования awk

В первом примере мы напечатаем третье поле записи (третью колонку, или третье слово записи, т. е. в примере ниже — имя пользователя):

```
$ ls -l | awk '{print($3)}'
```

Легко понять, что данный пример можно использовать для определения того, кому принадлежат файлы в текущем каталоге:

```
 $ ls -l | awk '{print($3)}' | sort | uniq -c
```

В результате будет напечатан список пользователей владельцев файлов в текущем каталоге, а также указано, сколько файлов принадлежат каждому пользователю.

Для печати сначала девятой, а затем первой колонки (имя файла и его описатель) можно сделать так:

```
$ ls -l | awk '{print($9,$1)}'
```

Для вставки дополнительных пробелов между выводными колонками используем

```
$ ls -l | awk '{print($9,"  ",$1)}'
```

или

```
$ ls -l | awk '{print("Колонка 5=",$5," А это колонка 1
",$1)}'
```

В предыдущих примерах предполагалось, что разделителем полей является пробел. Если используется другой символ, то это можно указать

```
$ awk -F: '{print($5,"  ",$1)}' FileName
```

Здесь в качестве разделителя полей используется двоеточие.

Наконец, если использовать оператор `print` без параметров, то входная строка будет напечатана полностью:

```
$ awk '{print}' FileName
```

Если вы захотите использовать готовый к интерпретации файл, содержащий программу на языке awk, то этого можно достичь двумя путями. Один из способов показан в нижеследующем примере:

```
$ awk -f awkprogram FileName
```

В этом примере файл с именем `awkprogram` содержит программу на языке `awk`, а файл с именем `FileName` содержит анализируемый текст.

Второй способ состоит в использовании в awk-сценарии нотации `#!/usr/bin/awk -f`, после чего awk-программу можно будет запускать непосредственно (не забудьте при помощи команды `chmod +x` разрешить ее выполнение!):

```
$ AwkProgram FileName
```

В этом случае файл с именем `AwkProgram` может содержать, например, нижеследующее:

```
#!/usr/bin/awk -f
{print($3)}
```

Файл с именем `FileName` содержит анализируемый текст.

Запуск awk

Программа GNU awk воспринимает параметры, перечисленные в табл. 4.26.

Таблица 4.26. Параметры командной строки awk

Параметр	Значение
-F fs --field-separator fs	Использовать значение fs в качестве разделителя полей
-v var=val --assign var=val	Перед выполнением программы переменной с именем var присвоить значение val. Значения переменных будут доступны уже в блоке BEGIN
-f program-file -file program-file	Считать исходный текст программы на языке awk из файла с именем program-file. Можно использовать несколько параметров -f и, соответственно, несколько файлов с программами. Таким образом, можно использовать библиотеку полезных awk-программ

продолжение

Таблица 4.26 *(продолжение)*

Параметр	Значение
-W lint --lint	Вывести предупреждения при использовании в программе конструкций, не переносимых в другие реализации awk
-W re-interval --re-interval	Разрешить использование шаблонных интервалов (смотрите раздел «»)
-W source program-text --source program-text	Рассматривать program-text как исходный текст на языке awk. Этот параметр можно использовать совместно с параметром -f. Такое совместное использование позволяет использовать библиотеки awk-программ совместно с awk-операторами в командной строке. Пример: ls -l | awk --source 'print' --file library-file В этом примере выполняется сначала программа awk из командной строки (в данном случае – команда print), а затем awk-программа из файла library-file

Более подробное описание параметров вы можете найти в руководстве, для вызова которого используйте команду `info gawk`.

Переменные, записи, поля

Переменные

Переменные awk являются динамическими; они создаются в момент первого обращения к ним. Значениями переменных могут быть целые, плавающие числа или символьные строки, что определяется контекстом использования переменной. Имеются одномерные массивы. Многомерные массивы могут быть смоделированы с помощью одномерных массивов. Несколько переменных, имеющих специальное значение, устанавливаются по ходу выполнения программы. Имена и значения таких переменных перечислены в разделе awkvar.

Записи

Обычно предполагается, что записи отделены одна от другой символом `\n`. В специальной переменной `RS` хранится текущее значение символа разделения записей, которое может быть изменено. Значения, содержащиеся в `RS`, могут интерпретироваться по-разному. Если `RS` содержит одиночный символ, то это есть символ разделения записей. Если

RS содержит более чем один символ, то это регулярное выражение. В результате любой текст, который удовлетворяет регулярному выражению является разделителем записей.

Поля

Программа awk читает входной файл по записям. Каждую запись awk разбивает на поля или слова, используя символ разделения полей, который содержится в переменной FS. Если FS содержит один символ, то этот символ является разделителем полей. Если FS содержит пустую строку, то каждый символ введенной записи считается отдельным полем. Если FS содержит более чем один символ, то эта строка рассматривается как регулярное выражение, которое используется для разбиения введенной записи на поля.

Встроенные переменные awk

awk позволяет использовать в программах ряд встроенных переменных. Следует заметить, что речь идет не о переменных окружения, а о внутренних переменных awk.

Массивы

Как уже упоминалось выше, массивы в языке awk являются одномерными. Массивы являются ассоциативными массивами, то есть значениями индексов являются строки (последовательности символов). Индексы массива располагаются в квадратных скобках. Например, выражение

```
k="A"; l="B";: \
x[k, l] = "Привет, ребята \n"
```

запишет строку Привет ребята \n в элемент массива x, который имеет в качестве индекса строку A\034B, то есть сцепленные символы A и B и значение переменной SUBSEP между ними.

В условных операторах awk можно использовать следующие конструкции:

```
if (Val in array) print array[Val]
```

то есть если в массиве array имеется индекс Val, то вывести значение элемента массива, соответствующего индексу

Val. Если массив имеет в качестве индекса, как показано выше, k, l, то следует использовать такую запись:

```
if ((k, l) in array) print array[k, l]
```

Встроенные функции языка awk

Язык awk имеет набор встроенных функций для выполнения некоторых, в известном смысле «стандартных», операций.

Строковые функции

Таблица 4.27. Строковые функции языка awk

Функция	Значение
gensub(r, s, h[, t])	Предназначена для поиска в строке t последовательности, которая удовлетворяет регулярному выражению r. Если строка h содержит в качестве первого символа знак g или G, то каждая найденная последовательность, соответствующая r заменяется на s. В противном случае, строка h должна быть числом, указывающим, какая по порядку последовательность, соответствующая выражению r, должна бытьзаменена на s. Если строка t опущена, то вместо нее используется $0. В пределах строки s могут быть использованы комбинации вида \n, где n есть целое от 0 до 9 включительно. Если n от 1 до 9, то такая комбинация указывает на текст в n-х по порядку скобках. Комбинация \0, аналогично символу &, обозначает найденный текст, соответствующий регулярному выражению r. В отличие от функций sub() и gsub(), модифицированная строка возвращается как результат функции, а строка t не изменяется
gsub(r, s [, t])	Каждую подстроку в строке t, соответствующую выражению r, заменить на строку s. В качестве значения функции возвращается количество произведенных замен. Если строка t опущена, то используется $0. Знак & (амперсанд) в подстановке заменяется на найденый текст, соответствующий выражению r. Если вы хотите использовать символ амперсанд в своем непосредственном значении, то его следует экранировать: \&. (Если вы собираетесь интенсивно использовать знака амперсанд, то полезно прочесть разделы описания языка awk, касающиеся использования специальных комбинаций символов, начинающихся обратным слэшем.)
index(s, t)	Вернуть индекс (смещение от начала) строки t в строке s, если строка t содержится в строке s. В противном случае вернуть 0
length[(s)]	Вернуть длину строки s или длину $0, если s опущена
match(s, r)	Вернуть позицию, с которой начинается последовательность, соответствующая регулярному выражению r. Если такой последовательности нет, возвращается 0. Функция также устанавливает значения переменных RSTART и RLENGTH

Функция	Значение
split(s, a [, r])	Разбить строку s на части и сохранить эти части в массиве a. Разбиение строки s производится, используя в качестве разделителя регулярное выражение r. Если выражение r не задано, используется значение переменной FS. Перед началом разбиения строки массив a очищается
sprintf(fmt, expr-list)	Отформатировать строку expr-list в соответствии с форматом fmt. Форматы задаются так же, как в функции языка C printf. Например, $ awk -v A=192 'print(sprintf("%20d",A))' t где t – имя файла, A – имя переменной, которой присваивается значение 192. Это значение форматируется в соответствии со спецификацией %20d
sub(r, s [, t])	Функция делает то же, что и gsub(), но заменяется только первая подходящая подстрока
substr(s, i [, n])	Функция в качестве своего результата возвращает подстроку, которая начинается в позиции i строки s с максимальной длиной в n символов. Если параметр n не указан, то возвращается часть s, начиная с позиции i
tolower(str)	Вернуть строку str, преобразовав все символы в строчные. Неотображаемые символы и числа не преобразуются
toupper(str)	Вернуть строку str, преобразовав все символы в прописные буквы. Неотображаемые символы и числа не преобразуются

Функции времени

Для получения текущего значения времени awk имеет две функции:

```
systime()
```

Возвращает количество секунд, прошедших с 1 января 1970 года.

```
strftime([format [,timestamp]])
```

Возвращает отметку времени (`timestamp`), отформатированную в соответствии с форматом `format`. Строка отметки времени должна иметь тот же вид, что и возвращаемое функцией `systime()` значение. Если строка `timestamp` опущена, то используется текущее время. Если строка `format` опущена, то используется формат, который применяется в программе `date`. Для получения списка допустимых форматов можно обратиться к спецификации функции `strftime()` в ANSI C.

Математические функции

Помимо функций, непосредственно связанных с обработкой текста, awk предоставляет в распоряжение пользователя математические функции.

Таблица 4.28. Математические функции awk

Функция	Значение
atan2(y, x)	Возвращает значение arctan(y/x) в радианах
cos(expr)	Возвращает значение косинуса expr радиан
exp(expr)	Возвращает значение экспоненту expr
int(expr)	Возвращает expr с отброшенной дробной частью
log(expr)	Возвращает значение натурального логарифма от expr
rand()	Возвращает случайное число в диапазоне от 0 до 1
sin(expr)	Возвращает значение синуса expr радиан
sqrt(expr)	Возвращает квадратный корень из expr
srand([expr])	Устанавливает новое исходное значение для генератора случайных чисел. Возвращает предыдущее исходное значение. Если expr опущено, то используется текущее время в секундах. Таким образом, если выхелаете, чтобы у вас при каждом новом запуске awk функция rand() генерировала новую псевдослучайную последовательность, то перед обращением к функции rand() достаточно вызвать функцию srand() (без параметров)

Определение новых функций

В программах awk также можно определять пользовательские функции. Определение функций выполняется следующим образом:

```
function name(parameter list) \ statements \
```

Внутри тела функции можно определить локальные переменные, которые должны быть описаны в поле параметров, но отделены от них более чем одним пробелом. Например,

```
function  MyFunction(i,j,   a,b,c)
# a, b и c являются локальными переменными.
{
...
}
{MyFunction(43,61)}
```

Отметим, что левая скобка в вызове функции должна следовать сразу за именем функции (без пробела).

Виды шаблонов, используемые в awk

Программа awk допускает следующие виды шаблонов:

- BEGIN
- END

- `/регулярное выражение/`
- `условное выражение`
- `шаблон1 && шаблон2`
- `шаблон1 || шаблон2`
- `шаблон1 ? шаблон2 : шаблон3`
- `(шаблон)`
- `! (шаблон)`
- `шаблон1, шаблон2`

`BEGIN` и `END` являются специальными видами шаблонов, которые не проверяются на соответствие с записями входного потока. Если встречен шаблон `BEGIN`, то соответствующее ему действие будет выполнено лишь однажды до начала чтения вводного файла. Этот шаблон удобно использовать для установки каких-либо исходных значений. Если встречен шаблон `END`, то соответствующее ему действие тоже выполняется однажды, после достижения конца файла во входном потоке. Для каждого из шаблонов `BEGIN` и `END` действие не может быть опущено.

Регулярные выражения уже рассматривались в разделе «Поиск по шаблону и регулярные выражения». awk понимает некоторые дополнительные специальные обозначения в регулярных выражениях. Для уточнения конкретных деталей обратитесь к описанию awk.

Операции `&&`, `||` и `!` обозначают соответственно логические «и», «или» и «не», как и в языке C.

Оператор `?:` интерпретируется так же, как в C. Если `шаблон1` не пуст, то для сопоставления с вводной строкой используется `шаблон2`, в противном случае используется `шаблон3`. Простой пример: если в качестве первого шаблона используется `$9`, то сравнение со вторым шаблоном будет происходить тогда и только тогда, когда во входной строке присутствует девятое поле.

Для указания последовательности выполнения логических операций можно использовать скобки.

Форма `шаблон1, шаблон2` называется шаблонным интервалом. Шаблонному интервалу удовлетворяют последовательные строки входного файла, которые начинаются первой строкой, удовлетворяющей `шаблону1`, и заканчиваются первой строкой, удовлетворяющей `шаблону2`.

Действия в awk

Действия должны быть заключены в фигурные скобки. В таблице 4.29 перечислены операторы, которые могут использоваться при описании действий.

Управляющие операторы

Таблица 4.29. Управляющие операторы awk

Оператор	Значение
if (условие) оператор1 [else оператор2]	Если выражение условия истинно, то выполнить оператор1, в противном случае – оператор2 (если он указан)
while (условие) оператор	Выполнять оператор в цикле до тех пор, пока условие истинно
do оператор while(условие)	Выполнять оператор в цикле до тех пор, пока условие истинно
for (выражение1; выражение2; выражение3) оператор	Выполняется следующая последовательность действий: 1. Сначала выполнить выражение1. 2. Затем, если выражение выражение2 истинно, то выполнить оператор. 3. Затем выполнить выражение3. 4. Перейти к пункту 2, Например, следующая команда выведет первые две колонки входного потока: $ awk ' for (i = 1; i <= 2; i++); print $i '.
for (var in array)оператор	Выполнить оператор для всех значений из массива array. Здесь var – имя переменной цикла
break	Выйти из цикла
continue	Перейти к следующему значению переменной цикла
delete array[index]	Удалить элемент массива
delete array	Удалить весь массив
exit [выражение]	Выйти из функции. Выражение определяет код возврата
(оператор1; оператор2; ...)	Составной оператор

Операторы ввода/вывода

Таблица 4.30. Операции ввода/вывода в awk

Оператор	Значение
close(file)	Закрыть файл или программный канал с именем file
getline	Записать в переменную $0 следующую строку. Также будут установлены значения переменных NF, NR и FNR
getline < file	Записать в переменную $0 содержание следующей записи файла с именем file; установить переменную NF
getline var	Записать в переменную var содержание следующей записи из входного потока; установить переменные NR и FNR
getline var < file	Записать в переменную var содержание следующей записи из файла с именем file
next	Остановить обработку текущей записи и прочесть следующую запись. Обработку начать с первого шаблона
nextfile	Остановить обработку текущего входного файла. Следующую запись прочесть из следующего входного файла. Установить переменные FILENAME, ARGIND, а также установить значение переменой FNR равной 1
print	Вывести текущую запись на устройство стандартного вывода
print expr-list	Вывести выражение expr-list на устройство стандартноговывода
print expr-list > file	Вывести выражение expr-list в файл с именем file
printf fmt, expr-list	Отформатировать выражение expr-list в соответствии с форматом fmt и вывести его на устройство стандартного вывода
printf fmt, expr-list >file	Отформатировать выражение expr-list в соответствии с форматом fmt и вывести его в файл с именем file
system(cmd-line)	Выполнить команду cmd-line и вернуть код ее завершения
fflush([file])	Вывести все буферы, связанные с файлом file. Если аргумент file опущен, то подразумевается стандартное устройство вывода. Если file есть пустая строка (то есть нотация команды fflush()а, то будут выведены буферы всех файлов и программных каналов, открытых на запись

Дополнительные примеры

Приведем несколько простых примеров использования awk.

Для того чтобы вывести из файла `WgetWhere.tex` строки, содержащие слово `ftp` и слово `http` одновременно, воспользуйтесь следующей командой:

```
$ awk '/ftp/ && /httprint' WgetWhere.tex
```

Чтобы просуммировать объем файлов в текущем каталоге:

```
$ ls -l | \
> awk 'BEGIN{a=0} {if (index($1,"d") == 0) a=a+$5 } \
> END{print a}'
```

Чтобы просуммировать объем файлов в текущем каталоге и вычислить средний размер файла:

```
$ ls -l | \
> awk 'BEGIN{a=0;b=0} { if (index($1,"d") == 0)
{a=a+$5;b=b+1} } \
> END{printf "%s,%10d,%s,%10d,%s", \
> "Объем=",a," Средний размер=",a/b,"\n"}'
```

Заключительные замечания по поводу awk

В заключение скажем, что поскольку программа awk является интерпретатором, то она исполняет программы на языке awk примерно с такой же скоростью, как, например, оболочка bash свои команды. Конечно, awk-программы работают медленнее, чем программы на C. Иными словами, если вы реализуете один и тот же алгоритм на awk и на C, то реализация на C будет выполняться в несколько раз быстрее. С другой стороны, составить на awk программу для анализа и преобразования текста во многих случаях проще и быстрее.

Для того чтобы использовать преимущества обоих языков (awk и C), имеется несколько вариантов программных конвертеров с языка awk на язык C. Один из них, awk2c, доступен по адресу ftp://sunsite.unc.edu/pub/Linux/utils/text/awk2c050.tgz. Информацию о других конвертерах (awka, awkcc) и не только можно найти по адресу http://ftp.umr.edu/pub/faqs/text/computer-lang/awk/faq. Существует также конвертер a2p для перевода awk-программ на язык Perl.

5. Система поддержки версий текстов CVS

Аббревиатура CVS означает Concurrent Versions System (многопоточная система версий). CVS представляет собой программную систему, которая помогает поддерживать много вариантов исходных текстов, над которыми работает одна или несколько групп разработчиков. Применяя эту систему, вы можете детально отслеживать историю изменений ваших исходных текстов, а также производить массу других операций по контролю за правильностью смены версий исходных.

CVS позволяет на основе хранимой истории воссоздать в любой момент любую версию исходных текстов, которая была в прошлом. Это верно как для всей совокупности исходных текстов, так и для отдельных файлов.

CVS предоставляет возможности надежного доступа к исходным текстам с других компьютеров в Интернете. Разработчики на удаленных узлах могут выполнять те же операции над текстами в хранилище, что и локально. Поддерживается режим параллельных разработок, когда разные программисты имеют возможность относительно независимо корректировать одни и те же исходные тексты в одно и то же время.

Отслеживание истории изменения часто помогает найти ошибки и неточности в большом комплексе программ. Вы сможете легко вернуться к прежним версиям ваших программ, или просто перейти от одного варианта текста программ к слегка модифицированному, подготовленному для других целей.

Исходные тексты могут иметь любой вид и назначение. Вы можете писать роман или программировать, или просто готовить отчет. Любой текст и его варианты могут быть сохранены с помощью CVS.

Роль такой системы трудно переоценить, когда коллектив разработчиков постоянно меняется, но в то же время требуется сохранять и поддерживать в продолжающемся проекте полезные наработки, полученные разными людьми на разных этапах проекта, а также привнесенные извне.

Хранение многих вариантов может потребовать массу дисковой памяти, и CVS предпринимает все меры для экономии места на диске. Так, при поддержке нескольких вариантов одного текста CVS хранит лишь базовый текст и изменения к нему.

CVS начиналась как набор сценариев, которые подготовил Dick Grune в 1986 году. В 1989 году работа в данном направлении была продолжена двумя людьми: Brian Berliner и Jeff Polk.

Обычно CVS является частью стандартной поставки Linux. Она свободно распространяется в Интернете. Можно отметить следующие серверы, на которых можно найти CVS:

http://www.cyclic.com/

http://www.loria.fr/~molli/cvs-index.html

Для публичного обсуждения вопросов, касающихся CVS, имеется группа новостей `news:comp.software.config-mgmt`.

Модель работы CVS

Перед дальнейшим обсуждением полезно отметить некоторые особенности модели работы системы CVS.

Все тексты хранятся в специальной системе каталогов CVS, которое называется *хранилище*, или *репозитарий*. *Хранилище* создается средствами системы CVS. К одному *хранилищу* могут иметь доступ несколько разработчиков.

Система позволяет обращаться к *хранилищу*, используя имя каталога в *хранилище*, имя отдельного файла или *модуля*. *Модуль* представляет собой группу файлов и/или каталогов, которые система CVS воспринимает как единый объект. Модуль определяется путем редактирования конфигурационного файла `modules`.

Тексты представляют собой, например, систему программ, которая разрабатывается и корректируется значительное

время, может несколько лет. За время разработки и корректировки может выпускаться несколько версий исходных текстов, может меняться состав разработчиков. При смене разработчиков особенно важно документировать изменения, происходящие в исходных текстах программ.

Когда разработчик планирует корректировать тексты какой-то программы, то он с помощью средств CVS копирует файлы из *хранилища CVS* в свой *рабочий* (текущий) каталог, создавая *рабочую копию* исходных текстов. По завершении корректировок, обновленные модули помещаются командами CVS в хранилище. При этом старая версия обновленных текстов также остается в хранилище. Таким образом, несколько разработчиков могут работать с одной и той же программой (обычно с разными частями программы) независимо друг от друга.

Хранилище может находиться как на той же машине, где работают разработчики, так и в любом месте в Интернете. Общая схема такой работы показана на рис. 5.1.

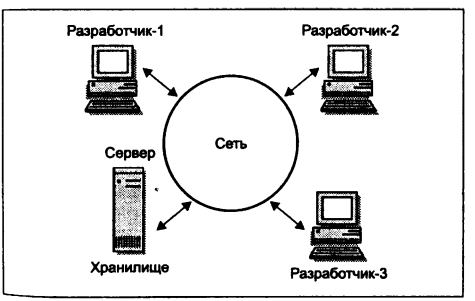


Рис. 5.1. Коллективная работа с использованием CVS

CVS поддерживает возможность слежения за любым файлом (watches), которое выражается в уведомлении разработчика, когда кто-то другой запросил копию этого файла, или получить список разработчиков, которые в данный момент работают с этим файлом.

Разработчик может пометить любой файл в хранилище как двоичный, чтобы запретить преобразования, которые имеют место при передаче текстовой информации.

Система позволяет также управлять всем процессом разработки. С использованием конфигурационных файлов `commitinfo`, `loginfo`, других можно установить автоматический вызов сценариев, при выполнении определенных действий в хранилище. В сценариях могут быть как автоматическая рассылка уведомлений о конкретных операциях всем разработчикам, так и контроль правильности использования тех или иных параметров.

Простые примеры использования CVS

Чтобы понять, как работает CVS, мы пройдем через некоторые типовые шаги. Первое, что полезно понять, — CVS хранит данные в специально организованном *хранилище* (*репозитарии*).

Данное обсуждение предполагает, что хранилище существует и его имя задано в переменной окружения `$CVSROOT`.

Положим, что вы разрабатываете программу (имя программы `tc`) на языке C. Заметим, что хотя программа и является небольшой, но все-таки состоит из нескольких модулей и одного файла `Makefile` для построения исполняемой программы.

Получение исходных текстов из хранилища

Первый шаг, который вы должны сделать, — это получить исходные тексты вашей программы из *хранилища* и поместить их *рабочую копию* в вашем *рабочем каталоге*. Это выполняется командой

```
cvs checkout tc
```

здесь:

- `cvs` — основная программа системы;
- `checkout` — параметр, команда системы CVS;
- `tc` — как мы помним, имя вашей программы.

По этой команде будет создан новый каталог с именем `tc` и в него будут скопированы исходные тексты программы `tc` из хранилища CVS. После выполнения этой команды вы можете перейти в каталог `tc`

```
cd tc
```

Если, далее, вы выполните команду `ls`, то увидите что-то похожее на нижеследующее:

```
$ ls
CVS    Makefile  backend.c   driver.c  frontend.c  parser.c
```

Здесь мы видим имена исходных модулей, которые составляют программу `tc` и каталог с именем `CVS`, который автоматически создан программой CVS во время выполнения команды `checkout`. Каталог необходим самой системе CVS, поэтому не стоит его удалять или модифицировать.

Далее, вы можете редактировать ваши исходные тексты, отлаживать программу `tc` и, вообще, делать то, что вы обычно делаете во время разработки или отладки.

После отладки и правки исходного текста модуля `backend.c` вы решили поместить его обновленный вариант в основное хранилище. Вы можете сделать это посредством команды

```
cvs commit backend.c
```

при этом будет автоматически вызван редактор текстов для того, чтобы дать вам возможность ввести комментарий по поводу изменения исходного текста. Вы можете, например, написать: `Добавлена возможность обхода ошибок`. Далее, вы выходите из редактора, сохранив файл с вашим комментарием, и CVS записывает все это (ваш новый вариант исходного текста и ваш комментарий) в хранилище CVS. Заметим, что поскольку вызывается редактор текста, то вводимый вами комментарий может быть весьма обширным. Например, он может состоять не из одной фразы, а из нескольких абзацев.

Имя редактора текста, который автоматически вызывает система, хранится в переменной окружения `$CVSEDITOR`. Если переменная `$CVSEDITOR` не установлена, то принимается переменная окружения `$EDITOR`. Если переменная `$EDITOR` тоже не установлена, то вызывается редактор по умолчанию, определенный для вашей операционной системы.

Во время автоматического вызова редактора по команде CVS `commit` вы можете увидеть список файлов, которые возможно были изменены. Этот список базируется на времени модификации файла. Если время модификации рабочей копии в вашем рабочем каталоге отличается от времени модификации этого же файла в хранилище CVS, то файл будет на подозрении, что он модифицирован. Однако во время операции `commit` система проверит, действительно ли файл был изменен или была модифицирована лишь дата модификации файла. Если была лишь изменена дата, то время модификации файла в хранилище CVS не будет изменено.

Если комментарий краток, вы можете избежать вызова редактора текста для ввода комментария:

```
cvs commit -m "Добавлена возможность обхода ошибок"
backend.c
```

Удаление рабочего каталога

По завершении работы с программой tc вы можете пожелать перейти к следующей работе. Тем самым вам может понадобиться удаление рабочего каталога (ведь все ваши изменения уже находятся в хранилище CVS!), чтобы не засорять ваш основной каталог.

Непосредственно перед удалением каталога полезно использовать команду `release` системы CVS, которая проверит состояние вашего рабочего каталога на предмет соответствия хранилищу. Ниже приводится последовательность: команда `cd` — вход из каталога `tc` в родительский каталог, выполнение команды `release` CVS, затем приведена диагностика времени выполнения команды.

```
$ cd ..
$ cvs release -d tc
  M driver.c
  ? tc
  You have [1] altered files in this repository.
  Are you sure you want to release (and delete) module
`Tc' n
  ** Release' aborted by user choice.
```

Здесь:

```
M driver.c
```

означает, что данный файл был модифицирован после того, как был скопирован из хранилища CVS. Следующая строка

```
? tc
```

означает, что файл `tc` неизвестен CVS. Поскольку файл с именем `tc` — это готовая к исполнению программа, то нет необходимости помещать ее в хранилище CVS. Следовательно, данное сообщение можно игнорировать. Последняя строка приведенного примера означает, что операция удаления была отменена.

Таким образом, команда `release` проверяет каждый файл в каталоге и просит пользователя подтвердить удаление файла.

Раз мы забыли, что были изменения в файле `driver.c`, то полезно установить, что именно изменилось, например, командой:

```
cd tc
cvs diff driver.c
```

Иными словами, мы снова перешли в каталог `tc`, удаление которого было отменено, и выполнили команду `diff` системы CVS. По этой команде система сравнивает рабочую копию модуля `driver.c` с той копией этого файла, которая содержится в хранилище системы. Вы обнаруживаете, в чем разница, и далее выполняете последовательность команд:

```
$ cvs commit -m "Добавлены новые циклы" driver.c
Checking in driver.c;
/usr/local/cvsroot/tc/driver.c,v  <--  driver.c
new revision: 1.2; previous revision: 1.1
done
$ cd ..
$ cvs release -d tc
? tc
You have [0] altered files in this repository.
Are you sure you want to release (and delete) module
`Tc': y
```

После этого вы можете спокойно удалить ваш рабочий каталог.

Хранилище CVS

Хранилище CVS предназначено для хранения всех исходных текстов (файлов и каталогов), смена версий которых должна быть под контролем.

Обычно вы никогда не имеете дело напрямую с файлами, содержащимися в хранилище. Предполагается, что вы с помощью средств CVS создаете рабочую копию одного или группы файлов в вашем рабочем каталоге. Следовательно, напрямую вы имеете дело только с рабочей копией. Когда вы изменили рабочую копию, то она должна быть помещена снова в хранилище, что также выполняется средствами CVS. Заметим попутно, что хранилище и ваш рабочий каталог — разные вещи, лучше их не путать и держать в разных местах, например, в разных каталогах.

Поскольку хранилище может находиться как на той же машине, где работают разработчики, так и на другом компьютере, то имеется несколько методов доступа к хранилищу CVS: локальный и удаленный методы доступа. Если метод доступа никак не обозначен, то предполагается, что хранилище находится на том же компьютере. Хранилище состоит из двух частей: одна часть — это собственно исходные тексты пользователя, другая — административная информация самой системы CVS.

Создание хранилища CVS

Выберите подходящий диск и каталог, учитывая возможный объем, который потребуется в будущем. Затем выполните команду создания хранилища CVS:

```
cvs init -d ~/ProJ init
```

По этой команде в вашем главном каталоге будет создан подкаталог с именем `ProJ`. Если вы посмотрите внутрь каталога `ProJ`, то увидите созданный административный подкаталог системы CVS с именем `ProJ/CVSROOT`. Если вы случайно укажете уже существующее хранилище в команде `init`, то команда не испортит существующего хранилища.

Параметр `-d` указывает, что имя каталога, в котором будет организовано хранилище, находится в командной строке в

качестве аргумента для данного параметра. Имя должно быть абсолютным именем. Имя каталога, в котором находится хранилище, можно указать в переменной окружения `$CVSROOT`. Таким образом, предыдущую команду создания можно было написать

```
cvs init
```

если переменная `$CVSROOT` уже содержит абсолютное имя каталога, где планируется разместить хранилище.

Форма представления данных в хранилище CVS

В ряде случаев полезно иметь некоторые представления об организации хранилища, чтобы понимать, например, какие права доступа надо устанавливать для файлов хранилища или как ограничивать доступ к файлам.

Какие файлы содержатся в хранилище

В целом, структура хранилища представляет собой иерархию (дерево) каталогов, соответствующую структуре рабочего каталога. Предположим, что хранилище находится в каталоге `/usr/local/cvsroot`. На рис. 5.2 приведено возможное дерево каталогов:

В каталогах имеются *исторические файлы*, по одному на каждый файл, который находится под контролем CVS. Имя исторического файла представляет собой имя файла под контролем CVS с окончанием `,v`. Таким образом, каталог `tc` выглядит примерно как показано на рис. 5.3.

Исторические файлы содержат достаточно информации, чтобы на их основе воссоздать любую версию файла, протокол всех комментариев команды `commit` и имя пользователя, который выполнил команду `commit`. Исторические файлы еще называют файлами RCS, поскольку RCS (система управления версиями) была первой системой, которая запоминала файлы в таком формате. Полное описание формата может быть найдено с помощью команды `man`, следует

смотреть страницу `rcsfile(5)`. Этот формат файлов стал весьма общим — много различных систем, а не только CVS или RCS могут, как минимум, импортировать исторические файлы в таком формате.

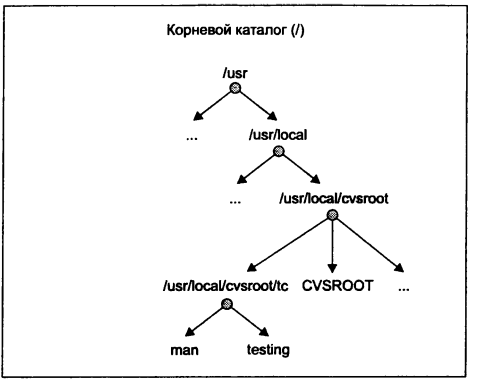


Рис. 5.2. Возможное дерево каталогов

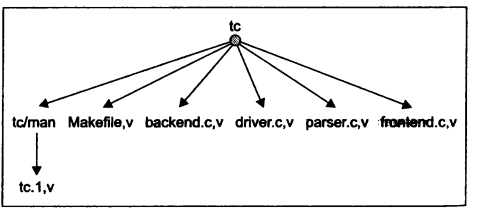


Рис. 5.3. Каталог tc

Тем не менее, исторические файлы CVS несколько отличаются от файлов RCS. Здесь мы не станем рассматривать этот вопрос подробнее.

Удаленные хранилища

Рабочие копии программ вы можете хранить в рабочем каталоге на вашей личной машине, а само хранилище может располагаться на другой машине в другом конце офиса, в другом здании или на другом континенте. Использование удаленного хранилища в целом происходит таким же порядком, как и использование локального хранилища. Исключение составляет лишь формат имени хранилища, который в удаленном случае имеет вид:

```
:METHOD:USER@HOSTNAME:/path/to/repository
```

Детали формата зависят от метода доступа, с помощью которого вы обращаетесь к удаленному хранилищу.

Метод доступа rsh

Система CVS может использовать протокол rsh (имеется в виду Remote SHell – удаленная оболочка), чтобы выполнить необходимые операции в удаленном хранилище. Таким образом, на компьютере (сервере), где находится хранилище, должен быть соответствующим образом заполнен файл с именем `.rhosts`.

Предположим, что вы зарегистрированы на вашей машине local.pnpi.spb.ru как пользователь с именем mozart. А ваш сервер находится на машине rsgi01.rhic.bnl.gov. При этом все тексты хранятся под именем пользователя bach. Тогда вам необходимо в файл rsgi01.rhic.bnl.gov:~bach/.rhosts добавить строку:

```
local.pnpi.spb.ru mozart
```

После этого на своей локальной машине local.pnpi.spb.ru проверьте, что все работает нормально:

```
rsh -l bach rsgi01.rhic.bnl.gov 'echo $PATH'
```

Вы должны увидеть содержимое переменной окружения $PATH на машине rsgi01.rhic.bnl.gov. Обратите внимание, что переменная должна указывать на имя каталога, в котором находится программный демон CVS. Вы можете указать другим способом, откуда вызывать демона системы CVS. Для этого вам следует установить на локальной машине

local.pnpi.spb.ru переменную окружения $CVS_SERVER с именем демона (программы), который вы хотите использовать, например, /usr/local/bin/cvs-2000.

Если используется протокол rsh, то нет необходимости устанавливать другие системные файлы, чтобы запускать демон сервера системы CVS.

Имеется два метода доступа, которые вы можете указать в переменной окружения $CVSROOT при использовании протокола rsh. Используя метод :pserver:, вы определяете внутреннего rsh-клиента, который поддерживается только некоторыми портами. А метод :ext: определяет внешнюю программу rsh. По умолчанию в команде :ext: используется программа rsh, однако вы можете задать имя программы в переменной окружения $CVS_RSH. Нетрудно догадаться, что если вы использовали другое имя, например, ssh, то вам следует использовать другие имена файлов — .shosts вместо rhosts.

Продолжая наш прежний пример, предположим, что вы хотите получить доступ к файлу foo в хранилище /usr/local/cvsroot/ на машине rsgi01.rhic.bnl.gov. Тогда, чтобы создать рабочую копию файла на вашей локальной машине local.pnpi.spb.ru, вам следует написать:

```
cvs -d :ext:bach@rsgi01.rhic.bnl.gov:/usr/local/cvsroot
checkout foo
```

При этом часть bach@ может быть опущена, если имя пользователя на обеих машинах совпадает.

Установки на серверной стороне

На серверной стороне необходимо отредактировать системный конфигурационный файл /etc/inetd.conf таким образом, что программа inetd будет *знать*, что надо выполнить команду

```
cvs pserver
```

когда сервер обнаружит попытку соединения на соответствующий порт. По умолчанию номер порта равен 2401. Номер порта может быть другим, если ваш клиент был скомпилирован с установленной переменной $CVS_AUTH_PORT, которая указывала другой порт.

Если ваша версия программы inetd допускает в файле `/etc/inetd.conf` обычные номера портов, то следующих двух строк достаточно:

```
2401  stream  tcp  nowait  root  /usr/local/bin/cvs
cvs -b /usr/local/bin pserver
```

Параметр `-b` определяет каталог, в котором содержатся готовые к исполнению программы RCS на вашем сервере. Можно также использовать параметр `-T`, чтобы определить каталог для временных файлов.

Если inetd предпочитает символические имена сервисов вместо простых номеров портов, то вы можете поместить в файл `/etc/services` строку

```
cvspserver      2401/tcp
```

Тогда вы сможете использовать символическое имя сервиса `cvspserver` в файле `/etc/inetd.conf` вместо номера порта.

Естественно, что после изменения файлов вам необходимо заставить inetd заново прочесть конфигурационные файлы или перезапустить программу inetd.

Поскольку клиент содержит и передает пароли так же, как обычный текст, то может быть использован отдельный парольный файл CVS. Этот файл находится в `$CVSROOT/CVSROOT/passwd`. Его формат такой же, как `/etc/passwd` за исключением того, что он имеет всего два поля: имя пользователя и пароль. Например,

```
shevel:ULtgRLXo7NRxs
oreshkin:1sOp854gDF3DY
```

Пароль закодирован в соответствии со стандартом функции `crypt()`. Следовательно, вы можете скопировать поле пароля из файла `/etc/passwd`.

Когда проверяется пароль, то CVS сначала проверяет, есть ли пользователь в файле `$CVSROOT/CVSROOT/passwd`. Если пользователь найден, то сверяется пароль. Если пользователя нет в этом файле или сам файл `$CVSROOT/CVSROOT/passwd` отсутствует, то сервер CVS пытается найти пользователя среди зарегистрированных пользователей сервера.

В то время, когда CVS выполняет процедуру аутентификации, она работает с теми же привилегиями, что и обычный пользователь с тем же именем.

Пока единственным способом записать пароль является копирование его из какого-то другого места. Быть может, скоро появится команда `cvs passwd`.

Установки на стороне клиента

До того как соединиться с сервером, клиент должен подключиться с помощью команды:

```
cvs login
```

Подключение заключается в проверке пароля на стороне сервера, а также в записывании пароля для более позднего взаимодействия с сервером CVS, для последующих транзакций. Команда `cvs login` должна получить имя пользователя, имя сервера и абсолютное имя каталога, где находится хранилище CVS. Она берет эту информацию из переменной окружения `$CVSROOT`. Команда `cvs login` запросит интерактивно пароль пользователя. Например,

```
cvs -d :pserver:bach@rsgi01.rhic.bnl.gov:/usr/local/
cvsroot login
CVS password:
```

Если в проверке пароля успеха не достигнуто — будет диагностика, что пароль неверен. А если проверка будет пройдена, то вы подключены и можете выполнять другие команды:

```
cvs -d :pserver:bach@rsgi01.rhic.bnl.gov:/usr/local/
cvsroot checkout foo
```

В обоих предыдущих командах используется метод `:pserver:`. Если бы он не был указан, то система CVS предполагала бы, что надо использовать rsh.

После того как файл из предыдущего примера `foo` переписан в ваш рабочий каталог, то при выполнении команд CVS в этом каталоге нет необходимости каждый раз указывать в явном виде, где расположено хранилище, поскольку оно было записано в вашем рабочем каталоге (подкаталог с именем `CVS`).

По умолчанию пароль записывается в файл `$HOME/.cvspass` в специальном формате. Вы можете сменить умолчание

посредством установки переменной окружения `$CVS_PASSFILE`. Если вы планируете использовать данную переменную, то установите ее новое значение *ДО* выполнения подключения к удаленному серверу.

Другая переменная окружения — `$CVS_PASSWORD` сменит *все* запомненные пароли. Если она установлена, то CVS будет использовать это значение для всех процедур аутентификации при соединении с любыми серверами.

Несколько хранилищ

В некоторых ситуациях полезно иметь более чем одно хранилище. Особенно это полезно, если вы имеете несколько групп разработчиков и/или несколько проектов, которые не имеют общих программ или частей программ. Если вы имеете несколько хранилищ, то, чтобы использовать нужное хранилище, вы должны задать его имя в переменной окружения `$CVSROOT` или указать в качестве аргумента в параметре `-d` при вызове программы CVS.

Большое преимущество использования нескольких хранилищ состоит в том, что хранилища могут находиться на разных серверах. Но, с другой стороны, вы не можете с помощью одной команды системы CVS обработать сразу несколько хранилищ (репозитариев).

Если же вы планируете вести несколько проектов одновременно и планируете иметь несколько логически раздельных хранилищ, то технически проще завести одно хранилище с несколькими деревьями каталогов, каждое из которых соответствовало бы отдельному проекту.

Версии файлов и программных продуктов

Для большинства случаев применения CVS нет необходимости уделять много внимания номерам версий исходных текстов: CVS автоматически назначает номера 1.1, 1.2, 1.3 и т. д. Тем не менее, некоторые разработчики хотят знать больше о номерах версий.

Если кто-то захочет иметь список версий для более чем одного файла, то это проще сделать с использованием понятия *тег* (*tag*), который представляет собой символьное имя версии. Оно может быть присвоено числовому значению версии в каждом файле.

Номера версий

Для каждой версии файла имеется уникальный *номер версии* (revision number). Номер версии выглядит как 1.1, 1.2, 1.3.2.2 или даже 1.3.2.2.4.5. Номер версии всегда имеет четное число целых чисел, разделенных точками. По умолчанию версия 1.1 является первой (или исходной) версией файла. Каждая последующая версия добавляет единицу в самую правую позицию номера версии. На рис. 5.4 показана простая структура последовательных версий.

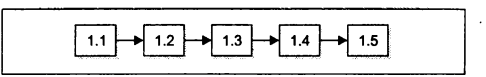


Рис. 5.4. Последовательные версии

Имеются номера версий, содержащих более чем одну точку, например как 1.3.2.2. Такие номера версий представляют версии *ветвей*, которые обсуждаются в разделе «Ветвление и объединение».

Присваивание версий

Когда вы добавляете новый файл, вторая цифра версии будет всегда 1, а первое число будет равно наибольшему первому числу в версии любого файла в данном каталоге. Например, если текущий каталог содержит файлы, чьи наиболее свежие версии имеют номера: 1.7, 3.1 и 4.12, тогда вновь добавляемому файлу будет присвоен номер версии 4.1.

Обычно нет нужды специально беспокоиться о номерах версий, система CVS сама наблюдает за ними и модифицирует, когда необходимо. Проще их воспринимать как некото-

рые внутренние переменные CVS. Теги (символьные имена версий) дают более удобный механизм для того, чтобы отделить, например, версию 1 от версии 2 вашего продукта. Однако если вы захотите самостоятельно изменить номера версий в хранилище, то это можно сделать командой:

```
cvs commit -r 3.0
```

Здесь параметр `-r` включает в себя параметр `-f` в том смысле, что все файлы будут отмечены в хранилище как обновленные независимо от того, имело ли место реальное изменение файлов. При этом последние версии всех файлов получат номер версии 3.0, если до выполнения данной команды не было номера больше, чем 3.0. Если же один из файлов уже имеет версию 3.0, то команда не будет выполнена, так как нельзя присвоить более ранний номер версии.

Если вы желаете поддерживать несколько параллельных версий вашего продукта (скажем, 1.x и 3.x), то следует использовать механизм *ветвления*.

Теги

Номера версий файлов живут своей собственной жизнью. Совсем не обязательно, чтобы они как-то были связаны с версиями вашего продукта.

В зависимости от того, как вы используете CVS, версии отдельных файлов могут измениться несколько раз между двумя последовательными версиями вашего продукта. В качестве примера мы можем рассмотреть комплект исходных текстов системы RCS 5.6, которые имеют следующие номера версий:

```
ci.c            5.21
co.c            5.9
ident.c         5.3
rcs.c           5.12
rcsbase.h       5.11
rcsdiff.c       5.10
rcsedit.c       5.11
rcsfcmp.c       5.9
rcsgen.c        5.10
```

```
rcslex.c        5.11
rcsmap.c        5.2
rcsutil.c       5.10
```

Вы можете использовать команду `tag`, чтобы дать символическое имя (тег) определенной версии отдельного файла. Вы можете также использовать команду `status` с параметром `-v`, чтобы увидеть все теги, которые имеет данный файл, а также номер версии, которую обозначает данный тег.

Имена тегов должны начинаться с буквы в любом регистре и могут содержать любые буквы, цифры и только два специальных знака: - (минус) и _ (подчеркивание). Два тега с именами `BASE` и `HEAD` зарезервированы за системой CVS. При выполнении большого проекта полезно придерживаться определенных соглашений об используемых именах тегов. Например, если ваша система имеет имя `TERCIYA`, то имя тега `TERCIYA-3_0` может представлять версию 3.0. Вы можете также использовать файл `taginfo`, чтобы описать соглашение об именах тегов.

Следующий пример показывает, как вы можете добавить тег к файлу. Команды добавления тега должны быть выданы внутри рабочего каталога, то есть там, где находится рабочая копия вашего файла (модуля).

```
$ cvs tag rel-0-4 backend.c
T backend.c
$ cvs status -v backend.c
===================================================================
File: backend.c  Status: Up-to-date

 Version:        1.4    Tue Dec  1 14:39:01 1992
 RCS Version:    1.4    /u/cvsroot/yoyodyne/tc/backend.c,v
 Sticky Tag:     (none)
 Sticky Date:    (none)
 Sticky Options: (none)

Existing Tags:
 rel-0-4                  (revision: 1.4)
```

Здесь приведен нечасто встречающийся пример, когда тегом помечается единственный файл (модуль). Обычно необходимо пометить определенным тегом все файлы, кото-

рые входят в продукт или составляют вместе определенный программный модуль. Это выполняется в те моменты, которые важны для всего проекта, например, при выпуске очередной версии программного продукта.

```
$ cvs tag rel-1-0 .
cvs tag: Tagging .
T Makefile
T backend.c
T driver.c
T frontend.c
T parser.c
```

В примере тегом помечаются все модули рабочего каталога (заметим попутно, что и все подкаталоги, если они есть). Если впоследствии вы захотите взять из хранилища лишь те файлы, которые относятся к версии продукта `rel-1-0`, то это выполняется командой

```
cvs checkout -r rel-1-0 tc
```

что, естественно, чрезвычайно удобно.

Когда тегом отмечаются несколько файлов, то это можно изобразить графически двояким образом. Оба варианта представлены на рис. 5.5.

Здесь видно, что одни файлы меняются со временем чаще, другие — реже. В какой-то момент времени мы отметили их тегом. Таким образом, мы сможем вызвать эти версии файлов в любой момент, когда нам это понадобится.

Липкий тег

Иногда рабочая копия комплекта исходных текстов имеет дополнительную информацию, например, такое может быть, когда вы работаете с ветвью или рабочая копия ограничена версиями определенной даты командами `checkout -D` или `update -D`. Поскольку такие данные сохраняются, то есть воздействуют на выполнение последующих команд, то будем их называть *липкие*.

В большинстве случаев *липкость* представляет собой скрытое свойство CVS, поэтому вам не требуется его изучать. Однако полезно знать о том, что липкие теги есть, хотя бы для избежания побочных эффектов.

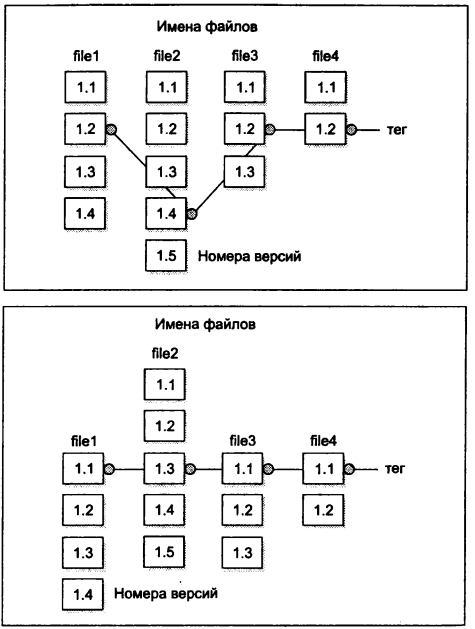


Рис. 5.5. Отметка тегом

Вы можете использовать команду `status`, чтобы увидеть, существуют ли липкие теги или установленные даты:

```
cvs status driver.c
 ===================================================
 File: driver.c    Status: Up-to-date

   Version:        1.7.2.1 Sat Dec  5 19:35:03 1992
   RCS Version:    1.7.2.1 /u/cvsroot/yoyodyne/tc/driver.c,v
   Sticky Tag:     rel-1-0-patches (branch: 1.7.2)
```

```
  Sticky Date:    (none)
  Sticky Options:(none)
```

Липкие теги останутся в ваших рабочих файлах до того момента, пока вы не удалите их с помощью команды

```
cvs update -A
```

Параметр `-A` разыскивает версию файла в голове ствола и снимает все липкие теги, даты или параметры.

Наиболее общеупотребительное использование липких тегов состоит в идентификации того, какая ветвь является рабочей в данный момент. Однако липкие теги используются и вне контекста ветвей. Например, вы хотите избежать обновления (команда `update`) вашего рабочего каталога, чтобы изолировать себя на время от дестабилизирующих частых изменений со стороны других разработчиков. Конечно, вы можете просто воздержаться от выполнения команды `update`. Однако вы хотели бы не обновлять лишь группу файлов из большого дерева каталогов. В этом случае может помочь липкий тег. Если вы выполните команду `cvs checkout` для определенной версии, например, 1.4, рабочая копия файлов станет липкой. Последующие команды `cvs update` не будут разыскивать последние версии файлов в хранилище для файлов, которые находятся в вашем рабочем каталоге до того момента, пока вы не выполните `cvs update -A`, очистив липкий тег.

Таким же образом, задание параметра `-D` в командах `checkout` и `update` устанавливает *липкую дату*, которая подобным же образом будет использоваться в последующих командах CVS.

Если вы захотите найти старую версию файла без установки липкого тега, используйте параметр `-p` в командах `checkout` или `update`, который направляет содержание файла на стандартное устройство вывода. Например, у вас есть файл с именем `file1` версии 1.1, и вы его удалили из хранилища. Теперь вы хотите добавить файл к хранилищу снова с тем же содержимым, которое было ранее. Ниже приведен способ, как это сделать:

```
$ cvs update -p -r 1.1 file1 >file1

===================================================================
Checking out file1
```

```
RCS:  /tmp/cvs-sanity/cvsroot/first-dir/Attic/file1,v
VERS: 1.1
***************
$ cvs add file1
cvs add: re-adding file file1 (in place of dead revision
1.2)
cvs add: use 'cvs commit' to add this file permanently
$ cvs commit -m test
Checking in file1;
/tmp/cvs-sanity/cvsroot/first-dir/file1,v  <--  file1
new revision: 1.3; previous revision: 1.2
done
$
```

Ветвление и объединение

CVS позволяет вам выделить какие-то изменения проекта в отдельную линию разработки, которую называют *ветвь* (branch). Когда вы меняете файлы в одной из ветвей, то эти изменения не появятся в основном стволе или в других ветвях.

Позднее вы сможет переместить ваши изменения из одной ветви в другую или в основной ствол посредством *объединения* (merging). Объединение включает прежде всего выполнение команды `cvs update -j`, чтобы объединить изменения в рабочем каталоге. После этого вы можете утвердить (`commit`) эту версию, тем самым реально скопировав изменения в другую ветвь разработки.

Для чего удобны ветви?

Предположим, что версия вашего продукта tc 1.0 вышла и начала использоваться. Вы продолжаете разрабатывать программу tc, очередная версия 1.1 которой выйдет в ближайшие несколько месяцев. Однако в это время ваши заказчики или партнеры стали жаловаться на фатальные ошибки в версии 1.0. Вы начали искать и быстро обнаружили ошибку, которую немедленно следует устранить. Однако наиболее свежие версии файлов представляют весьма нестабильное состояние и станут стабильными лишь через месяц или более. Таким образом, вы не можете исправить ошибку в наиболее свежих версиях и предоставить их заказчикам.

В такой ситуации удобнее всего создать отдельную ветвь для всех файлов, которые составляют ваш продукт tc версии 1.0. Там вы сможете менять файлы, исправляя текущие ошибки в программах. Эти исправления никак не будут влиять на основной ствол разработки. Когда срочные исправления внесены и заказчики удовлетворены, вы можете выбрать стиль поведения: оставить ваши исправления в ветви или поместить их в основной ствол разработки.

Создание ветви

Вы можете создать ветвь командой `tag -b`; например, предполагая, что вы в рабочем каталоге:

```
cvs tag -b rel-1-0-patches
```

Эта команда отделяет ветвь с именем `rel-1-0-patches`, которая базируется на текущих версиях файлов в рабочем каталоге.

Важно понять, что ветви создаются в хранилище, а не в рабочем каталоге. Создание ветви, базирующейся на текущих версиях файлов, как показано в предыдущем примере, не означает, что рабочая копия комплекта файлов в рабочем каталоге немедленно начнет рассматриваться как новая ветвь. Как это сделать, мы узнаем чуть позже.

Новую ветвь можно создать, не используя рабочей копии, посредством команды `rtag`:

```
cvs rtag -b -r rel-1-0 rel-1-0-patches tc
```

Здесь параметр `-r rel-1-0` говорит, что новая ветвь должна корениться (отрастать) в версии, которая соответствует тегу `rel-1-0`. Совсем не обязательно, что это будет наиболее свежая версия. Часто удобно выделить ветвь в старой версии продукта (той версии, где была исправлена ошибка).

Как и с командой tag, параметр `-b` создает новую ветвь и с командой `rtag`. Заметим, однако, что числовое значение номера версии, который удовлетворяет `rel-1-0`, может быть различным для каждого файла.

Таким образом, полный эффект команды состоит в создании новой ветви с именем `rel-1-0-patches` в модуле `tc`, основывающейся на комплекте файлов, входящих в версию `rel-1-0` модуля (продукта) tc.

Доступ к ветвям

Вы можете получить копию ветви одним из двух способов: прочесть ветвь из хранилища с помощью команды `checkout` или переключить существующую рабочую копию на требуемую ветвь проекта.

Чтение конкретной ветви из хранилища с помощью команды `checkout` выполняется следующим образом:

```
cvs checkout -r rel-1-0-patches tc
```

Здесь за параметром `-r` следует имя тега с именем `rel-1-0-patches`, который описывает ветвь.

Переключить рабочий каталог, в котором уже имеется рабочая копия, на требуемую ветвь можно следующим образом:

```
cvs update -r rel-1-0-patches tc
```

Несущественно, если содержимое рабочего каталога соответствовало основному стволу или другой ветви. По команде `update -r` будет выполнено слияние (объединение) тех изменений, которые вы уже сделали в рабочем каталоге, и содержимое ветви в хранилище. Если при слиянии возникнут конфликты, то вы получите соответствующую диагностику.

Раз содержимое вашего рабочего каталога соответствует конкретной ветви, то это соответствие таким и останется, пока вы не скажете явно что-то другое. Это значит, что любые подтверждения (`commit`) изменений в файлах будут проявляться лишь в конкретной ветви. В то же время никаких изменений не будет происходить ни в других ветвях, ни в основном стволе.

Чтобы определить, какая ветвь находится в данный момент в рабочем каталоге, можно использовать команду `status`:

```
cvs status -v driver.c backend.c
===================================================================
File: driver.c              Status: Up-to-date

   Version:             1.7      Sat Dec  5 18:25:54 1992
   RCS Version:         1.7      /u/cvsroot/yoyodyne/tc/
                                 driver.c,v
   Sticky Tag:          rel-1-0-patches (branch: 1.7.2)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

```
  Existing Tags:
      rel-1-0-patches                  (branch: 1.7.2)
      rel-1-0                          (revision: 1.7)

===================================================================
File: backend.c             Status: Up-to-date

  Version:              1.4      Tue Dec  1 14:39:01 1992
  RCS Version:          1.4      /u/cvsroot/yoyodyne/tc/
                                 backend.c,v
  Sticky Tag:           rel-1-0-patches (branch: 1.4.2)
  Sticky Date:          (none)
  Sticky Options:       (none)

  Existing Tags:
      rel-1-0-patches                  (branch: 1.4.2)
      rel-1-0                          (revision: 1.4)
      rel-0-4                          (revision: 1.4)
```

В выводимой информации обратите внимание на липкий тег. Он содержит информацию о конкретной ветви, если таковая есть.

Не стоит смущаться тем фактом, что номера версий ветви для каждого файла различны (1.7.2 и 1.4.2 соответственно). Тег ветви является тем же самым (`rel-1-0-patches`) и, следовательно, эти файлы находятся в одной ветви. Номера 1.7.2 и 1.4.2 отражают лишь конкретное состояние соответствующих файлов в момент создания ветви. По этим числам из вышеприведенного примера можно лишь заключить, что файл `driver.c` менялся чаще, чем файл `backend.c` до момента создания данной ветви.

Ветви и номера версий

Как уже отмечалось, обычно история последовательных версий продукта имеют линейную структуру, как показано на рис. 5.4. Однако CVS не ограничивается лишь линейными представлениями процесса разработки. Основное дерево (ствол) версий может быть разбито на несколько независимых ветвей. Изменения, которые имеют место в конкретной ветви, могут быть позже перемещены как в основной ствол, так и в другую ветвь.

Каждая ветвь имеет номер, который состоит из нечетного числа десятичных целых, разделенных точками. Номер ветви составляется из номера версии программного продукта с добавлением десятичного целого. Наличие номера ветви позволяет иметь несколько ветвей от одной версии продукта.

Все версии в ветви составлены из номера ветви, за которым следует порядковый номер версии. Рисунок 5.6 поясняет нумерацию ветвей на примерах.

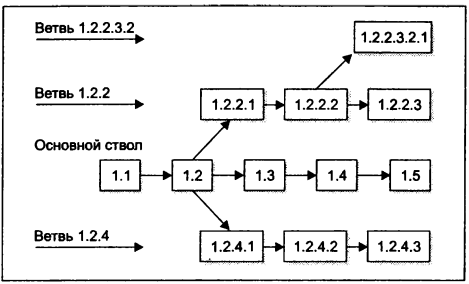


Рис. 5.6. Возможная структура ветвей

Обычно не нужно точно знать, как образуются новые номера ветвей, поэтому обрисуем их нумерацию лишь кратко. Когда CVS образует ветвь, она добавляет в конец номера версии четное число, начиная с 2. Таким образом, если вы захотите образовать ветвь от версии 7.6, то номер ветви будет равен 7.6.2, вторая ветвь от этой версии будет иметь номер 7.6.4 и т. д. Все ветви с номерами, оканчивающимися на 0 (нуль), используются самой системой CVS.

Магические номера ветвей

Этот раздел описывает возможность CVS, называемую *магическими ветвями*. Они не требуются пользователю при обычной работе, но полезно представлять, для чего они служат.

Номера ветвей состоят из нечетного числа десятичных целых, разделенных точками. Однако это не полная правда.

В целях повышения эффективности своей работы CVS иногда вставляет дополнительный ноль (0) во вторую справа позицию номера ветви (таким образом 1.2.4 становится 1.2.0.4, 8.9.10.11.12 становится 8.9.10.11.0.12, и т. д.).

CVS выполняет всю работу связанную с магическими номерами внутри, но иногда они могут появляться снаружи:

- Магический номер появляется в выводе команды `cvs log`.
- Вы не сможете определить символьное имя ветви в команде `cvs admin` у ветви с магическим номером.

Вы можете использовать команду `admin`, чтобы переприсвоить символьное имя ветви, как его ожидает увидеть RCS. Если имя `R4patches` присвоено ветви 1.4.2 (магический номер ветви 1.4.0.2) в файле `numbers.c`, то вы можете сделать:

```
cvs admin -NR4patches:1.4.2 numbers.c
```

Это будет работать, если только хотя бы одна версия была уже утверждена (committed) в ветви. Будьте внимательны, чтобы не назначить тег неверному номеру, поскольку на другой день не будет возможности увидеть, чему назначен тег.

Слияние ветвей

Вы можете объединить изменения в отдельной ветви и изменения в вашем рабочем каталоге, используя параметр `-j branch` в команде `update`. По этой команде CVS объединяет все изменения, сделанные в ветви от момента ее создания, и содержимое рабочего каталога. Очевидно, что `-j` означает объединение (join).

Рассмотрим дерево версий на рис. 5.7.

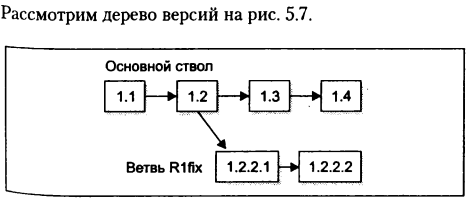


Рис. 5.7. Дерево версий

Ветви 1.2.2 присвоено символьное имя (тег) `R1fix`. Следующий пример предполагает, что модуль `mod` содержит один файл `m.c`.

```
$ cvs checkout mod      # Поместить в рабочий каталог
                        # последнюю версию, 1.4

$ cvs update -j R1fix m.c  # Объединить все изменения,
                           # сделанные 1.2 и 1.2.2.2,
                           # с изменениями, которые
                           # вы сделали в рабочем
                           # каталоге
$ cvs commit -m "Included R1fix" # Создать версию 1.5.
```

Во время объединения разных версий могут возникнуть конфликты. Если такое случится, то конфликты должны быть разрешены до утверждения изменений командой `commit`.

Повторное слияние

Продолжая наш пример, дерево версий теперь выглядит так, как показано на рис. 5.8.

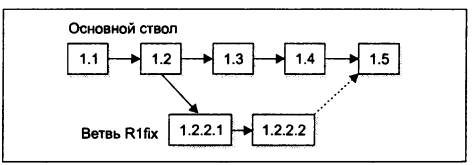


Рис. 5.8. Дерево версий (продолжение)

На рисунке 5.8 штрих-пунктирной линией обозначены слияния ветви `R1fix` с основным стволом, как мы обсуждали в прошлом разделе.

Теперь, предположим, разработка ветви `R1fix` продолжается, как показано на рис. 5.9.

Теперь, если вы захотите объединить эти новые изменения и основной ствол разработки, то просто наберите команду

```
cvs update -j R1fix m.c
```

снова. CVS вновь попробует объединить изменения, но это может также дать нежелательные побочные эффекты, поскольку объединение уже выполнялось ранее.

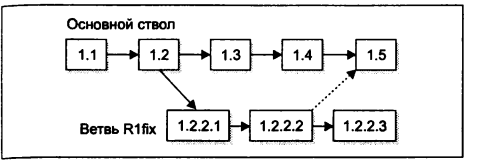


Рис. 5.9. Дерево версий (продолжение)

Чтобы избежать побочного эффекта, вам следует определить, что вы хотите объединить изменения в ветви, которая еще не была объединена с основным стволом. Чтобы так сделать, вы определяете два параметра `-j`, и тогда CVS объединит изменения первой версии и второй. Например, в данном случае могло бы быть так:

```
cvs update -j 1.2.2.2 -j R1fix m.c
```

Объединить изменения в рабочем каталоге с версией 1.2.2.2, а затем с ветвью `R1fix` (то есть с версией 1.2.2.3). Проблема здесь лишь в том, что версию 1.2.2.2 вам следует указать вручную. Чуть лучшее решение — с использованием даты, когда имело место последнее объединение:

```
cvs update -j R1fix:yesterday -j R1fix m.c
```

Еще чуть лучше, помечать тегом ветвь R1fix каждый раз, когда производится объединение в основном стволе, и позднее использовать этот тег в последующих объединениях:

```
cvs update -j merged_from_R1fix_to_trunk -j R1fix m.c
```

Объединение различий между двумя любыми версиями

С двумя параметрами `-j REVISION` команды `update` (и `checkout`) могут объединить отличия между любыми двумя версиями, записав результат в ваш рабочий файл. Так

```
cvs update -j 1.5 -j 1.3 backend.c
```

удалит все изменения, сделанные между версиями 1.3 и 1.5. Обратите внимание на порядок версий в командной строке!

Если вы попытаетесь использовать эту возможность сразу на множестве файлов, то помните, что числовые значения версий могут весьма отличаться для различных файлов, которые вместе составляют единый модуль для CVS. Обычно удобно использовать символьные теги, а не номера версий, когда имеете дело сразу со многими файлами.

Форма хранения данных в рабочем каталоге

Раз мы обсуждаем некоторые детали работы хранилища CVS, то полезно кратко описать, что помещает система CVS в ваш рабочий каталог в подкаталог с именем `CVS`. В некоторых случаях может оказаться полезным заглянуть внутрь подкаталога `CVS` в вашем рабочем каталоге. Напомним, что такой подкаталог появится, когда вы выполните команду

```
cvs checkout ProJ
```

Здесь `ProJ` есть имя каталога в хранилище CVS. По этой команде в вашем рабочем каталоге будет создан подкаталог с именем `ProJ`, в который будет скопировано содержимое каталога `ProJ` из хранилища CVS. Когда вы перейдете в каталог `ProJ`, то увидите подкаталог с именем `CVS`, который используется системой CVS.

Каталог `CVS` содержит несколько файлов, каждый из которых является просто текстовым файлом. Рассмотрим их чуть детальнее.

Таблица 5.1. Файлы каталога CVS

Имя файла	Назначение
Root	Этот файл содержит имя каталога, в котором располагается хранилище CVS, то же, что в переменной $CVSROOT
Repository	Этот файл содержит имя каталога в хранилище CVS, которое соответствует данному рабочему каталогу. Имя может быть как абсолютным, так и относительным по отношению к содержимому файла Root
Entries	Этот файл содержит списки файлов и каталогов в рабочем каталоге

Имя файла	Назначение
Entries.Log	Этот файл содержит практически то же самое, что и файл Entries. Он используется, чтобы обновление файла Entries не приводило к потере информации, даже если работа обновляющей программы завершилась аварийно
Entries.Backup	Временный файл
Entries.Static	Служебный файл CVS. Для CVS важно лишь, существует этот файл или нет
Tag	Этот файл содержит теги (описатели) или даты для подкаталогов
Checkin.prog Update.prog	Эти файлы хранят программы, специфицированные параметрами -i и -u в файлах модулей соответственно
Notify	Этот файл хранит сообщения, пока еще не посланные на сервер
Notify.tmp	Вспомогательный файл для поддержания файла Notify
Base	Если используются наблюдатели (watches), тогда команда edit запоминает исходную копию файла в каталоге Base. Это позволяет выполнить команду unedit, даже если нет возможности связаться с сервером
Baserev	Этот файл перечисляет версии всех файлов в каталоге Base
Baserev.tmp	Временный файл, служит для поддержания Base
Template	Этот файл содержит заготовку, определенную посредством файла rcsinfo. Файл используется только на клиентской стороне

Конфигурационный файл modules

Файл `modules` хранит ваши определения символьных имен (модулей) для группы исходных текстов, находящихся в хранилище. CVS будет также использовать эти определения, если вы вызовете ее для обновления самого файла `modules`.

Файл `modules` может содержать пустые строки и комментарии (строки, начинающиеся знаком # — решетка) и строки с определением файлов, которые входят в объект (модуль). Длинные определения можно продолжать на другой строке, указав обратный слэш в конце той строки, которую хотите продолжить.

Имеется три типа объектов, которые могут быть описаны в файле `modules`:

- *alias modules* — алиасные модули;
- *regular modules* — регулярные модули;
- *ampersand modules* — амперсандные модули.

Различие между ними состоит в способе отображения файлов хранилища в файлы рабочего каталога. Удобнее всего рассмотреть это на примерах. Во всех нижеследующих примерах мы полагаем, что хранилище содержит каталог `first-dir`, который содержит два файла: `file1` и `file2`, а также подкаталог `sdir`. Каталог `first-dir/sdir` содержит файл `sfile`.

Простейший вид объекта: алиасный модуль

Алиасный модуль — это простейший вид объекта в хранилище. Общий вид этого описания (строки) приведен ниже.

```
mname -a [aliases ...]
```

Это простейший способ определения модуля с именем `mname`. Параметр `-a` означает, что CVS будет воспринимать `mname`, если встретит его в качестве аргумента команды, как список имен файлов `aliases ...`. Список `aliases ...` может содержать другие имена модулей или пути.

Когда вы используете имя пути в качестве элемента `aliases ...`, то команда `checkout` создает промежуточный каталог в вашем рабочем каталоге, как будто вы использовали имя пути явно в командной строке `checkout`.

Например, если файл `modules` содержит строку:

```
amodule -a first-dir
```

тогда следующие две команды просто эквивалентны по своим результатам:

```
cvs checkout amodule
cvs checkout first-dir
```

Они произведут вывод, похожий на тот, что приведен ниже:

```
cvs checkout: Updating first-dir
U first-dir/file1
U first-dir/file2
cvs checkout: Updating first-dir/sdir
U first-dir/sdir/sfile
```

Регулярные модули

Общий вид строки файла `modules`, описывающей регулярные модули, представлен ниже:

```
mname [options] dir [files...]
```

В простейшем случае эта форма записи редуцируется до

```
mname dir
```

Эта строка определяет все файлы в каталоге `dir` как модуль с именем `mname`. Имя `dir` является относительным к `$CVSROOT`. Во время выполнения команды `checkout` одиночный каталог с именем `mname` будет создан как рабочий каталог. Не будет использовано никаких промежуточных каталогов, даже если `dir` включает несколько промежуточных уровней.

Например, если модуль определен как:

```
regmodule first-dir
```

то `regmodule` будет содержать файлы из каталога `first-dir`. Тогда выполнение показанной ниже команды даст следующее

```
cvs checkout regmodule
cvs checkout: Updating regmodule
U regmodule/file1
U regmodule/file2
cvs checkout: Updating regmodule/sdir
U regmodule/sdir/sfile
```

Явно описывая файлы в определении модуля после `dir`, вы можете выбрать отдельные файлы из каталога `dir`. Например,

```
regfiles first-dir/sdir sfile
```

С таким определением получение модуля с именем `regfiles` создаст одиночный рабочий каталог с именем `regfiles`, содержащий перечисленные файлы, которые будут взяты из ниже лежащего каталога хранилища CVS.

```
$ cvs checkout regfiles
U regfiles/sfile
```

Амперсандные модули

Модульное определение может ссылаться на другие модули включением строки `&module` в себя:

```
mname [options] &module ...
```

Тогда получение модуля создает подкаталог для каждого такого модуля в каталоге, содержащем модуль. Например, если модули содержат

```
ampermod &first-dir
```

тогда команда `checkout` создаст каталог с именем `ampermod`, который содержит подкаталог с именем `first-dir`, который, в свою очередь, содержит все каталоги и файлы, живущие там. Например, команда

```
$ cvs checkout ampermod
```

создаст следующие файлы:

```
ampermod/first-dir/file1
ampermod/first-dir/file2
ampermod/first-dir/sdir/sfile
```

Имеется небольшая погрешность в диагностике: сообщения, которые выводит CVS, не содержат слова `ampermod`, то есть положение создаваемых файлов указывается не совсем корректно:

```
$ cvs co ampermod
cvs checkout: Updating first-dir
U first-dir/file1
U first-dir/file2
cvs checkout: Updating first-dir/sdir
U first-dir/sdir/sfile
$
```

В будущем это, видимо, будет изменено.

Исключение каталогов

Алиасный модуль может исключать отдельные каталоги посредством указания восклицательного знака перед их именами.

Например, если файл `modules` содержит

```
exmodule -a !first-dir/sdir first-dir .
```

тогда по команде `checkout` для модуля `exmodule` будут созданы рабочие копии всех файлов и каталогов из каталога `first-dir`, исключая файлы из каталога `first-dir/sdir`.

Модульные параметры

Регулярные модули и амперсандные модули могут содержать параметры, которые обеспечивают дополнительную информацию, касающуюся модуля.

Таблица 5.2. Параметры модулей

Параметр	Назначение
-d name	Дать рабочему каталогу имя name, а не использовать для этого имя модуля
-e prog	Определить программу prog, которая будет вызвана с именем модуля в качестве параметра в момент экспорта модуля
-i prog	Определить программу prog, которая будет вызвана в момент выполнения команды commit. Единственным параметром будет абсолютное имя каталога в хранилище. Файлы commitinfo, loginfo, verifymsg обеспечивают другие способы для вызова какой-то программы в момент выполнения commit
-o prog	Определить программу prog, которая будет вызвана в момент выполнения команды checkout. Единственный параметр команды – имя модуля
-s status	Присвоить статус модулю. Когда модульный файл печатается по команде cvs checkout -s, модули отсортированы в соответствии с основными состояниями модуля и в соответствии с именами модулей. Этот параметр использовать для того, чтобы перечислить лиц, ответственных за модуль
-t prog	Определить программу prog, которая будет вызвана в момент выполнения команды rtag. Программа prog выполняется с двумя аргументами: имя модуля и символьное имя, тег (tag), которое определено командой rtag. Программа prog не вызывается, когда исполняется команда tag. Во многих случаях файл taginfo дает лучшее решение
-u prog	Определить программу prog, которая будет вызвана в момент выполнения команды cvs update, начиная с верхнего каталога модуля, для которого выполняется checkout. Программа prog выполняется с единственным аргументом – абсолютное имя для модуля в хранилище

Файл установки фильтров

Фильтры позволяют вам автоматически сделать специальное преобразование файлов при внесении их в хранилище CVS, а также при копировании файлов из хранилища.

Файл с именем `cvswrappers` определяет сценарий, который будет выполняться над файлом, если его имя удовлетворя-

ет регулярному выражению. Если говорить более точно, существует два сценария. Один — тот, который выполняется над файлом/каталогом перед тем, как файл помещается в хранилище (параметр `-t`), а другой — тот, который вызывается, когда файл копируется из хранилища (параметр `-f`). Параметры `-t` и `-f` не действуют, если используется CVS типа клиент/сервер.

Сценарий `cvswrappers` имеет параметр `-m`, чтобы определить методологию слияния файлов, если только они не двоичные, когда файлы обновляются. Метод `MERGE`, указанный после параметра, означает обычное поведение CVS, то есть будет предпринята попытка слияния. А метод `COPY` означает, что `cvs update` не будет пытаться слить файлы, как она не делает слияния для двоичных файлов.

ВНИМАНИЕ Не стоит пробовать этот параметр для версий CVS ранее, чем 1.9.

Параметр `-m` в файле `cvswrappers` воздействует на поведение CVS лишь когда производится слияние по команде `update`. При этом нет никакой разницы, в каком виде файлы хранятся. Базовый формат файла `cvswrappers` следующий:

```
wildcard        [option value][option value]... ,
```

где параметрами могут быть:

- `-f` — фильтрация при копировании из CVS;
- `-t` — фильтрация при записи в CVS;
- `-m` — метод обновления (update);
- `-k` — подстановка ключевых слов.

А значениями параметров являются строки, ограниченные одиночными апострофами, например:

```
*.nib       -f 'unwrap \%s' -t 'wrap \%s \%s' -m 'COPY'
*.c             -t 'indent \%s \%s'
```

`wildcard` означает в примере шаблон, описывающий имена файлов. Вышеприведенный пример файла `cvswrappers` устанавливает, что все файлы и каталоги в хранилище, име-

на которых заканчиваются на `.nib`, должны фильтроваться программой `wrap` непосредственно перед помещением в хранилище. Такие файлы должны быть подвергнуты фильтрации программой `unwrap` после выдачи из хранилища. В примере устанавливается также, что будет использован метод `COPY` во время выполнения команды `update`, то есть не будут предприняты попытки слияния.

В последней строке примера устанавливается, что для файлов, имена которых оканчиваются на `.c`, будет предпринята фильтрация программой `indent` перед помещением в хранилище.

Фильтр `-t` вызывается с двумя аргументами: первый — имя файла/каталога, который должен фильтроваться, второй — путь, куда должен быть помещен результат фильтрации.

Фильтр `-f` вызывается с одним аргументом, именем файла/каталога, который подлежит фильтрации. Конечный результат фильтрации будет находиться в пользовательском каталоге.

Заметим, что параметры `-t` и `-f` не вполне эффективно справляются с частью операций CVS, например, с установлением факта модификации файлов. CVS может ошибочно воспринимать каталог не модифицированным, даже если какой-то файл внутри модифицирован, но вы сможете заставить CVS выполнить операцию `commit` корректно, использовав параметр `-f` в команде `cvs commit`.

Другой пример. Следующая команда импортирует каталог, обрабатывая файлы, имена которых заканчиваются на `.exe` как двоичные:

```
cvs import -I ! -W "*.exe -k 'b'" first-dir vendortag
reltag
```

Конфигурационные файлы для поддержки команды commit

Параметр `-i` в файле `modules` может быть использован, чтобы запустить определенную программу в момент обновления хранилища по команде `commit`. Конфигурационные файлы, рассматриваемые в данном разделе, обеспечивают

более гибкие способы для запуска программ, когда что-то обновляется в хранилище по команде `commit`.

Имеется три вида программ, которые могут быть запущены по команде `commit`. Эти программы определяются в конфигурационных файлах в хранилище в каталоге `$CVSROOT/CVSROOT` и описываются ниже. Нижеследующий список описывает имена конфигурационных файлов и назначение соответствующих программ.

- `commitinfo` — в этом файле может быть указана программа, которая ответственна за проверку того, что команда `commit` разрешена. Если программа завершится с ненулевым кодом завершения, то команда `commit` не будет дана.
- `verifymsg` — здесь может быть указана специальная программа, которая используется, чтобы обработать сообщение для протокола и, возможно, проверить, что сообщение содержит необходимые поля. Наиболее интересно использовать в комбинации с файлом `rcsinfo`, который может содержать шаблон для сообщения в протокол.
- `editinfo` — в файле может быть определена программа, которая вызывается, чтобы отредактировать сообщение для протокола.
- `loginfo` — может быть указана специальная программа, которая вызывается, когда `commit` завершена. Она воспринимает сообщение для протокола и некоторую дополнительную информацию, что позволяет записать все это в файл, послать сообщение конкретным лицам или сделать что угодно.

Далее мы рассмотрим только `commitinfo`.

Конфигурационные файлы `commitinfo`, `loginfo`, `rcsinfo`, `verifymsg` и другие имеют общий формат. Каждая строка содержит следующее:

- Регулярное выражение. Синтаксис регулярного выражения тот же, что используется в GNU Emacs.
- Разделитель полей — один или более пробелов и/или знаков табуляции `\t`.
- Шаблон имени файла или команды.

Пустые строки игнорируются. Строки, начинающиеся с символа # (решетка) рассматриваются как комментарий. Длинные строки *не могут* быть разбиты на две строки.

Используется первая часть регулярного выражения, которая соответствует имени текущего каталога в хранилище. Остаток строки используется как имя файла или как командная строка соответственно.

Файл commitinfo

Файл `commitinfo` определяет программы, которые будут вызваны, когда CVS готовится выполнить команду `commit`. Эти программы используются для проверки, что добавляемые, удаляемые или модифицируемые файлы действительно готовы для планируемых операций. Такая проверка могла бы заключаться в том, что внесенные изменения соответствуют стандартам, принятым в вашей организации.

Каждая строка файла `commitinfo` состоит из регулярного выражения и шаблона команды. Шаблон состоит из имени программы (сценария) и любого количества параметров. Полный путь к текущему хранилищу добавляется к шаблону, за которым следуют имена любых файлов, над которыми выполняется операция `commit`.

Будет использована первая строка файла `commitinfo`, в которой встретилось регулярное выражение, соответствующее имени каталога в хранилище. Если команда возвращает ненулевой код завершения, то операция прекращается.

Если имя хранилища не соответствует никакой строке файла `commitinfo`, то будет выполнена команда из строки `DEFAULT`, если таковая есть.

Все появления имени `ALL` в регулярных выражениях в файле `commitinfo` будут использованы после соответствующего регулярного выражения или после `DEFAULT`.

ПРИМЕЧАНИЕ Когда CVS использует удаленное хранилище, то будет использован файл commitinfo на серверной стороне, а не на стороне клиента.

Поддержка свежей рабочей копии

Часто оказывается очень желательным чтобы дерево рабочих каталогов оно содержало наиболее свежие версии файлов в хранилище. Например, вы хотите работать с наиболее свежими текстами программ, без необходимости постоянно следить за этим. Другой пример. Допустим, вы обслуживаете WWW-сервер и содержите страницы в хранилище CVS. Тогда было бы весьма удобно, чтобы любое изменение в хранилище сразу становилось доступно вашему WWW-серверу.

Хороший способ выполнять все то, о чем мы говорили, — это использовать в файле `loginfo` команду `cvs update`. Однако использовать команду `cvs update` прямо, как записано, будет затруднительно из-за того, что CVS блокирует доступ к хранилищу на время изменения файлов хранилища. Поэтому команда `cvs update` должна выполняться в фоновом режиме. Ниже следует пример, в котором все должно быть записано в одной строке:

```
^cyclic-pages (date; cat; (sleep 2; cd /www/docs/CSD;
cvs -q update -d) &) >> $CVSROOT/CVSROOT/updatelog 2>&1
```

Эта последовательность выполнится, когда будут изменяться файлы, имена которых начинаются с `cyclic-pages`, то есть будет обновлено дерево `/www/docs/CSD` (выполнена команда `update`).

Протоколирование операций CVS

Вы можете определить виды протоколирования и описать дополнительные действия, которые должны будут выполняться при различных командах CVS. Такие действия выполняются путем интерпретации сценариев в то или иное время. Сценарии могут добавлять какую-то стандартную информацию к протоколу, например, имя программиста, другую полезную информацию, посылать уведомления некоторому кругу лиц, а также производить другие полезные действия. Для того чтобы протоколировать команду `commit`, можно использовать файл `loginfo` и другие (см. раздел «Конфигурационные файлы для поддержки

команды commit»). Чтобы протоколировать выполнение команд `commit`, `checkout`, `export`, `tag`, необходимо использовать параметры `-i`, `-o`, `-e`, `-t` в файле `modules` соответственно. Более гибким способом уведомления нескольких лиц об изменениях в хранилище является вызов команды `cvs watch add`. Эта команда полезна, даже если вы не используете команду `cvs watch on` (см. следующий раздел).

Файл `taginfo` определяет программу, которая вызывается, когда кто-то выполняет команду `cvs tag` или `cvs rtag`. Файл `taginfo` имеет стандартную форму для административных файлов такого вида, где каждая строка представляет собой регулярное выражение и команду, которая выполнится, если регулярное выражение «сработало». Аргументы передаются команде в следующем порядке:

`TAGNAME`, `OPERATION`, `REPOSITORY` и все остальное передается парами: `FILENAME REVISION`. Ненулевой код завершения программы прекращает выполнение соответствующей команды, то есть тег не будет установлен. Поле `OPERATION` может иметь следующие значения:

- `add` для команды `tag`;
  `mov` для команды `tag -F`;
- `del` для команды `tag -d`.

Ниже приведен пример файла `taginfo`

```
ALL /usr/local/cvsroot/CVSROOT/loggit
```

где файл `/usr/local/cvsroot/CVSROOT/loggit` содержит следующее:

```
#!/bin/sh
echo "$@" >>/home/kingdon/cvsroot/CVSROOT/taglog
```

Кто редактирует файл

Для многих групп разработчиков использование CVS с параметрами по умолчанию вполне приемлемо. Ряд других групп предпочитают знать, кто и какие файлы редактирует. Здесь описываются средства CVS, которые позволяют разработчикам автоматически информировать друг друга о предпринимаемых действиях, что позволяет редактировать и отлаживать один и тот же файл двум и более лицам.

Максимальная польза для разработчиков состоит в использовании команд `cvs edit`, чтобы сделать файл доступным по чтению/записи на время его модификации, и `cvs release`, чтобы удалить рабочий каталог, который более не используется. Однако CVS не имеет средств, чтобы заставить всех поступать именно таким образом.

Включение/выключение режима слежения

Чтобы включить режим слежения за определенными файлами, используется команда:

```
cvs watch on [-lR] FILES ...
```

Эта команда определяет, что разработчики должны использовать команду `cvs edit` *до* начала редактирования файлов `FILES`. При этом CVS будет создавать рабочую копию файлов `FILES` с правами доступа *только чтение*, чтобы напомнить разработчикам, что *до* редактирования следует использовать команду `cvs edit`.

Если `FILES` содержит имя каталога, CVS будет использовать режим слежения для всех файлов этого каталога в хранилище. Режим слежения распространяется также на все вновь добавленные файлы. Содержимое каталогов обрабатывается рекурсивно, если явно не указано обратное параметром `-l`. Параметр `-R` снова включает рекурсивный режим обработки каталогов, если он был выключен, например, в конфигурационном файле `.cvsrc` в домашнем каталоге пользователя.

Если `FILES` опущено, то подразумевается текущий каталог.

Выключение режима слежения производится командой:

```
cvs watch off [-lR] FILES ...
```

Не производить никаких уведомлений о работе с файлами `FILES`, а рабочие копии создавать с правами доступа *чтение/запись*.

Значения `FILES` и других параметров имеют тот же смысл, что и в команде `cvs watch on`.

Способы уведомления о действиях CVS

Вы можете сообщить CVS, что вы хотели бы иметь уведомление о действиях, предпринятых над конкретным файлом.

Это можно реализовать без использования команды `cvs watch on` для данного файла, хотя в общем случае лучше использовать именно ее.

Формат команды:

```
cvs watch add [-a ACTION] [-lR] FILES ...
```

Назначение — добавить текущее имя пользователя к списку лиц, получающих уведомление о действиях над файлами `FILES`.

Параметр `-a` определяет действие, о котором вы будете получать уведомление. Значением `ACTION` может быть следующее:

- `edit` — другой пользователь использовал команду `cvs edit` по отношению к указанному файлу;
- `unedit` — другой пользователь применил команду `cvs unedit` к указанному файлу;
- `commit` — другой пользователь изменил указанный файл в хранилище (применил команду `commit`);
- `all` — другой пользователь применил к указанному файлу одну из перечисленных выше команд;
- `none` — ничего из вышеприведенного, то есть не надо уведомлять. (Полезно вместе с командой `cvs edit`).

Параметр `-a` может появиться в одной команде более чем один раз или не появиться совсем. Если параметр опущен, то подразумевается значение `all`.

Значение `FILES` и другие параметры имеют тот же смысл, что и в команде `cvs watch`.

Наконец, по команде

```
cvs watch remove [-a ACTION] [-lR] FILES ...
```

удаляются запросы на уведомление, которые были созданы командой

```
cvs watch add ...
```

Аргументы — те же самые. Если присутствует параметр `-a`, то удаляются только те действия, которые определены значением параметра.

Когда возникает условие для уведомления, то CVS вызывает файл `notify`. Этот файл редактируется так же, как и любые другие административные файлы. Файл следует обычным соглашениям для административных конфигурационных файлов — он состоит из строк, каждая из которых содержит регулярное выражение, за которым следует команда. Команда должна содержать одиночное вхождение `%s`, которое будет заменено именем пользователя, которого надо уведомить. Остаток информации касающейся уведомления будет передан команде на стандартное устройство ввода. Ниже следует пример строки файла `notify`:

```
ALL mail %s -s "CVS notification"
```

Эта строка означает, что пользователи будут уведомлены посредством электронной почты. CVS не уведомляет вас о ваших собственных изменениях.

Как редактировать наблюдаемый файл

Поскольку рабочая копия наблюдаемого файла во время выполнения команды `checkout` создается с правами доступа *только чтение*, то вы не можете немедленно начать редактировать такой файл. Чтобы сделать наблюдаемый файл доступным для редактирования и, одновременно, информировать об этом других разработчиков, вам следует использовать команду `cvs edit`.

Команда выглядит так:

```
cvs edit [OPTIONS] FILES ...
```

Приготовиться к редактированию файлов `FILES`. Это включает смену прав доступа к файлам `FILES` с *только чтение* на *чтение/запись* и уведомляет пользователей, которые запросили уведомление о редактировании любого файла из `FILES`.

`cvs edit` воспринимает те же параметры, что и команда `cvs watch add`, а также организует для пользователя, давшего команду `cvs edit`, временное слежение за файлами `FILES`. Временное слежение за файлами `FILES` будет прекращено, после выполнения команд `cvs unedit` или `cvs commit` по отношению к файлам `FILES`. Если пользователь не хочет получать никаких уведомлений, то он должен использовать параметр `-a none`.

FILES и другие параметры имеют тот же смысл, что и в команде cvs watch.

ВНИМАНИЕ Если параметр PreservePermissions включен в хранилище, то CVS не будет изменять права доступа ни к каким файлам из FILES.

Обычно, когда вы заканчиваете редактирование группы файлов, вы производите операцию cvs commit, которая вносит изменения в хранилище и возвращает права доступа наблюдаемых файлов в состояние *только чтение*. Однако если вы передумали и решили не делать никаких изменений или решили аннулировать уже сделанные изменения в рабочей копии, то вам следует воспользоваться командой cvs unedit. Команда выглядит так:

```
cvs unedit [-lR] FILES ...
```

Она аннулирует любые изменения в рабочей копии файлов FILES, если таковые имели место. CVS устанавливает права доступа в состояние *только чтение* для тех файлов из FILES, для которых были запросы на уведомление с использованием команды cvs watch on со стороны других пользователей. Кроме того, CVS уведомляет пользователей, которые ранее запрашивали уведомление об операциях cvs unedit для каких-то файлов из FILES.

FILES и другие параметры в команде имеют тот же смысл, что и в команде cvs watch.

Если слежение не было включено, то команда unedit может не выполняться. Тогда единственный способ вернуться к версии, которая имеется в хранилище, состоит в удалении файлов, а затем в выполнении команды cvs checkout для них. Значение такой последовательности действий не совсем точно совпадает с командой unedit, поскольку будут внесены также изменения, которые возможно имели место с момента предыдущей команды checkout или update.

Когда используется схема CVS клиент/сервер, то вы можете использовать cvs edit или cvs unedit даже, если CVS не смогла в данный момент успешно связаться с сервером. Уведомления будут посланы сразу после очередной успешной команды CVS.

Кто наблюдает и редактирует

Команда

```
cvs watchers [-lR] FILES ...
```

перечисляет всех пользователей, которые следят за изменениями в файлах FILES. Отчет будет содержать всех наблюдателей, их электронные адреса, имена файлов, за которыми они следят.

Команда

```
cvs editors [-lR] FILES ...
```

перечисляет пользователей, которые работают в данный момент над файлами из FILES. Отчет включает адреса каждого пользователя, время, когда пользователь начал работать над файлом, имя хоста, где пользователь работает, а также имя рабочего каталога, в котором пользователь работает с данным файлом.

FILES и другие параметры имеют тот же смысл, что и в команде `cvs watch`.

Создание файлов проекта в хранилище

Поскольку переименование файлов и каталогов является неудобной операцией, то прежде чем начать новый проект полезно установить некоторые соглашения об именах и об организации файлов в целом. Переименование не является полностью невозможным делом, однако, если изменяются имена десятков или сотен файлов и каталогов, такая операция увеличивает вероятность внесения ошибок. Может случиться так, что ошибки в именах обнаружатся (если такое произойдет) спустя значительное время после выполнения переименований.

Существующие файлы

Когда вы решили начать использование CVS, вы, возможно, уже ведете какие-то проекты и имеете группы файлов, которые вы хотели бы поместить в хранилище CVS. В таком случае удобнее всего использовать команду

```
cvs import
```

Предполагается, что установлена переменная окружения \$CVSROOT, указывающая на местоположение хранилища CVS. Если ваши файлы находятся, например, в каталоге WorkDIR, то вам удобнее сначала перейти в WorkDIR

```
cd WorkDIR
```

Предположим далее, что вам удобнее разместить ваш проект в каталоге хранилища под названием PHENIX. Тогда

```
cvs import -m "Imported sources" PHENIX MyProgs start
```

По этой команде содержимое каталога WorkDIR будет переписано в хранилище $CVSROOT/PHENIX. Несмотря на то что вы указали параметр -m, CVS вызовет редактор текста для ввода комментария. Строка MyProgs является специальным тегом, который должен быть здесь, а start — это освобождающий тег.

Далее можно проверить, что все «сработало» так, как ожидалось.

```
cd ..
mv WorkDIR WorkDIR_Initial
cvs checkout PHENIX
ls -R PHENIX
rm -r WorkDIR_Initial
```

Удаление исходного каталога является неплохой идеей, поскольку это будет гарантировать, что вы случайно не возьмете для работы файлы из каталога WorkDIR в обход системы CVS.

Команда checkout системы CVS может иметь в качестве аргумента как имя каталога в хранилище, так и имя отдельного модуля, но любое имя является относительным к $CVSROOT

Если некоторые файлы, которые вы хотели бы импортировать, являются двоичными, то, возможно, вы захотите использовать особенность wrappers, чтобы установить, какие файлы являются двоичными, а какие текстовыми.

Исходные тексты из разных компаний

Если вы модифицируете программу, учитывая особенности применения в вашей организации, то вы захотите, чтобы в следующих версиях данной программы ваши изменения так-

же были включены. Система CVS также может помочь вам в этом.

В терминологии, которая используется в CVS, поставщик программ называется *вендор* (vendor). Немодифицированная программа, полученная от вендора, помещается в хранилище в отдельную ветвь, которая именуется *вендорной ветвью* (vendor branch). CVS резервирует для этой цели номер ветви 1.1.1.

Когда вы модифицируете тексты программ и производите операцию `commit`, то ваша версия попадет в основной ствол (main trunk). Когда вендор приготовил новую версию программы, вы производите операцию `commit` в вендорной ветви и копируете модификации в основной ствол.

Для создания и обновления вендорной ветви используется команда `import`. Когда вы импортируете новый файл, вендорная ветвь становится *главной версией* (head revision). Иными словами, любой, кто выполняет операцию `checkout`, получит именно это версию, если специально не указано ничего иного. Когда изменения в рабочем каталоге вносятся в хранилище операцией `commit`, то они помещаются в основной ствол, который и становится *главной версией*.

Первоначальный импорт

Для первоначального помещения текстов в хранилище следует использовать команду `import`. Когда вы используете `import` для поддержки ваших изменений в поставляемых со стороны текстах, то тег `vendor tag` и теги `release tags` очень полезны. Тег `vendor tag` является символьным именем ветви. Обычно эта ветвь имеет номер 1.1.1, если вы не использовали параметр `-b BRANCH`. Теги `release tags` являются символьными именами отдельных версий, например, `SF1005`.

Заметим, что команда `import` не изменяет содержимого каталога, откуда импортируются исходные тексты. В частности, команда не устанавливает этот каталог в качестве рабочего каталога системы CVS. Если вы хотите работать с исходными текстами, то их следует импортировать, затем выполнить команду `checkout` в рабочий каталог.

Положим, вы имеете текст программы с именем wusage в каталоге `wusage-0.1` и хотели бы сделать изменения в тексте для ваших персональных нужд так, чтобы использовать эти изменения и в будущих версиях wusage. Вы начинаете с помещения ваших текстов в хранилище:

```
$ cd wusage-0.1
$ cvs import -m "Import of WUSAGE v. 0.1" wusage WDIST
WUSAGE_0_1
```

Вендорный тег имеет имя `WDIST`, а единственный тег версии имеет имя `WUSAGE_0_1`.

Изменение модуля командой import

Когда новая версия исходных текстов получена от поставщика, вы помещаете ее в хранилище той же самой командой `import`, которую вы использовали при создании хранилища. Единственное отличие состоит в используемых тегах:

```
$ tar xfz wusage-0.2.tar.gz
$ cd wusage-0.2
$ cvs import -m "Import of WUSAGE v. 0.2" wusage WDIST
WUSAGE_0_2
```

Для файлов, которые не были модифицированы локально, вновь создаваемая версия становится главной версией. Если вы сделали какие-то изменения в конкретном файле, то команда `import` предупредит вас, что вы должны объединить изменения в основном стволе и попросит вас выполнить `checkout -j`.

```
$ cvs checkout -jWDIST:yesterday -jWDIST wusage
```

Вышеприведенная команда создаст рабочую копию наиболее свежей версии wusage и объединит в рабочей копии изменения, сделанные в вендорной ветви со вчерашнего дня. Если возникнут конфликты, то они разрешаются обычным образом (см. «Простой пример разрешения конфликта при объединении версий»). После этого модифицированные файлы могут быть помещены в хранилище командой `commit`.

Использование даты, как описано выше, предполагает, что вы импортируете не более одной версии программного продукта в день. Если же вы импортируете более одной версии

программного продукта в день, то вам следует использовать команду:

```
$ cvs checkout -jWUSAGE_0_1 -jWUSAGE_0_2 wusage
```

Возврат к исходному варианту

Вы можете проигнорировать все локальные изменения и получить наиболее свежую версию, которую поставил вам вендор. Это выполняется переходом к вендорной ветви для всех файлов. Например, если вы имеете копию исходных текстов в локальном рабочем каталоге `work.d/wusage` и хотите вернуться к наиболее свежей версии вендора, то вы должны сделать

```
$ cd work.d/wusage
$ cvs admin -bWDIFF .
```

Не должно быть пробелов между `-b` и значением в последней строке примера.

Как управлять подстановкой ключевых слов во время импорта

Тексты программ, которые вы импортируете, могут содержать ключевые слова. К примеру, вендор (поставщик) может использовать систему CVS или другую систему поддержки версий, которая использует ключи. Если вы просто импортируете файлы в режиме, который используется по умолчанию, то будут присвоены значения ключей для вашего варианта CVS. Однако может оказаться более удобным поддерживать подстановку ключевых слов, которые обеспечивает поставщик, поскольку такая подстановка обеспечивает вас информацией об источнике, откуда получены тексты.

Чтобы обеспечить подстановку ключевых слов, установленную поставщиком, следует использовать параметр `-ko` в команде `import`, когда вы импортируете файл в первый раз. Этот параметр выключит подстановку ключевых слов для всего файла. Если вы захотите селективно выбирать режим подстановки, то вам следует подумать, как использовать параметр `-k` в командах `update` и `admin`.

Несколько вендорных ветвей

Все примеры до сих пор предполагали, что имеется лишь одна вендорная ветвь, из которой вы получаете тексты программ. В ряде случаев вы можете получить тексты из нескольких мест. Например, вы связаны с проектом, который ведут несколько групп разработчиков. Имеется несколько путей, как можно управлять несколькими источниками текстов, но один из самых эффективных — это сложить все варианты в хранилище CVS.

Для управления ситуацией, когда имеется более чем один поставщик, полезно использовать параметр `-b` в команде `cvs import`. По умолчанию `-b 1.1.1`. Для простоты, предположим, что у нас две группы разработчиков: красные и голубые. Они посылают вам тексты своих программ. Когда вы хотите работать с текстами красных, вы используете тег `RED`. Если вы хотите использовать тексты голубых, то следует использовать тег вендора `BLUE`. Команды получения текстов могли бы быть такими:

```
$ cvs import dir RED RED_1-0
$ cvs import -b 1.1.3 dir BLUE BLUE_1-5
```

Заметим, что если вендорный тег не соответствует значению параметра `-b`, CVS не распознает такой случай. Например,

```
$ cvs import -b 1.1.3 dir RED RED_1-0
```

Будьте внимательны: этот вид несоответствия гарантированно приведет к конфузиям.

Как ваша система построения программ взаимодействует с CVS

Как упоминалось во введении, CVS не обеспечивает средств для построения готовых программ из исходных текстов (таких, как `make`). В этом разделе описывается, каким образом ваша система построения программ могла бы взаимодействовать с CVS.

Как система построения могла бы соответствовать наиболее свежим версиям текстов программ? Так как CVS сама по себе обеспечивает рекурсивную обработку каталогов, то нет

нужды специально заботиться о соответствующих изменениях файла `Makefile` (или какого-то другого конфигурационного файла, который используется в вашей системе построения программ из исходных текстов), чтобы сделать его соответствующим наиболее свежей версии исходников. Вместо этого просто используйте две команды: `cvs -q update` и затем `make` для вызова вашей системы построения.

Предлагается следующий подход. Сначала выполнить команду `update`, чтобы обновить файлы в рабочем каталоге. Затем внести изменения, которые вы планируете добавить. Отладить и проверить их и лишь после выполнить операцию `commit`, то есть внести ваши изменения в хранилище. Может оказаться, что вам следует снова выполнить `update`, а затем — `commit`.

Производя постоянно только что описанную последовательность с каждой вашей разработкой, вы можете быть уверены, что ваш рабочий каталог в целом соответствует наиболее свежим версиям текстов программ.

Следующий вопрос, который является общим для многих разработок, состоит в том, как составить список версий всех файлов, которые должны использоваться для построения конечного продукта (готовой к исполнению программной системы). Наилучший способ реализовать этот вид функциональности состоит в использовании тега, то есть команды (tag), чтобы записать, какие версии каких модулей будут включены в данном варианте конечного продукта.

Использование CVS прямо в соответствии с основным ее назначением поддержки множества версий является сравнительно нетрудным. В этом случае каждый разработчик будет иметь копию дерева текстов в своем рабочем каталоге. Это дерево и будет источником для построения конечной программы или системы. Если исходное дерево невелико или разработчики рассеяны географически, то это лучший способ выполнения совместных работ. Если дерево очень велико, то есть проект велик, то стандартный способ — разбить проект на независимые части. Тогда каждая группа разработчиков будет заниматься своей частью, а не полным набором (деревом) исходных файлов.

Другой подход состоит в том, что каждый разработчик имел у себя относительно небольшой фрагмент кодов, который он отлаживает или модифицирует, а остальная часть проекта содержится в централизованном хранилище. Многие люди работают с некоторыми такими системами, используя ссылки или VPATH, которая имеется во многих версиях утилиты make. Одно из таких средств, полезных при построении программ, можно найти на ftp://ftp.cs.colorado.edu/pub/distribs/odin.

Специальные файлы

В обычных обстоятельствах CVS работает только с регулярными файлами. Предполагается, что каждый файл проекта является постоянным (persistent): он может быть открыт, прочитан, закрыт и т. д. CVS игнорирует права доступа к файлу и права собственности; эти моменты по предположению разрешаются во время установки. Таким образом, невозможно внести в хранилище устройство (файл устройства). Если файл устройства не может быть открыт, то CVS не станет с ним работать. Обычно файлы теряют признаки прав доступа и прав собственности после помещения в хранилище.

Если в хранилище установлена конфигурационная переменная `$PreservePermissions`, то CVS сохранит некоторые характеристики файла:

- принадлежность данной группе и данному пользователю;
- права доступа;
- главный и дополнительный номера устройств;
- ссылки;
- постоянную ссылочную структуру.

Если установлена переменная `$PreservePermissions`, то это сильно влияет на поведение CVS. Часть операций CVS будет возможна только для пользователя с именем `root`.

Часть команд CVS не могут быть выполнены успешно, например, `cvs status`, так как команда не распознает постоянную ссылочную структуру.

Более серьезные последствия могут быть, когда CVS полагает файл с измененными правами доступа измененным файлом. В этом случае команда `update` может заменить все или часть прав доступа в вашем рабочем каталоге.

Изменение постоянных ссылок в каталоге CVS – весьма деликатная операция.

Игнорирование файлов посредством cvsignore

Имеются несколько файлов, которые часто встречаются в вашем рабочем каталоге, но вы не планируете их запоминать в хранилище и хотели бы, чтобы CVS не обращала на них внимание. Такими файлами могут быть, например, файлы типа `*.ps`, `*.bak`, `core` и прочие. Список шаблонов имен файлов, которые CVS игнорирует по умолчанию, приведен ниже:

```
RCS          SCCS     CVS      CVS.adm
RCSLOG       cvslog.*
Tags     TAGS
.make.state          .nse_depinfo
*~       #*       .#*       ,*       _$*      *$
*.old    *.bak    *.BAK     *.orig   *.rej    .del-*
*.a      *.olb    *.o       *.obj    *.so     *.exe
*.Z      *.elc    *.ln
core
```

Могут существовать также дополнительные списки шаблонов имен файлов, которые необходимо игнорировать.

- Список для хранилища в файле `$CVSROOT/CVSROOT/cvsignore` добавляется к вышеприведенному списку.
- Список для пользователя в файле `.cvsignore`, который располагается в главном домашнем каталоге, также добавляется к вышеприведенному списку.
- Любые шаблоны, которые хранятся в переменной окружения `$CVSIGNORE`, также добавляются к вышеприведенному списку.
- Любые шаблоны, добавленные параметром `-I`.

- Любой подкаталог в вашем рабочем каталоге может содержать файл `.cvsignore`. Таким образом, любые шаблоны из этого файла будут добавлены к ранее сформированным спискам шаблонов имен файлов. Однако список из этого файла действует лишь на тот подкаталог, в котором он хранится.

В любом из перечисленных 5 мест вы можете поставить восклицательный знак, который очистит весь список шаблонов имен файлов, которые CVS должна игнорировать. Таким образом, вы имеете возможность сохранять даже те файлы, которые CVS игнорирует по умолчанию.

Определяя `-I !` для команды `cvs import` означает, что вы будете импортировать все подряд, без разбора, что оказывается полезным, если нет каких-то экстраординарных файлов. Перед выполнением такой команды полезно удалить все файлы `.cvsignore` из импортируемых каталогов во избежание недоразумений.

Заметим, что простой синтаксис для игнорирования файлов не позволяет указывать имена файлов, содержащих пробелы. Пробелы здесь являются разделителями. Также нет способов указать комментарии.

Простой пример разрешения конфликта при объединении версий

Предположим, что версия 1.4 файла `driver.c` содержит следующее:

```
#include <stdio.h>

void main(){
       parse();
       if (nerr == 0)
           gencode();
       else
           fprintf(stderr, "No code generated.\n");
       exit(nerr == 0 ? 0 : 1);
}
```

А версия 1.6 того же файла содержит:

```
#include <stdio.h>

int main(int argc,
       char **argv){
       parse();
       if (argc != 1)
       {
           fprintf(stderr, "tc: No args expected.\n");
           exit(1);
       }
           if (nerr == 0)
               gencode();
           else
               fprintf(stderr, "No code generated.\n");
           exit(!!nerr);
}
```

В то же время вы работаете с рабочей копией, которая основывается на версии 1.4. Такая ситуация легко может возникнуть, если два человека независимо модернизируют один файл. Итак, ваша рабочая копия содержит следующее:

```
#include <stdlib.h>
#include <stdio.h>

void main(){
       init_scanner();
       parse();
       if (nerr == 0)
           gencode();
       else
           fprintf(stderr, "No code generated.\n");
       exit(nerr == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
}
```

Теперь вам надо выполнить команду `cvs update`. При выполнении команды вы получаете диагностику:

```
$ cvs update driver.c
RCS file: /usr/local/cvsroot/yoyodyne/tc/driver.c,v
retrieving revision 1.4
retrieving revision 1.6
Merging differences between 1.4 and 1.6 into driver.c
rcsmerge warning: overlaps during merge
cvs update: conflicts found in driver.c
C driver.c
```

Таким образом, CVS информирует вас, что возникли конфликты при обновлении. Ваш исходный немодифицированный рабочий файл будет сохранен под именем `.\#driver.c.1.4`. А новая версия файла `driver.c` теперь содержит следующее:

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc,
          char **argv){
      init_scanner();
      parse();
      if (argc != 1)
      {
          fprintf(stderr, "tc: No args expected.\n");
              exit(1);
      }
      if (nerr == 0)
          gencode();
      else
          fprintf(stderr, "No code generated.\n");
<<<<<<< driver.c
exit(nerr == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
=======
exit(!!nerr);
>>>>>>> 1.6
}
```

Заметим, что все *не* перекрывающиеся модификации выполнены в новом варианте рабочей копии файла `driver.c`. В то же время, перекрывающиеся части ясно показаны маркерами `<<<<<<<`, `=======` и `>>>>>>>`. Вы можете разрешить конфликт простым редактированием, удалив ошибочные строки и маркеры.

Предположим, вы получили следующий файл:

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc,
          char **argv){
      init_scanner();
      parse();
      if (argc != 1)
      {
          fprintf(stderr, "tc: No args expected.\n");
```

```
        exit(1);
    }
    if (nerr == 0)
        gencode();
    else
        fprintf(stderr, "No code generated.\n");
    exit(nerr == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
}
```

Теперь вы можете выполнить операцию `commit` и получите версию 1.7:

```
$ cvs commit -m "Initialize scanner. Use symbolic exit
values." \
driver.c
Checking in driver.c;
/usr/local/cvsroot/yoyodyne/tc/driver.c,v  <--  driver.c
new revision: 1.7; previous revision: 1.6
done
```

CVS не будет ничего менять в хранилище (не выполнит команду `commit`), если в файле остались неразрешенные конфликты. Сейчас, чтобы разрешить конфликт, вам необходимо сменить дату модификации файла. Если в файле остались маркеры, то CVS предупредит вас об этом, но создаст новую версию в хранилище.

В компоненте pcl-cvs (GNU emacs интерфейс для CVS) имеется специальный пакет, который может помочь разрешать конфликты. Подробности следует смотреть в описании pcl-cvs, например, http://www.loria.fr/~molli/cvs/pcl--cvs/.

Список команд CVS

Команды CVS выглядят таким образом:

```
CVS [global-options] COMMAND [command-options] [command-
args]
```

Далее следует список команд системы CVS:

- `add [options] [files]` — добавить в хранилище новый файл/каталог.
  - `-k kflag` — установить подстановку ключевых слов;
  - `-m msg` — установить описание файла (комментарий).

- `admin [options] [files ...]` — администрирование протокольными файлами в хранилище.
  - `-b[rev]` — установить ветвь по умолчанию;
  - `-cstring` —установить начало комментария;
  - `-ksubst` —определить подстановку ключевых слов;
  - `-l[rev]` — ограничить доступ версии `rev` или последней версии;
  - `-mrev:msg` — заменить комментарий в хранилище;
  - `-orange` — удалить версии из хранилища;
  - `-q` — минимум сообщений;
  - `-sstate[:rev]` — установить состояние файла;
  - `-t` —ввести описание (комментарий) из стандартного ввода;
  - `-tfile` — ввести описание (комментарий) из файла с именем `file`;
  - `-t-string` — заменить описание (комментарий) на строку `string`;
  - `-u[rev]` — открыть доступ к версии `rev` или к последней версии.
- `annotate [options] [files ...]` — показать измененные строки из последней версии файлов.
  - `-D date` — показать наиболее свежую версию, но не позднее даты `date`;
  - `-f` — использовать версию из основного ствола разработки, если указанный тег или дата не найдены;
  - `-l` — выполнять локально без подкаталогов;
  - `-R` — выполнять рекурсивно с подкаталогами;
  - `-r tag` — показать измененные строки из версии `tag`.
- `checkout [options] modules...` — получить копию исходных текстов.
  - `-A` — сбросить любые липкие теги, даты, параметры;
  - `-c` — вывести базу данных с именами модулей;

- `-D date` — получить копию исходных текстов до даты `date`;
- `-d dir` — получить копию исходных текстов в каталог `dir`;
- `-f` — использовать версии из основного ствола, если указанные теги или даты не найдены;
- `-j rev` — объединить изменения;
- `-k kflag` — использовать строку `kflag` для постановки ключевых слов;
- `-l` — выполнять локально без подкаталогов;
- `-N` — не сокращать путь к модулю, если используется `-d`;
- `-n` — не выполнять программу модуля, если она есть;
- `-P` — пропустить пустые каталоги;
- `-p` — вывести копию исходных текстов на стандартное устройство вывода (избегая липкости);
- `-R` — выполнять рекурсивно, включая подкаталоги (по умолчанию);
- `-r tag` — получить копию исходных текстов версии `tag`;
- `-s` — то же что `-c`, но включая состояние модуля.

- `commit [options] [files ...]` — проверить есть ли изменения в рабочем каталоге, и перенести изменения файлов из рабочего каталога (если они имели место) в хранилище.
  - `-F file` — прочесть комментарий на изменение из файла `file`;
  - `-f` — отметить файл проверенным; запретить рекурсию;
  - `-l` — выполнять локально без подкаталогов;
  - `-m msg` — использовать строку `msg` в качестве комментария, который будет записан в журнал (log-файле);
  - `-n` — не выполнять программу модуля, если таковая есть;
  - `-R` — выполнять рекурсивно с подкаталогами (по умолчанию);
  - `-r rev` — выполнить `commit` для версии `rev`.

- `diff [options] [files ...]` — показать различия между версиями файлов. В дополнение к нижеописанным параметрам имеется ряд дополнительных параметров, которые используются для описания вывода программы.
  - `-D date1` — получить различия между версией с определенной датой и версией в рабочем каталоге;
  - `-D date2` — получить различия между `date1/rev1` и `date2`;
  - `-l` — выполнять локально без подкаталогов;
  - `-N` — включить `diff` для добавленных и удаленных файлов;
  - `-R` — выполнять рекурсивно с подкаталогами (по умолчанию);
  - `-r rev1` — получить различия между версией `rev1` и рабочим каталогом;
  - `-r rev2` — получить различия между версией `date1/rev1` и `rev2`.
- `edit [options] [files ...]` — перейти к редактированию наблюдаемого (watched) файла.
  - `-a actions` — определить действия для временного наблюдения, где действия могут быть такими: `edit`, `unedit`, `commit`, `all`, `none`;
  - `-l` — выполнять локально без подкаталогов;
  - `-R` — выполнять рекурсивно с подкаталогами (по умолчанию).
- `editors [options] [files ...]` — показать, кто редактирует наблюдаемый файл.
  - `-l` — выполнять локально без подкаталогов;
  - `-R` — выполнять рекурсивно с подкаталогами (по умолчанию).
- `export [options] modeules...` — экспортировать файлы из CVS.
  - `-D date` — экспортировать версию, датированную `date`;
  - `-d dir` — экспортировать версию в каталог `dir`;

- `-f` — использовать версии из основного ствола, если указанные теги или даты не найдены;
- `-k kflag` — установить подстановку ключевых слов;
- `-l` — выполнять локально; без подкаталогов;
- `-N` — не сокращать путь к модулю, если используется `-d`;
- `-n` — не выполнять программу модуля, если она есть;
- `-P` — пропустить пустые каталоги;
- `-p` — вывести копию исходных текстов на стандартное устройство вывода (избегая липкости);
- `-R` — выполнять рекурсивно, включая подкаталоги (по умолчанию);
- `-r tag` — получить копию исходных текстов версии `tag`.

- `history [options] [files ...]` — показать историю доступов к хранилищу.
  - `-a` — все пользователи;
  - `-b str` — назад к записи содержащей строку `str` в поле `module/file/repos`;
  - `-c` — отчет о модифицированных (committed) файлах;
  - `-D date` — начиная с даты `date`;
  - `-e` — отчет о записях всех типов;
  - `-l` — отчет о наиболее поздних изменениях;
  - `-m module` — отчет о модуле `module`;
  - `-n module` — в модуле `module`;
  - `-o` — отчет о прочитанных (checkout) модулях;
  - `-r rev` — начиная с версии `rev`;
  - `-T` — выдать отчет о всех тегах;
  - `-t tag` — начиная с момента, когда тег `tag` попал в файл истории (от любого пользователя);
  - `-u user` — для пользователя `user`. Может использоваться несколько раз в командной строке;
  - `-w` — рабочий каталог должен соответствовать;

- `-x types` — отчет о типах `types`, `types` может включать набор букв из следующих: `TOEFWUCGMAR` (одна буква — один тип);
- `-z zone` — вывод для временной зоны `zone`.

`import [options] repository vendor-tag release-tags...` — внести файлы в хранилище CVS, используя ветвь поставщика.

- `-b bra` — внести ветвь `bra` в ветвь поставщика;
- `-d` — использовать время модификации файлов в качестве времени импорта;
- `-k kflag` — установить режим подстановки ключевых слов по умолчанию;
- `-m msg` — использовать `msg` как запись в журнале (log-файле);
- `-I ign` — файлы, которые следует игнорировать;
- `-W spec` — фильтры (wrappers).

`init` — создать хранилище, если оно не существует.

`log` — напечатать историю файлов в хранилище.

- `-b` — только список версий в ветви по умолчанию;
- `-d dates` — определить даты (`D1<D2`) для интервала и `D` для свежайшей версии до даты `D`);
- `-h` — напечатать только заголовки;
- `-l` — локально без подкаталогов;
- `-N` — не перечислять теги;
- `-R` — печатать только имена файлов RCS;
- `-rrevs` — печатать только версии `revs`;
- `-s states` — только версии с определенным состоянием;
- `-t` — только заголовок и описательный текст (комментарий);
- `-wlogins` — только список версий, внесенных определенными пользователями;

- `login` — запрос на аутентификацию пользователя.
- `logout` — удалить запомненный пароль для аутентифицирующего сервера.
- `rdiff [options] modeules...` — показать различия между версиями.
  - `-c` — контекстное различие (по умолчанию);
  - `-D date` — выбрать версии, базирующиеся на дате `date`;
  - `-l` — локально без подкаталогов;
  - `-R` — рекурсивно с подкаталогами (умолчание);
  - `-r rev` — выбрать версии, базирующиеся на версии `rev`;
  - `-s` — краткий формат — одна строка на файл;
  - `-t` — различие между двумя последними версиями;
  - `-u` — выводной формат `unidiff`;
  - `-V vers` — использовать RCS версию `vers` для подстановки ключевых слов (устаревшее).
- `release [options] directory` — отметить для CVS, что каталог `DIRECTORY` более не используется.
  - `-d` — удалить данный каталог.
- `remove [options] [files ...]` — удалить вход (файл, каталог) в хранилище.
  - `-f` — вычеркнуть файл до его удаления.
  - `-l` — локально; без подкаталогов.
  - `-R` — рекурсивно с подкаталогами (умолчание).
- `rtag [options] tag modules ...` — добавить символьный тег к модулю.
  - `-a` — очистить тег из удаленных файлов, которые иначе не смогли бы быть помечены тегом;
  - `-b tag` — создать ветвь с именем `tag`;
  - `-D date` — пометить версию с датой `date`;
  - `-d` — удалить данный тег;
  - `-F` — переместить тег, если он уже существует;

- `-f` — использовать основной ствол, если указанные тег или дата не найдены;
- `-l` — локально; без подкаталогов;
- `-n` — не выполнять программу тега;
- `-R` — рекурсивно с подкаталогами (по умолчанию);
- `-r tag` — пометить существующий тег `tag`.

- `status [options] files ...` — отобразить информацию о состоянии в рабочем каталоге.
  - `-l` — локально без подкаталогов;
  - `-R` — рекурсивно с подкаталогами (по умолчанию);
  - `-v` — включить информацию о тегах в каждом файле.
- `tag [options] tag [files ...]` — добавить символьный тег к файлам, скопированным (checkouted) из хранилища в рабочий каталог.
  - `-b` — создать ветвь с именем `tag`;
  - `-D date` — пометить тегом версию с датой `date`;
  - `-d` — удалить данный тег;
  - `-F` — переместить тег, если он уже существует;
  - `-f` — использовать основной ствол, если указанные тег или дата не найдены;
  - `-l` — локально без подкаталогов;
  - `-n` — не выполнять программу тега;
  - `-R` — рекурсивно с подкаталогами (по умолчанию);
  - `-r tag` — пометить тегом существующий тег `tag`.
- `unedit [options] [files ...]` — отменить действие команды `edit`.
  - `-a actions` — определить действия для временного наблюдения, где действиями могут быть: `edit`, `unedit`, `commit`, `all`, `none`;
  - `-l` — локально; без подкаталогов;
  - `-R` — рекурсивно с подкаталогами (по умолчанию);

- `update [options] [files ...]` — привести рабочее дерево (в рабочем каталоге) в соответствии с хранилищем.
  - `-A` — сбросить любые липкие теги/даты/параметры;
  - `-D date` — получить из хранилища наиболее свежие версии файлов, но не позднее даты `date`;
  - `-d` — создать каталоги;
  - `-f` — использовать основной ствол, если указанные тег или дата не найдены;
  - `-I ign` — игнорировать файлы из списка `ign` (восклицательный знак (`!`) означает отмену игнорирования);
  - `-j rev` — объединить версии;
  - `-k kflag` — установить подстановку ключевых слов;
  - `-l` — локально без подкаталогов;
  - `-P` — опустить пустые каталоги;
  - `-p` — выдать файлы на стандартное устройство вывода;
  - `-R` — рекурсивно с подкаталогами (по умолчанию);
  - `-r tag` — использовать при выводе из хранилища версию `tag`;
  - `-W spec` — установить фильтры (wrappers).
- `watch [on|off|add|remove] [options] [files ...]` — включить/выключить доступ *только чтение* файлов, которые скопированы (checkouted) из хранилища.
  - `-a actions` — определить действия для временно наблюдаемых файлов. Действия могут быть `edit`, `unedit`, `commit`, `all`, `none`;
  - `-l` — локально без подкаталогов;
  - `-R` — рекурсивно с подкаталогами (по умолчанию).
- `watchers [options] [files ...]` — показать, кто наблюдает за файлами (то есть выдал ранее команду `watch`).
  - `-l` — локально без подкаталогов;
  - `-R` — рекурсивно с подкаталогами (по умолчанию).

Описание команд CVS

Общий формат команд CVS имеет следующий вид:

```
cvs [cvs_options] cvs_command [command_options]
[command_args]
```

- Здесь: `CVS` — имя программы CVS.
- `cvs_options` — некоторые параметры программы CVS, которые воздействуют на все команды программы CVS.
- `cvs_command` — одна из команд программы CVS. Некоторые из этих команд имеют псевдонимы, которые могут быть использованы вместо основного имени команды. Рассматриваются только две ситуации, когда может быть опущено поле команды `cvs_command`:
  - `cvs -H` — показать некоторые вспомогательные возможности системы CVS;
  - `cvs -v` — показать версию программы CVS.
- `command_options` — список параметров команды CVS.
- `command_args` — аргументы команды.

Можно, к сожалению, перепутать `cvs_options` и `command_options`. Например, `-l` как параметр программы CVS воздействует лишь на некоторые команды. Когда `-l` используется на месте `command_options`, он имеет совершенно другое значение и воспринимается большим числом команд.

Коды завершения CVS

CVS может устанавливать код завершения. Если все нормально, то CVS устанавливает нулевой код завершения. Исключение составляет `cvs diff`. В данном случае нулевой код завершения будет установлен, если не найдено различий в сравниваемых файлах. В противном случае будет установлен ненулевой код завершения.

Инициализационный файл CVS: .cvsrc}

Часть значений поля `command_options` могут быть установлены в инициализационном файле `.cvsrc`.

Формат файла `.cvsrc` очень прост. Файл просматривается программой CVS, в поиске строк, которые начинаются с имени текущей команды CVS (поля `cvs_command`). Если такая строка найдена, то остаток строки рассматривается как последовательность параметров данной команды, которые используются *до* любых параметров, использованных в командной строке.

Если команда имеет два имени (основное имя и синоним), то основное имя (а не синоним) будет использоваться при сканировании файла `.cvsrc`, независимо от того, какое имя (синоним или основное) использовалось в командной строке. Предположим, что содержимое файла `.cvsrc` таково, как приведено ниже

```
log -N
diff -u
update -P
co -P
```

Если вы введете команду `cvs checkout foo`, то CVS добавит параметр `-P`, поскольку `co` есть синоним `checkout`.

Если вы используете имя `cvs`, то, тем самым, сможете установить глобальные параметры CVS. Например, строка

```
cvs -z6
```

в файле `.cvsrc` будет означать, что CVS будет использовать уровень компрессии (сжатия данных при передаче по сети) 6.

Глобальные параметры CVS

Здесь перечислены имеющиеся параметры (поле `cvs_options`, которое находится слева от поля `cvs_command`).

`--allow-root=rootdir` — определить допустимое значение для имени корневого каталога `rootdir`.

- `-a` — аутентифицировать все обмены между клиентом и сервером CVS. Воздействует лишь на клиентскую часть.
- `-b bindir` — используется для совместимости с прежними версиями CVS. В текущей версии CVS ничего не делается.

- `-T tempdir` — определяет каталог, где будут находиться временные файлы. Каталог должен задаваться абсолютным именем. Параметр заменяет значение, которое было определено при трансляции, а также то, которое было определено в переменной окружения `$TMPDIR`.
- `-d cvs_root_directory` — использовать `cvs_root_directory` как корневой каталог, вместо определенного в переменной окружения `$CVSROOT`.
- `-e editor` — использовать редактор текста `editor` для ввода комментариев во время операции `commit` вместо другого редактора текста.
- `-f` — не читать конфигурационный файл `.cvsrc`.
- `-H` или `--help` — вывести информацию об использовании команды `cvs_command`.

  `-l` — не записывать поле `cvs_command` в историю команд, но выполнить ее.

  `-n` — не изменять никаких файлов. Параметр полезен, если вы хотите использовать незнакомую команду или не уверены в ее правильности. Ничего не будет изменено, но диагностика будет похожа на ту, что выдается при реальном выполнении команды.
- `-Q` — выполняться без диагностики. Сообщения будут выданы лишь в случае очень серьезных проблем.
- `-q` — выполнение команд не должно сопровождаться обширными сообщениями.
- `-r` — создать новые рабочие файлы с режимом доступа *только чтение*. Действия аналогичны установленной переменной окружения `$CVSREAD`. По умолчанию все рабочие файлы имеют доступ *чтение и запись*, если не использована команда `watches`.
- `-s variable=value` — установить переменную окружения пользователя.
- `-t` — трассировать выполнение программы; отображать шаги выполнения программы CVS. Может оказаться полезной совместно с параметром `-n`, чтобы понять, как выполняется конкретная команда CVS.

- `-v` — отобразить версию программы CVS.
- `-x` — зашифровать все обмены данными между клиентом и сервером. Оказывает влияние лишь на клиента. По умолчанию возможность шифровки не обеспечивается. Она должна быть разрешена специальной конфигурационной переменной `--enable-encryption`, когда вы строите программу CVS.
- `-z gzip-level` — установить уровень сжатия (компрессии) данных при передаче по сети. Имеет смысл лишь на стороне клиента.

Общие параметры команд CVS

Этот раздел описывает параметры, которые являются общими для многих команд CVS. Эти параметры всегда находятся справа от команд CVS. Не все команды CVS поддерживают все общие параметры. Заметным исключением является команда `history`, которая интерпретирует параметры иначе.

- `-D date` — использовать наиболее свежую версию, но не позднее даты `date`. Здесь аргумент `date` означает одиночный аргумент, который имеет смысл *даты* создания или обновления файла или просто *даты* в прошлом.

  Эта спецификация является *липкой*, когда вы используете ее для создания частной рабочей копии исходного текста. Таким образом, когда вы получаете рабочий файл с использованием параметра `-D`, то CVS записывает дату, которую вы определили, так что дальнейшие изменения в том же каталоге будут использовать ту же самую дату.

  Параметр `-D` используется в командах `checkout`, `diff`, `export`, `history`, `rdiff`, `rtag`, `update`, хотя в команде `history` этот параметр интерпретируется чуть иначе, чем в остальных.

  CVS понимает довольно широкий набор форматов, в которых может быть представлена дата. Используется стандартное представление в соответствии со стандартом ISO 8601 (ISO означает International Standards

Organization) и в соответствии со стандартами, используемыми в электронной почте (RFC-822).

Даты в соответствии со стандартом ISO 8601 могут выглядеть различно:

`1999-04-06` — означает 6 апреля 1999 года;

`1999-04-06 10:44` — означает 6 апреля 1999 года 10 часов 44 минуты.

При этом время интерпретируется в соответствии с локальной временной зоной, если не указано специально что-то иное.

В дополнение к стандартам, принятым в электронной почте (имеется в виду Internet e-mail) CVS распознает некоторые сокращения, например:

```
10 Apr 1999 10:05
10 Apr
```

Описанные два формата дат являются рекомендованными, однако воспринимаются и другие форматы, которые часто привязаны к национальным правилам и не являются хорошо документированными.

Один из таких форматов `MONTH/DAY/YEAR`.

Не забудьте использовать кавычки в командной строке, когда будете использовать параметр `-D`, например:

```
cvs rdiff -D "4 days ago" -s BOOK/CVS_Struct.tex
```

- `-f` — когда вы определяете частную дату или тег для команд CVS, то файлы, не соответствующие данной дате или тегу, обычно игнорируются. Если же указан параметр `-f` и не найдены файлы, соответствующие заданной дате, то будет произведен поиск самой свежей версии данного файла.

  `-f` может использоваться в командах `annotate`, `checkout`, `export`, `rdiff`, `rtag`, `update`.

  Команды `commit` и `remove` также могут использовать параметр `-f`, но он интерпретируется чуть иначе.

- `-k kflag` — изменить стандартную подстановку ключевых слов на значение `kflag`. Ваша спецификация `kflag` является *липкой*, когда вы используете параметр, чтобы создать вашу частную рабочую копию исходного текста. Таким

образом, когда вы используете этот параметр вместе с командами update или checkout, то CVS связывает заданный вами kflag с вашим файлом и продолжает использовать это значение параметра -k с последующими командами до тех пор, пока вы не определите явно что-то другое.

Параметр -k может использоваться с командами add, checkout, diff, import, update.

- -l — локально; работать только в текущем каталоге, без подкаталогов.

ПРИМЕЧАНИЕ Это не тот же параметр, что CVS -l, который используется *слева* от команды CVS.

Параметр -l используется с командами annotate, checkout, commit, diff, edit, editors, export, log, rdiff, remove, rtag, status, tag, unedit, update, watch, watchers.

- -m message — использовать message в качестве информации, которая будет записана в журнал (или протокол — log-файл) вместо вызова редактора текста. Используется в командах add, commit, import.
- -n — не выполнять никакие программы для команд checkout, commit и tag. Вызов таких программ записывается в базе данных (файле) modules. Параметр -n предотвращает вызов этих программ.

  Параметр может использоваться с командами checkout, commit, export, rtag.

ПРИМЕЧАНИЕ Это не тот же параметр -n, который может использоваться слева от команды CVS.

- -P — опустить пустые каталоги.
- -p — направить найденные в хранилище файлы на стандартное устройство вывода, а не записывать их в текущий каталог. Параметр может быть использован вместе с командами checkout, update.
- -R — обрабатывать каталоги рекурсивно, начиная с текущего каталога. Такой стиль обработки установлен по умолчанию.

Этот параметр может использоваться с командами `annotate`, `checkout`, `commit`, `diff`, `edit`, `editors`, `export`, `rdiff`, `remove`, `rtag`, `status`, `tag`, `unedit`, `update`, `watch`, `watchers`.

`-r tag` — использовать версию, указанную в `tag` вместо главной (наиболее свежей) версии исходных текстов по умолчанию. В качестве `tag` могут использоваться как произвольные теги, установленные командами `tag` и `rtag`, так и два стандартных тега: `HEAD` — тег, обозначающей наиболее свежую версию в хранилище, и `BASE` — тег, обозначающий версию в хранилище, которая была переписана последней в ваш рабочий каталог с помощью команды `checkout`.

Спецификация параметра `-r` является «липкой», когда вы используете этот параметр для команд `checkout` или `update`, чтобы образовать рабочую копию исходного текста в рабочем каталоге. CVS помнит эти теги и будет использовать указанные значения в дальнейших командах, если вы не укажете явно обратное. Тег может быть как символьным, так и числовым значением.

Использование глобального параметра `-q` вместе с параметром `-r` часто оказывается полезным, т.к. предотвращает предупредительные сообщения, когда файл RCS не содержит указанный тег.

ПРИМЕЧАНИЕ Обсуждаемый здесь параметр `-r` нельзя путать с глобальным параметром `-r`.

Параметр `-r` может быть использован вместе с командами `checkout`, `commit`, `diff`, `history`, `export`, `rdiff`, `rtag`, `update`.

- `-W` — Определить имена файлов, которые должны быть подвергнуты фильтрации. Этот параметр можно использовать несколько раз в одной командной строке. После параметра может идти шаблон имени файлов в таком же виде, как в файле `.cvswrappers`.

Параметр может использоваться с командами `import`, `update`.

admin — административный интерфейс для rcs

- Требуется: хранилище, рабочий каталог.
- Изменяет: хранилище.
- Синоним: rcs.

admin — административный интерфейс для RCS, которая документирована в `rcs(1)`. Команда `admin` просто передает все параметры и аргументы RCS без всякого изменения или проверки. Эта команда выполняется рекурсивно вниз по всей иерархии каталогов, начиная с текущего, поэтому следует использовать ее весьма осмотрительно.

В UNIX, если существует группа `cvsadmin`, то только члены этой группы могут выполнять команду `cvs admin`. Такая группа должна существовать на сервере и на машине, на которой выполняется CVS в режиме одиночного исполнения (не клиент/сервер). Чтобы запретить использование `cvs admin` для всех пользователей, создайте группу `cvsadmin`, которая не содержит ни одного пользователя.

Параметры команды admin

Часть из перечисленных параметров сохранились лишь по историческим соображениям и, возможно, будут исключены в будущем.

- `-Aoldfile` — добавить в конец списка доступа RCS строку `oldfile`. Может не работать с CVS.
- `-alogins` — добавить разделенный запятыми список `logins` к списку доступа файла RCS. Может не работать с CVS.
- `-b[rev]` — установить имя по умолчанию для ветви `rev`. В CVS вы обычно не манипулируете именами ветвей по умолчанию; липкие теги являются более мощным средством для этих целей. Не должно быть пробелов между `-b` и `rev`.
- `-cstring` — установить начало комментария в `string`. CVS не использует теперь этот параметр.

- `-e[logins]` — удалить имена, содержащиеся в списке, разделенном запятыми `logins`, из списка доступа в файле RCS. Если `logins` опущено, то удаляется весь список. Может не работать с CVS.
- `-I` — выполнять интерактивно, даже если стандартным вводом является не терминал. Видимо, этот параметр скоро не будет поддерживаться.

  `-i` — бесполезно использовать с CVS.
- `-ksubst` — установить значение по умолчанию для подстановки ключевых слов в `subst`. Употребление параметра `-k` в командах `update`, `export` и `checkout` будет замещать это умолчание.
- `-l[rev]` — закрыть версию с номером `rev`. Если дано имя ветви, то закрывается самая свежая версия ветви. Если `rev` опущено, то закрывается наиболее свежая версия ветви. Не должно быть пробелов между `-l` и `rev`.
- `-L` — установить строгое ограничение доступа. Строгое ограничение доступа означает, что владелец файла RCS не имеет никаких преимуществ при записи файла. Для использования с CVS должно быть включено строгое ограничение доступа.
- `-mrev:msg` — заменить текст комментария в версии `rev` текстом в строке `msg`.
- `-nname[:[rev]]` — ассоциировать символьное имя `name` с ветвью или версией `rev`. Обычно лучше использовать теги, установленные командами `tag` или `rtag`. Если двоеточие и `rev` отсутствуют, то удалить символьное имя. Если `name` уже связано с другим номером версии, то выдать сообщение об ошибке. Если `rev` есть символьное имя, то оно расширяется до связывания. Строка `rev`, содержащая номер ветви, за которым следует точка, обозначает наиболее свежую версию данной ветви. Двоеточие с пустым значением `rev` подразумевает наиболее свежую версию в ветви по умолчанию, обычно главный ствол.

  Например, `cvs admin -nname:` связывает (ассоциирует) `name` с текущими наиболее свежими версиями всех

файлов RCS, что контрастирует с `cvs admin -nname:$`, которая связывает `name` с номерами версий, извлеченными из ключевых слов соответствующих рабочих файлов.

- `-orange` — удалить версии, заданные в `range`. Заметим, что команда довольно опасна. Перед выполнением вы должны хорошо понимать, что вы делаете.

  `range` может быть в различных форматах:

  - `rev1::rev2` — выбросить все версии между `rev1` и `rev2`, таким образом, что CVS запомнит лишь различие между `rev1` и `rev2`, удалив все промежуточные шаги. Например, если вы выполнили `-o1.3::1.6`, то вы сможет позже получить версию 1.3 или 1.6, но не сможете получить версию 1.4 или 1.5 или разницу между 1.4 и 1.5. Другой пример, поясняющий работу команды: `-o 1.3::1.4`, не будет иметь никаких негативных последствий, так как нет никаких промежуточных версий между 1.3 и 1.4;
  - `::rev` — выбросить все версии между началом ветви, которая содержит `rev`, и самой `rev`. Например, `-o ::1.3.2.6` удалит версию 1.3.2.1 и т. д., но оставит 1.3 и 1.3.2.6;
  - `rev::` — удалить все версии между `rev` и концом ветви, содержащей `rev`;
  - `rev` — удалить версию `rev`;
  - `rev1:rev2` — удалить версии от `rev1` до `rev2` включительно;
  - `:rev` — удалить версии от начала ветви, содержащей `rev` до `rev` включительно;
  - `rev:` — удалить все версии, начиная с версии `rev` (включительно), до конца ветви, содержащей `rev`.

- `-q` — выключить диагностику.
- `-sstate[:rev]` — установить состояние версии `rev` в значение `state`. Если `rev` есть номер ветви, то предполагается наиболее свежая версия в ветви. Если `rev` опущено, то предполагается наиболее свежая версия в ветви по

умолчанию. Любой идентификатор может быть использован как значение `state`. Полезные значения состояний могут быть следующими:

- `Exp` — экспериментальное (experimental);
- `Stab` — стабильное (stable);
- `Rel` — для выпуска (release).

По умолчанию всем новым модулям состояние `state` устанавливается как `Exp`. Состояние видно в выводе команды `log`, а также в подстановках ключевых слов `$Log$` и `$State$`.

Заметим, что CVS использует состояние `dead` для своих внутренних целей; чтобы перевести файл в состояние `dead` или из состояния `dead`, следует использовать команды `remove` и `add`, соответственно, а не `admin`.

- `-t[file]` — записать текст комментария из `file` в файл RCS, удаляя при этом существующий текст. Имя файла `file` не может начинаться со знака минус. Не может быть пробелов между `-t` и `file`. Текстовый комментарий будет виден в выводе команды `cvs log`.

Если `file` опущен, текст будет введен с устройства стандартного ввода. В этом случае текст заканчивается знаком конца текста или знаком точка. Это не работает в схеме клиент/сервер.

- `-t-string` — записать строку `string` на место комментария в файл RCS, удалив при этом существующий текст. Не допускается пробелов между `-t` и аргументом.
- `-U` — установить нестрогое ограничение доступа. Нестрогое ограничение доступа означает, что владелец файла не закрывает доступ к версии для помещения ее в хранилище. Однако для использования с CVS должно быть установлено строгое ограничение доступа.
- `-u[rev]` — снять ограничения по доступу к версии номер `rev`. Если дано имя ветви, то снять ограничения по доступу к наиболее свежей версии в ветви. Если `rev` опущено, то удалить последние по времени ограничения по доступу, которые были установлены данным пользова-

телем. Обычно снять ограничения по доступу может лишь тот, кто установил их. Если кто-то дугой пытается снять ограничения, то это нарушит защиту и приведет к автоматической посылке сообщения тому, кто наложил ограничения по доступу. Не должно быть пробелов между `-u` и его аргументом.

- `-VN` — устаревший параметр. Вызовет ошибку при исполнении.
- `-xsuffixes` — в предыдущих версиях CVS этот параметр описывался как метод формирования имен файлов RCS. Однако сейчас CVS требует, чтобы имена файлов RCS всегда заканчивались последовательностью `,v` (запятая и строчная латинская буква v). Следовательно, этот параметр не сможет принести много пользы в системе CVS.

checkout — получение исходных текстов из хранилища для редактирования

- Формат использования:

```
cvs checkout [options] modules ...
```

- Требуется: хранилище.
- Изменяется: рабочий каталог.
- Синонимы: `co`, `get`.

По команде `checkout` буден создан рабочий каталог, содержащий копии исходных файлов, которые описаны как `modules`. Вы должны выполнить команду `checkout` до использования большинства других команд CVS, поскольку они предполагают доступ к рабочему каталогу с рабочими копиями файлов.

Аргумент `modules` представляет собой одно из двух:

- символьные имена, обозначающие некоторое множество каталогов и/или файлов;
- пути к каталогам или файлам в хранилище (относительно `$CVSROOT`).

Символьные имена определены в конфигурационном файле `modules`.

В зависимости от модулей, которые вы специфицировали, checkout может рекурсивно создавать иерархию каталогов в вашем рабочем каталоге и заполнять ее файлами из хранилища. После этого вы можете редактировать исходные тексты (файлы) в любое время, несмотря на то, что другие файлы могут в это же время редактироваться другими разработчиками. Вы можете обновлять файлы в рабочем каталоге, чтобы включить те изменения, которые были внесены другими разработчиками в файлах, содержащимися в хранилище. Наконец, вы можете все ваши изменения записать в хранилище исходных текстов (файлов).

Обратим внимание, что checkout используется для создания дерева каталогов. Верхний каталог создается в том рабочем каталоге, в котором вы вызвали CVS. Обычно создаваемый каталог имеет то же имя, что и модуль, который вы хотите отредактировать.

Файлы, которые создает команда checkout, имеют по умолчанию права доступа на чтение-запись, если не указано другое: глобальным параметром -r программы CVS, переменной окружения $CVSREAD, командой watch для данного файла.

Вторичное выполнение команды checkout в том же каталоге, в котором уже выполнялась ранее предыдущая команда checkout, вполне допустима и имеет то же значение, что и команда update с параметром -d. Иными словами, любые изменения в хранилище, которые произошли с момента предыдущей команды checkout, будут отображены в вашем рабочем каталоге.

Параметры команды checkout

Ниже приведены стандартные параметры, которые поддерживаются командой checkout.

- -D date — использовать наиболее свежую версию, но не позднее, чем дата date. Этот параметр предполагает параметр -P.
- -f — полезно использовать вместе с параметром -D или с параметром -r. Если не найдено подходящих версий, то будет произведен поиск наиболее свежих версий.

Более подробное обсуждение ветвей смотрите в разделе «Ветвление и объединение».

- `-N` — параметр имеет смысл лишь при совместном использовании с параметром `-d`. CVS предотвращает укорачивание имени файла в рабочем каталоге при выполнении команды. Смотрите описание параметра `-d`.
- `-s` — действие параметра похоже на использование `-c`, но включает состояние всех модулей и сортирует их по строке статуса.

diff — показать отличия между версиями

Формат использования:

```
diff [-lR] [format_options] [[-r rev1 | -D date1]
                [-r rev2 | -D date2]] [files...]
```

Необходимы: рабочий каталог, хранилище.

Изменения: ничего не меняется.

Команда `diff` используется, чтобы увидеть отличия в различных версиях одного файла. По умолчанию сравниваются файл в рабочем каталоге с версиями этого файла в хранилище и показываются все отличия. Если дано имя файла, то сравнивается файл с этим именем, а если дано имя каталога, то сравниваются все файлы каталога. Выработка кода завершения для команды `diff` отличается от других команд системы CVS. Эта команда возвращает значение 0 (нуль), если нет никаких различий между файлами, и ненулевое значение, если встретились различия или ошибки.

Параметры команды diff

Команда `diff` поддерживает следующий стандартный набор параметров:

- `-D date` — использовать наиболее свежую версию, но не позднее даты `date`. Смотрите параметр `-r`, чтобы яснее понять влияние на сравнение.
- `-k kflag` — обрабатывать ключевые слова в соответствии со значением `kflag`.

- `-l` — выполнять только в текущем каталоге (без подкаталогов).
- `-R` — обрабатывать файлы рекурсивно, то есть текущий каталог вместе с подкаталогами. Это значение параметра принимается по умолчанию.
- `-r tag` — сравнить с версией `tag`. Могут присутствовать: один параметр `-r`, два параметра `-r` или ни одного. Если не указан параметр `-r`, то будет производиться сравнение рабочего файла с той версией в хранилище, на которой базируется файл в рабочем каталоге. Если присутствует два параметра `-r`, то будут сравниваться две версии в хранилище (содержание рабочей копии файла в рабочем каталоге не будет иметь влияние). Один параметр `-r` или оба могут быть заменены параметрами `-D`, если это требуется.

Следующие параметры определяют формат вывода. Они имеют то же значение, что и в утилите GNU diff. В связи с этим, мы только перечислим их.

- `-0 -1 -2 -3 -4 -5 -6 -7 -8 -9`
- `--binary`
- `--brief`
- `--changed-group-format=arg`
- `-c`
- `-C nlines`
- `-e` или `--ed`
- `-t` или `--expand-tabs`
- `-f` или `--forward-ed`
- `--horizon-lines=arg`
- `--ifdef=arg`
- `-w` или `--ignore-all-space`
- `-B` или `--ignore-blank-lines`
- `-i` или `--ignore-case`
- `-I regexp`

- `--ignore-matching-lines=regexp`
- `-h`
- `-b` или `--ignore-space-change`
- `-T` или `--initial-tab`
- `-L label` или `--label=label`
- `--left-column`
- `-d` или `--minimal`
- `-N` или `--new-file`
- `--new-line-format=arg`
- `--old-line-format=arg`
- `--paginate`
- `-n` или `--rcs`
- `-s` или `--report-identical-files`
- `-p`
- `--show-c-function`
- `-y` `--side-by-side`
- `-F regexp` или `--show-function-line=regexp`
- `-H` или `--speed-large-files`
- `--suppress-common-lines`
- `-a` или `--text`
- `--unchanged-group-format=arg`
- `-u`
- `-U nlines`
- `--unified[nlines]`
- `-V arg`
- `-W columns`
- `--width=columns`

Примеры использования команды diff

Следующая строка производит сравнение между версиями 1.1 и 1.2 файла `CVS_Diff.tex`. Параметр `-u` означает Unidiff. Параметр `-kk` означает, что никакие подстановки

ключевых слов не производятся, так что не будет никакой разницы между файлами, если она порождена лишь различной подстановкой ключевых слов.

```
cvs diff -kk -u -r 1.1 -r 1.2 CVS_Diff.tex
```

Предположим, что экспериментальная ветвь EXPR1 базируется на наборе файлов, которые описываются тегом с именем RELEASE_1_0. Чтобы увидеть, что происходит с этой ветвью, можно воспользоваться командой

```
cvs diff -r RELEASE_1_0 -r EXPR1
```

Здесь будут сравниваться файлы из ветви EXPR1 и комплект файлов с тегом RELEASE_1_0.

Подобная команда может использоваться, чтобы получить изменения в версиях

```
cvs diff -c -r RELEASE_1_0 -r RELEASE_1_1 > diffs
```

Если вы ведете журнал изменений, то нижеследующая команда позволит вам перед выполнением команды `commit` подготовить записи в ваш журнал. Будут напечатаны все изменения, которые вы еще не подтвердили в хранилище.

```
cvs diff -u | less
```

export — экспорт исходных текстов из хранилища

- Использование:

  ```
  export [-flNnR] [-r rev \verb"|" -D date] [-k subst]
  [-d dir] module...
  ```

- Требует: хранилище.
- Изменения: текущий каталог.

Эта команда представляет собой особый вариант команды `checkout`. Команда `export` копирует исходный текст модуля из хранилища, но не создает административных каталогов CVS, как это делает команда `checkout`. Например, если вы предполагаете передать кому-то исходный текст модуля, то, возможно, будет удобно воспользоваться командой `export`. Эта команда требует, чтобы вы указали дату или тег (с параметром `-D` или `-r`), так что вы сможете произвести необходимые действия над текстом, который вы передаете.

Обычно полезно использовать параметр `-kv` с командой `export`. Это приведет к подстановке всех ключевых слов в файле. Следовательно, никакая информация о модуле не будет утеряна. Однако вам следует знать, что двоичные файлы не могут быть корректно экспортированы таким образом. Также следует помнить, что после использования параметра `-kv` нельзя выполнять команду `ident`, которая является частью пакета RCS (смотрите описание `ident`) и ищет ключевые строки. Если вы желаете использовать позже команду `ident`, вы не должны использовать параметр `-kv`.

Параметры команды export

Команда `export` поддерживает следующие стандартные параметры.

- `-D date` — использовать наиболее свежую версию до даты `date`.
- `-f` — если не найдено указанной версии, использовать последнюю версию (наиболее свежую).
- `-l` — локально; выполнять (экспортировать) только в локальном рабочем каталоге (без подкаталогов).
- `-n` — не выполнять никаких программ, которые вызывает `checkout`.
- `-R` — экспортировать каталоги рекурсивно, то есть включая подкаталоги.
- `-r tag` — использовать версию `tag`.

В дополнение, поддерживаются следующие параметры, которые являются общими с командой `checkout`.

- `-d dir` — создать каталог с именем `dir` для рабочих файлов вместо использования имени модуля.
- `-k subst` — установить режим подстановки ключевых слов.
- `-N` — полезно использовать лишь вместе с параметром `-d dir`. Смотрите также описание параметров команды `checkout`.

history — показать историю изменения состояния хранилища

Использование:

```
history [-report] [-flags] [-options args] [files...]
```

Требуется: файл `$CVSROOT/CVSROOT/history`

- Изменяется: ничего не изменяется.

CVS может хранить историю в файле, который содержит каждое использование команд: `checkout`, `commit`, `rtag`, `update`, `release`. Вы можете использовать команду `history`, чтобы отобразить содержание файла `$CVSROOT/CVSROOT/history` в различных форматах.

ВНИМАНИЕ Команда history интерпретирует параметры -f, -l, -n, -p иначе, чем принято в других командах CVS.

Параметры команды history

Ниже приведен список параметров для управления, какая информация будет напечатана.

- `-c` — вывести сообщение после выполнения команды `commit` (когда изменялось содержимое хранилища).
- `-e` — вывести все типы записей из файла `$CVSROOT/CVSROOT/history`.
- `-m module` — вывести сведения об отдельном модуле. Можно использовать этот параметр несколько раз в командной строке.
- `-o` — вывести информацию о том, когда выполнялась команда `checkout`.
- `-T` — вывести информацию о всех тегах.
- `-x type` — выдать из файла `$CVSROOT/CVSROOT/history` записи типа `type`. Типы записей указываются одной буквой. В одной строке можно указать более одного типа

записей, то есть использовать комбинацию букв. Часть команд имеют лишь один тип записи:

- `F` — команда `release`.
- `O` — команда `checkout`.
- `E` — команда `export`.
- `T` — команда `rtag`.
- Команда `update` может давать четыре типа записи в файл истории.
- `C` — было необходимо слияние изменений, однако обнаружены коллизии, которые могут быть устранены только вручную.
- `G` — было успешно выполнено необходимое слияние изменений.
- `U` — рабочий файл был скопирован из хранилища.
- `W` — рабочая копия файла была удалена во время выполнения команды `update`, поскольку ее нет в хранилище.
- Команда `commit` может порождать три типа записей в файле истории.
- `A` — вывести данные всех пользователей (по умолчанию выводятся данные лишь того пользователя, который выполняет команду `history`).
- `M` — файл был модифицирован.
- `R` — файл был удален.

Следующие параметры ограничивают или расширяют объем вывода без дополнительных аргументов для этих параметров.

- `-a` — показать данные для всех пользователей (по умолчанию показываются данные лишь того пользователя, который выполняет команду `history`).
- `-l` — показать только последнюю модификацию.
- `-w` — показать только записи модификации, которые были выполнены из того же рабочего каталога, где выполняется команда `history`.

Следующие параметры, за которыми следует аргумент, приводят к тому, что вывод строится в соответствии со значением аргумента.

- `-b str` — показать записи, которые содержат строку `str` в имени модуля, имени файла или в имени пути к хранилищу.
- `-D date` — показать записи, начиная с даты `date`. Это отличается от обычного использования параметра `-D` в CVS, когда оно обозначает наиболее свежую версию до даты `date`.
- `-p repository` — показать данные для отдельного хранилища. Можно использовать несколько параметров `-p` в одной командной строке.
- `-r rev` — показать записи, ссылающиеся на версии, начиная с версии или тега с именем `rev`, который появляется в отдельных файлах RCS. Поиск каждого файла производится в соответствии с версией или тегом `rev`.
- `-t tag` — показать записи, начиная с момента, когда тег с именем `tag` был добавлен к файлу истории. Это отличается от параметра `-r` (выше), в котором читается лишь файл истории, а не файлов RCS, что значительно быстрее.
- `-u name` — показать записи для пользователя с именем `name`.

import — импортирование текстов в CVS

Использование:

```
import [-options] repository vendortag releasetag...
```

Требуется: хранилище, каталог с исходными текстами.

Изменения: хранилище.

Команда `import` предназначена для внесения исходных текстов в хранилище системы CVS из внешнего источника. Вы можете использовать эту команду как для первоначального создания хранилища, так и для полного обновления в модуле из внешнего источника.

Аргумент `repository` дает имя каталога внутри корневого каталога хранилища системы CVS. Если каталог не существует, то он будет создан.

Когда команда `import` используется для обновления исходных текстов в хранилище, которые были модифицированы прошлым вызовом команды `import`, вы будет проинформированы о любых файлах, которые конфликтуют в двух ветвях разработки. Следует использовать `checkout -j`, чтобы объединить модификации.

Если существует конфигурационный файл `$CVSROOT/CVSROOT/cvswrappers`, то любые импортируемые файлы, чьи имена удовлетворяют спецификациям в упомянутом конфигурационном файле, будут рассматриваться как пакеты. К ним будет применена специальная фильтрация до выполнения реального импортирования.

Импортированный внешний комплект исходных текстов сохраняется в хранилище в ветви первого уровня, по умолчанию: 1.1.1. Обновления остаются в той же ветви. Например, файлы из импортированной коллекции исходных текстов будут иметь версию 1.1.1.1, тогда как файлы из только что импортированной и раз обновленной версии будут иметь номер версии 1.1.1.2, и т. д.

В команде `import` требуются как минимум три аргумента.

- `repository` требуется, чтобы идентифицировать данный комплект исходных текстов.
- `vendortag` является тегом для обозначения полной ветви, например, 1.1.1.
- `releasetag` используется, чтобы обозначить файлы, которые создаются каждый раз при выполнении команды `import`.

Заметим, что команда `import` не изменяет каталог, в котором она выполняется. В частности, она не устанавливает текущий каталог в качестве рабочего каталога CVS. Если вы желаете работать с исходными текстами в контексте CVS, вам следует импортировать исходные тексты, а затем выполнить команду `checkout` в другом каталоге.

Параметры команды import

Среди общих параметров, которые поддерживаются командой `import`, мы упомянем один

- `-m message` — записать строку `message` в файл протокола, вместо вызова редактора текста.

Имеются также дополнительные параметры, которые воспринимаются командой `import`.

- `-b branch` — смотрите раздел «Несколько вендорных ветвей», где рассматривается несколько ветвей поставщика (vendor branches).
- `-k subst` — требуется режим подстановки ключевых слов. Этот режим будет установлен для всех файлов, созданных во время импортирования, но не будет действовать для тех файлов, которые уже имеются в хранилище. Смотрите описание режимов подстановки ключевых слов в разделе «Подстановка ключевых слов».
- `-I name` — определить имена файлов, которые будут игнорироваться во время импортирования. Вы можете использовать этот параметр несколько раз. Чтобы избежать игнорирования файлов вообще, используйте `-I !`.

  Строка `name` может содержать то же, что и строка для игнорирования имен в файле `.cvsignore`. Смотрите раздел «Игнорирование файлов посредством cvsignore».
- `-W spec` — определить имена файлов, которые должны быть отфильтрованы во время выполнения команды `import`. Этот можно использовать несколько раз.

Строка `spec` может быть образцом имени файла такого же типа, как и в файле `.cvswrappers`. См. раздел «Файл установки фильтров».

Вывод команды import

Команда `import` информирует вас о выполняемых действиях выводом диагностических строк о состоянии каждого файла (по одной строке на файл). Каждому имени файла предшествует буква, показывающая статус файла.

- `U file` — файл с именем `file` уже существует в хранилище и не модифицирован локально. Создана новая версия файла (если это было необходимо).
- `N file` — новый файл с именем `file`, который был добавлен в хранилище.
- `C file` — файл с именем `file` уже существует в хранилище, но был модифицирован в локальном рабочем каталоге. Сначала следует выполнить операцию слияния изменений с содержимым хранилища.
- `I file` — файл с именем `file` игнорируется, смотрите раздел «Игнорирование файлов посредством cvsignore».
- `L file` — Файл с именем `file` является символьной ссылкой; команда `import` игнорирует символьные ссылки. В то же время различные параметры в файле `modules` позволяют воссоздать символьные ссылки в командах: `checkout`, `update` и т. д. См. раздел «Конфигурационный файл modules».

log — выдать протокольную информацию для файлов

- Использование:

```
cvs log [options] [files...]
```

- Требуется: хранилище, рабочий каталог.
- Изменения: ничего не изменяется.

Команда отображает информацию о файлах в хранилище и в рабочем каталоге (точнее, выводит информацию на устройство стандартного вывода). Для выполнения команда `log` вызывала ранее утилиту системы RCS. Сейчас это не соответствует действительности (по меньшей мере, для версии CVS 1.10), но стиль вывода и способ использования параметров чуть отличаются от обычного в CVS.

Вывод включает в себя расположения файла RCS, заголовок версии, все символические имена (теги) и другую информацию.

ВНИМАНИЕ Параметр -R команды log имеет значение, отличающееся от обычного внутри CVS.

Параметры команды log

По умолчанию `log` печатает всю имеющуюся информацию о файлах. Параметры используются для того, чтобы ограничить объем вывода.

- `-b` — выдать информацию о версиях в подразумеваемой ветви, обычно последняя по времени ветвь в основном стволе разработки.
- `-d dates` — выдать информацию о версиях файлов вместе с датами и временами ввода в хранилище. Даты могут быть разделены точкой с запятой, если их несколько. Даты могут быть комбинированы, как показано ниже:
  - `d1<d2; d2>d1` — выбрать версии, которые были помещены в хранилище между датами `d1` и `d2` (не включая);
  - `<d; d>` — выбрать все версии, которые были внесены в хранилище раньше даты `d`;
  - `>d; d<` — выбрать все версии, которые были внесены в хранилище позже даты `d`;
  - `d` — выбрать одну версию, датированную `d` или ранее.

  За знаками > (больше) и < (меньше) могут следовать знак = (равно), чтобы обозначить закрытый интервал, а не открытый.

  Заметим, что разделителем является знак ; (точка с запятой).
- `-h` — напечатать только имя файла RCS, имя файла в рабочем каталоге, заголовок, ветвь по умолчанию, список доступа, защиту, символические имена и суффикс.
- `-l` — локально; выполнять только в локальном каталоге (без подкаталогов). Умолчание — выполнять рекурсивно, с подкаталогами.
- `[-N]` — не выводить список тегов для файла. Этот параметр очень полезен, если в вашей группе разработчиков используется много тегов.

- `-R` — вывести только имя файла RCS.
- `-rrevisions` — вывести информацию о версиях. Нижеследующий список показывает допустимые форматы строки `revisions`:
  - `rev1:rev2` — версии от `rev1` до `rev2`, которые должны быть в одной ветви;
  - `:rev` — версии начиная с ветви по умолчанию до `rev` включительно;
  - `rev:` — версии, начиная с `rev` до конца ветви, содержащей `rev`;
  - `branch` — аргумент означает имя ветви, то есть имеются в виду все версии из этой ветви;
  - `branch1:branch2` — ряд ветвей, подразумеваются все версии из этого ряда ветвей;
  - `branch.` — наиболее свежая версия в ветви `branch` (в параметре точка на конце).

  Параметр `-r` без аргументов означает наиболее свежую версию в ветви по умолчанию. Обычно это основной ствол. Между параметром `-r` и аргументом не должно быть пробелов.
- `-s states` — выдать информацию о версиях, чьи атрибуты состояния удовлетворяют одному из состояний данных в списке состояний, разделенных запятыми `states`.
- `-t` — выдать то же самое, что и `-h` плюс поле комментариев.
- `-wlogins` — выдать информацию о версиях, которые были внесены в хранилище пользователями, имена которых перечислены в списке имен, разделенных запятыми, в строке `logins`. Если `logins` опущена, то подразумевается имя пользователя, который выполняет команду `log`. Между параметром `w` и его аргументом не должно быть пробелов.

Команда `log` выдает пересечение (intersection) версий, выбранных параметрами `-d`, `-s`, `-w`, с объединением версий, отобранных посредством параметров `-b` и `-r`.

rdiff — различия между версиями в формате patch

- Использование:

```
rdiff [-flags] [-V vn] [-r t | -D d [-r t2 | -D d2]]
modules...
```

- Требуется: хранилище.
- Изменения: ничего не меняется.
- Синоним: `patch`.

Команда `rdiff` строит входной файл для программы patch, которую разработал Larry Wall (Ларри Уолл), чтобы преобразовать старую версию исходного текста в последнюю версию. Команда `rdiff` является одной из нескольких команд CVS, которые работают напрямую с хранилищем, не требуя выполнения перед ними команды `checkout`. Вывод команды `rdiff` направляется на стандартное устройство вывода.

Используя стандартные параметры `-r` и `-D`, вы можете определить любую комбинацию одной или двух версий или дат модификации. Если указана лишь одна версия или дата, то результирующий файл будет отражать различия между указанной версией и текущей главной версией в файле RCS.

Заметим, что если выбранная программная версия содержится более, чем в одном каталоге, тогда может оказаться необходимым определить параметр `-p` для программы patch, когда будет происходить обновление старых версий. Таким образом, patch сможет найти файлы, которые находятся в других каталогах.

Параметры rdiff

Нижеследующие стандартные параметры поддерживаются командой `rdiff`:

- `-D date` — использовать наиболее свежую версию, но не позже даты `date`.
- `-f` — если не найдено соответствующей версии, то взять последнюю версию, вместо игнорирования файла.

- `-l` — локально; не обрабатывать подкаталоги, работать только в локальном каталоге.
- `-R` — обрабатывать все подкаталоги.
- `-r tag` — использовать версию `tag`.

В дополнение к вышеприведенным, имеется несколько дополнительных команд:

- `-c` — использовать контекстный формат `diff`. Параметр выполняется по умолчанию.
- `-s` — создать сводный отчет об изменениях вместо файла для программы patch. Сводный отчет включает информацию о файлах, которые были изменены или добавлены в промежутке между двумя версиями. Отчет посылается на стандартное устройство вывода. Этот параметр удобен, чтобы определить, какие файлы менялись в промежутке между двумя датами.
- `-t` — различия двух последовательных последних версий посылаются на стандартный вывод. Это одна из удобных возможностей определить, как менялся конкретный файл.
- `-u` — использовать формат unidiff для контекстного построения различий. Этот параметр недоступен для вашей программы `diff`, если она не поддерживает формат unidiff. Помните, что старые версии программы patch не понимают формат unidiff.

Примеры использования команды rdiff

Предположим, что вас спросили о последних изменениях файла `CVS_Struct.tex` в модуле `BOOK`. Проще всего составить такой запрос:

```
cvs rdiff -t BOOK/CVS_Struct.tex
```

В ответ вы получите примерно нижеследующее:

```
*** BOOK/CVS_Struct.tex:1.10    Thu Apr  1 16:24:04 1999
--- BOOK/CVS_Struct.tex Fri Apr  2 19:00:58 1999
***************
*** 69,74 ****
--- 69,75 --
  \input{CVS_Export.tex}                          % Export
  \input{CVS_History.tex}                       % History
```

```
  \input{CVS_import.tex}                    % Import
+ \input{CVS_Log.tex}              % Log
  \input{CVS_commit.tex}                    % Commit
  \input{CVS_Update.tex}                    % Update
  \input{CVS_Rtag.tex}              % Rtag
```

Здесь знаком + (плюс) помечена строка, где произошли изменения последний раз. Видно также, что последний раз файл `BOOK/CVS_Struct.tex` изменялся 2 апреля 1999, а предыдущее изменение имело место 1 апреля 1999.

Предположим, вы образовали версию 1.3 и создали ветвь с именем `R13fix` для коррекции возможных ошибок. При этом ветвь `R131` соответствует версии 1.3.1, которую вы сделали некоторое время назад. Теперь вы хотите увидеть объем работы по изменениям исходных текстов. Это можно выполнить с помощью команды:

```
cvs rdiff -s -r R131 -r R13fix module-name
```

commit — команда записи изменений в хранилище

- Формат команды:

  ```
  commit [-lnRf] [-m 'log_message' | -F file] [-r
  revision] [files...]
  ```

- Требуется: рабочий каталог, хранилище.
- Изменяется: хранилище.
- Синоним: `ci`.

Команда `commit` используется, чтобы внести изменения, произведенные в рабочем каталоге, в основное хранилище.

Если вы не определили конкретный файл или группу файлов, то CVS будет проверять, все файлы в рабочем каталоге, выясняя, изменились ли они. `commit` проверяет файлы весьма тщательно и делает изменения в хранилище лишь в случае реального изменения содержимого файла, а не просто даты создания. По умолчанию или явным указанием параметра `-R` будут проверяться все подкаталоги рабочего каталога. Вы можете ограничить область действия команды `commit` лишь текущим каталогом, использовав параметр `-l`.

Если содержимое файла не изменилось, то `commit` сообщает об этом и ничего не изменяет. Если `commit` обнаруживает, что необходимо выполнить команду `update`, то вам будет сообщено об этом, но не будет никакого автоматического вызова `update`. Предполагается, что вы лучше знаете, когда следует выполнять команду `update`.

Когда все нормально завершилось, вызывается редактор текста, чтобы вы могли ввести какие-то комментарии к данной операции `commit`. Комментарии будут переданы одной или нескольким протокольным программам, которые запишут эти комментарии в файл RCS внутри хранилища. Эти комментарии могут быть найдены позже с помощью команды `log`. Короткий комментарий может быть определен в командной строке с помощью параметра `-m message`, тогда редактор не будет вызываться. Если вы хотите записать относительно длинный комментарий и, одновременно, избежать вызова редактора текста, можно использовать параметр `-F file`. Комментарий будет взят из файла с именем `file`.

Параметры commit

Здесь описаны стандартные параметры, которые поддерживаются командой `commit`.

- `-l` — ограничить область действия текущим каталогом (без подкаталогов).
- `-n` — не выполнять никаких программ, которые определены в файле `modules`.
- `-R` — установить область действия команды `commit`: текущий рабочий (каталог) и все подкаталоги.
- `-r revision` — выполнить `commit` для версии `revision`. Значением `revision` должно быть одно из двух: ветвь или версия в основном стволе, которая находится выше, чем любой существующий номер ветви. Выполнить `commit` для отдельной версии в ветви невозможно.

Кроме вышеперечисленных, команда `commit` поддерживает следующие параметры.

- `-F file` — взять текст комментария из файла с именем `file` вместо вызова редактора текста.

- `-f` — заставляет систему CVS выполнить команду `commit`, образовав при этом новую версию исходных текстов, даже если вы не производили никаких изменений в файлах. Если текущая версия текстов есть 1.4, то следующие две команды эквивалентны:

```
cvs commit -f FILE
cvs commit -r 1.5 FILE
```

Параметр `-f` выключает рекурсию, то есть содержит параметр `-l`. Чтобы заставить `commit` обработать все файлы во всех подкаталогах, вам следует использовать два параметра `-f -R`.

- `-m message` — использовать строку `message` в качестве комментария вместо вызова редактора текста.

Примеры использования команды commit

Вы можете выполнить `commit` для версии в ветви (содержит четное число разделительных точек) с использованием параметра `-r`. Чтобы создать версию ветви, используйте параметр `-b` в командах `tag` или `rtag`. После этого вы сможете, используя команды `update` или `checkout`, привести в соответствие состояние вашего рабочего каталога и вновь образованной версии в ветви, не разрушив ваши исходные тексты в основном стволе разработки. Например, вам необходимо создать заплату (исправление) вашего продукта версии 1.2, хотя вовсю идет разработка варианта 2.0. Сделать это можно следующим образом:

```
cvs rtag -b -r FCS1_2 FCS1_2_Patch product_module
cvs checkout -r FCS1_2_Patch product_module
cd product_module
[[ hack away ]]
cvs commit
```

Это действует автоматически, поскольку параметр `-r` является липким.

Создание ветви после редактирования

Скажем, вы работали над экспериментальным программным обеспечением, которое базировалось на какой-то версии, которую вы взяли в рабочий каталог (выполнили `checkout`)

на прошлой неделе. Если другие разработчики тоже работают с теми же версиями исходных текстов, то чтобы не портить основной поток разработки, вы могли бы записать ваши экспериментальные варианты в отдельную новую ветвь. Другие разработчики смогут использовать ваши экспериментальные наработки и, одновременно, целиком использовать мощь CVS в отношении разрешения конфликтов. Сценарий мог бы быть таким:

```
[[ hacked sources are present ]]
$ cvs tag -b EXPR1
$ cvs update -r EXPR1
$ cvs commit
```

Команда `update` сделает параметр `-r EXPR1` липким для всех файлов. Заметим, что ваши изменения в файлах никогда не будут удалены командой `update`. Команда `commit` будет автоматически работать с верной ветвью, поскольку параметр `-r` — липкий. Вы также можете делать так:

```
[[ hacked sources are present ]]
$ cvs tag -b EXPR1
$ cvs commit -r EXPR1
```

но тогда только измененные вами файлы станут липкими. Если вы уберете изменения и выполните `commit` без указания параметра `-r EXPR1`, то некоторые файлы неожиданно для вас могут попасть в основной ствол.

Другие разработчики могут использовать ваши экспериментальные наработки, если выполнят команду:

```
cvs checkout -r EXPR1 whatever_module
```

update — синхронизировать рабочий каталог и хранилище

- Формат вызова:

  ```
  cvs update [-AdflPpR] [-d] [-r tag|-D] files ...
  ```

- Требуется: хранилище, рабочий каталог.
- Изменения: рабочий каталог.

После того как вы выполнили команду `checkout`, чтобы создать рабочий каталог, который содержит вашу персональную копию исходных текстов из общего хранилища, другие

разработчики продолжают работать над содержимым хранилища. Естественно, при этом меняются исходные тексты в общем хранилище, в том числе и те тексты, которые вы используете в своей работе. Это означает, что время от времени, когда вам это удобно, необходимо приводить содержимое вашего рабочего каталога в соответствие с состоянием общего хранилища.

Эта операция производится с помощью команды `update`.

Параметры команды update

Здесь приведены стандартные параметры команды update.

- `-D date` — использовать наиболее свежую версию, но не позже даты `date`. Этот параметр липкий и включает параметр `-P`.
- `-f` — полезно только вместе с параметрами `-D date` или `-r tag`. Если не найдено подходящих версий, то производится поиск наиболее свежих версий.
- `-k kflag` — обрабатывать ключевые слова RCS в соответствии с `kflag`. Этот параметр липкий: будущие изменения этого файла в этом рабочем каталоге будут использовать то же значение `kflag`. Команда `status` показывает установленные значения липких тегов. Например,

  ```
  cvs status CVS_cvscat.tex
  ```

  в ответ будет напечатано что-то похожее на следующее.

```
===================================================================
File: CVS_cvscat.tex    Status: Up-to-date

   Working revision:    1.1     Wed Feb 24 18:09:45 1999
   Repository revision: 1.1     /home/shevel/WorkCVS/
                                BOOK/CVS_cvscat.tex,v
   Sticky Tag:              MyCVS (revision: 1.1)
   Sticky Date:             (none)
   Sticky Options:          (none)
```

- `-l` — локально; выполнять только в текущем рабочем каталоге (без подкаталогов).
- `-P` — удалить пустые каталоги.

- `-p` — вывести файлы на стандартное устройство вывода, например, направить в следующий фильтр.
- `-R` — выполнять рекурсивно. Параметр задан по умолчанию.
- `-r tag` — найти версию `tag`. Этот параметр связан; предполагает использование `-P`.

Далее следуют специальные параметры, которые могут использоваться с командой `update`.

- `-A` — сбросить (reset) липкие значения: `tag`, `date` или значения, установленные параметром `-k`.
- `-d` — создать все каталоги, которые имеются в общем хранилище, но отсутствуют в вашем рабочем каталоге. Обычно `update` воздействует только на те каталоги и файлы, которые уже находятся в вашем рабочем каталоге.

  Такая возможность полезна для приведения вашей собственной копии рабочего материала (программ, описаний, прочего) в соответствие с общим хранилищем. Если вы намеренно избегаете копирования определенных каталогов из общего хранилища в ваш рабочий каталог посредством использования имени модуля или точным перечислением имен файлов и каталогов в командной строке команды `checkout`, то использование параметра `-d` в команде `update` приведет к копированию всех каталогов. Может оказаться, что копирование всех каталогов и файлов из хранилища совсем не входит в ваши планы.
- `-I name` — игнорировать файлы, если имя удовлетворяет строке `name` (в вашем рабочем каталоге) во время выполнения команды `update`. Вы можете использовать параметр `-I` несколько раз в одной команде, чтобы указать несколько файлов, которые вы хотели бы пропустить. Чтобы отменить игнорирование всех файлов, следует использовать `-I !`.
- `-Wspec` — определить имена файлов, которые (имена) должны быть отфильтрованы во время выполнения команды `update`. Этот параметр может применяться несколько раз в одной команде.

- `-jrevision` — если параметр `-j` используется однократно, то будет выполнено слияние предыдущей версии с версией, которая описывается параметром `-j`. Результат помещен в рабочий каталог. Термин *предыдущая версия* означает первого общего предка версии, которая находится в рабочем каталоге и версии, которая указана параметром `-j`.

  Два параметра `-j` означают, что следует выполнить слияние изменений в версии, описываемой первым параметром `-j`, с изменениями в версии, описываемой вторым параметром `-j`, а результат записать в рабочий каталог.

  В дополнение, каждый параметр `-j` может специфицировать дату, которая при использовании с *ветвями* может ограничить выбранные версии определенными датами. Дату можно задать в следующем виде:

  ```
  -jsymbolic_tag:date
  ```

  Как видно, дата (`date`) отделяется от тега (`symbolic_tag`) двоеточием.

Описание диагностики команды update

Команды `update` и `checkout` информируют вас о выполняемых действиях путем вывода сообщений (по одной строке на обработанный файл). В качестве первого символа в строке используются несколько букв для обозначения состояния файла:

- `U file` — файл в вашем рабочем каталоге приведен в соответствие с состоянием файла в хранилище. Это делается для всех файлов, которые уже имеются в вашем рабочем каталоге, но не в хранилище. Даже если вы не изменяли какой-то файл в вашем рабочем каталоге, но он был изменен в хранилище, то он будет также приведен в соответствие с состоянием в хранилище.
- `A file` — файл был добавлен к вашей рабочей копии исходных текстов. Он будет записан в хранилище, когда вы выполните команду `commit`.
- `R file` — файл был удален из вашего рабочего каталога (из вашей рабочей копии исходных текстов) и будет удален из хранилища, когда вы выполните команду `commit`.

- `M file` — файл был модифицирован в вашем рабочем каталоге. Обозначение `M` может означать одно из двух состояний, в котором находится файл, с которым вы работаете:
  - не было никаких модификаций этого файла в хранилище, а имеют место только те изменения, что вы сами внесли в вашу копию файла в вашем рабочем каталоге;
  - имели место как изменения в хранилище, так и изменения в вашем рабочем каталоге, однако эти обе версии изменений были объединены без конфликтов в вашем рабочем каталоге.

CVS будет выводить ряд сообщений, если она выполняет слияние вашей работы и делает резервную копию (backup) вашего рабочего файла, сохраняя тот вид, который он имел до выполнения команды `update`.

- `C file` — встречен конфликт (перекрывающиеся изменения) во время выполнения слияния. Файл с именем `file` в вашем рабочем каталоге будет представлять собой вывод команды `rcsmerge(1)`. Не модифицированная копия файла также находится в вашем рабочем каталоге и имеет имя `.#file.REVISION`, где `REVISION` есть версия в стандарте RCS, с которой вы начали модификации. Как разрешить конфликт, см. в разделе «Простой пример разрешения конфликта при объединении версий».
- `? file` — файл с именем `file` в вашем рабочем каталоге не соответствует ничему в хранилище и не находится в списке файлов, которые следует игнорировать в параметре `-I`. Иными словами, система CVS ничего о нем не знает.

Заметим, что не выводится никаких сообщений по поводу ложных каталогов, которые CVS просто игнорирует.

Примеры использования команды update

Нижеследующая строка является примером команды `update`, использованной для того чтобы посмотреть сразу все файлы в текущем рабочем каталоге без изменений.

```
cvs -n -q update
```

rtag — добавить символический тег

- Формат использования:

```
cvs rtag [-falnR] [-b] [-d] [-r tag | -Ddate]
symbolic_tag modules...
```

- Требования: хранилище.
- Изменения: хранилище.
- Синоним: `rfreeze`.

Вы можете использовать команду `rtag`, чтобы присвоить теги (описатели, ярлыки, имена) явно определенным версиям в хранилище. Команда `rtag` работает непосредственно с содержимым хранилища (не требует предварительного выполнения команды `checkout`). Для присваивания имен версиям файлов в вашем рабочем каталоге следует использовать команду `tag`, а не `rtag`. При этом команду `tag` можно выполнять лишь после команды `checkout`.

Если вы попытаетесь использовать имя тега, которое уже используется, CVS будет диагностировать такую попытку, но ничего не изменит. Если же вы хотите переместить имя тега (назвать этим же именем другую группу файлов), то чтобы изменения имели место, надо использовать параметр `-F`.

Параметры команды rtag

Ниже приведен стандартный набор параметров, который поддерживается командой `rtag`.

- `-D date` — присвоить тег наиболее свежей версии, но не позже чем дата `date`.

  `-f` — полезно только с параметрами `-D date` или `-r tag`. Если не найдено подходящих версий, то используется наиболее свежая версия.
- `-F` — переприсвоить существующий тег с тем же именем другой версии.
- `-l` — локально; выполнять лишь в текущем рабочем каталоге (без подкаталогов).
- `-n` — не запускать никакую программу, которая была определена параметром `-t` в файле `modules`.

- `-R` — обновить каталоги рекурсивно. Принято по умолчанию.
- `-r tag` — присвоить тег только тем файлам, которые содержат тег с именем `tag`. Этот параметр может быть использован, чтобы переименовать тег. Иными словами, присвоить символьное имя лишь тем именам файлов, которые были отмечены старым тегом `tag`, а затем удалить старый тег.

В дополнение к стандартным параметрам, имеются специфические параметры, которые понимает команда `rtag`:

- `-a` — этот параметр используется для того, чтобы `rtag` просмотрел файл `Attic` для нахождения удаленных файлов, которые содержат определенный тег. Тег удаляется из этих файлов, так что это позволяет продолжать использовать тег по мере продолжения разработки, то есть файлы удаляются из будущих поставок (distribution).
- `-b` — обозначить данным тегом новую ветвь в дереве версий.
- `[-d]` — удалить тег. В общем, теги (часто символьные имена различных вариантов поставок) не должны удаляться. Тем не менее, `-d` используется, чтобы удалить целиком старую версию или ошибочно помеченную версию.

tag — добавить тег в рабочем каталоге

- Формат использования:

```
tag [-lR] [-b] [-c] [-d] symbolic_tag [files...]
```

- Требуется: рабочий каталог, хранилище.
- Изменения: в хранилище.
- Синоним: `freeze`.

Эту команду используют, чтобы присвоить теги (имена, признаки) в хранилище наиболее свежим версиям ваших рабочих исходных текстов. Теги немедленно присваиваются файлам хранилища, так же как в случае команды `rtag`, однако версии определяются опосредованно, по записям CVS в вашей рабочей истории файлов (обычно, в рабочем каталоге файл `CVS/Entries`), а не явно.

Одно из применений тегов — запись списка текущих исходных файлов во время замораживания программ, то есть когда эти программы перестают менять. Поскольку ошибки ищутся в основном после даты заморозки, то лишь измененные исходные тексты, которые будут частью новой версии продукта, надо будет пометить заново.

Тегами обозначают, какие версии каких файлов были использованы в программной поставке (distribution). Команды `checkout` и `update` позволяют вам извлечь точную копию помеченного тегом набора программ в любое время в будущем, несмотря на то, что файлы могли быть модифицированы, добавлены или удалены с того момента, когда они были помечены тегом.

Кроме того, данная команда может быть использована, чтобы удалить тег или создать новую ветвь дерева версий.

Если вы попытаетесь использовать имя тега, который уже существует, то CVS выдаст диагностику, но ничего не изменит. Использованием параметра `-F` можно преодолеть данное правило.

Параметры команды tag

Ниже приведены стандартные параметры, которые поддерживаются командой `tag`.

- `-F` — присвоить существующий тег с тем же самым именем другой версии.
- `-l` — локально; выполнять только в текущем каталоге.
- `-R` — обрабатывать подкаталоги рекурсивно. Принято по умолчанию.

Имеется несколько дополнительных параметров.

- `-b` — создать тег ветви дерева версий. Это позволяет выполнять параллельную полностью изолированную разработку. Такая возможность очень удобна для создания заплат (patch) к ранее выпущенным версиям программного продукта.
- `-c` — проверить, что все файлы, которые будут помечены тегом, являются немодифицированными. Это может

оказаться полезным, гарантируя, что вы можете реконструировать текущее содержание файлов.

- `-d` — удалить тег. Если вы используете `cvs tag -d symbolic_tag`, то символьный тег с именем `symbolic_tag` будет удален.

ВНИМАНИЕ Удаление тега вызывает удаление без возврата части информации, которая позже может оказаться весьма важной.

release — освободить рабочий каталог

- Формат использования:

```
cvs release [-d] directories...
```

- Требуется: рабочий каталог.
- Изменения: рабочий каталог, журнал истории.

Команда release сообщает CVS, что вы намерены освободить рабочий каталог. Если рабочий каталог освобожден, то его можно удалить без потери информации. Команда `release` не выполняет никаких реальных удалений файлов или других данных, а только проверяет состояние вашего рабочего каталога. Иными словами, CVS по этой команде сообщит вам, может ли ваш рабочий каталог быть освобожденным без потери информации. Эта команда означает нормальное окончание цикла редактирования и изменения содержимого рабочего каталога. То есть, команда информирует CVS, что цикл работ, начатый командой

```
cvs checkout ...
```

завершен. Рекомендуется всегда выполнять команду `release` перед тем, как удалять рабочий каталог или надолго прерывать работу с ним. Очевидно, что если каталог не удалялся, то после перерыва в работе полезно выполнить команду `update`.

Команда `release` проверяет, есть ли в рабочем каталоге измененные файлы, которые не сохранены в хранилище. Предполагается, что команда `release` выполняется точно над вашим рабочим каталогом, то есть выше по дереву каталогов.

Параметры команды release

Команда `release` поддерживает один параметр `-d`. Использование параметра означает: удалить рабочую копию того файла, который успешно проверен (то есть копия в хранилище совпадает с рабочей копией). Если параметр `-d` не использован, то все файлы остаются в рабочем каталоге без изменений.

ВНИМАНИЕ Команда cvs release удаляет каталоги и файлы рекурсивно. Это имеет побочный эффект: если вы создали новый каталог внутри вашего рабочего каталога, но не добавили его к хранилищу посредством команды `add`, то он будет удален без диагностики, даже если каталог не пуст.

Вывод release

До того как команда `release` «освободит» ваши исходные тексты, она будет выводить сообщения для каждого имени файла, который изменен по сравнению с тем же файлом в хранилище.

- `U file` или `P file` — существует более свежая версия этого файла в хранилище, а вы не модифицировали локальную копию.
- `C file` — имя файла `file` — конфликтное.
- `A file` — файл с именем `file` был добавлен в вашем рабочем каталоге, но не сохранен в хранилище с помощью команды `commit`. Если теперь файл будет удален, то он будет потерян.
- `R file` — файл с именем `file` удален из вашего рабочего каталога, но не удален из хранилища, так как удаление не было подтверждено командой `commit`.
- `M file` — файл с именем `file` модифицирован в вашем рабочем каталоге.
- `? file` — файл с именем `file` находится в рабочем каталоге, но система CVS ничего о нем не знает. Если вы удалите ваш рабочий каталог, то этот файл будет потерян.

Примеры использования команды release

«Освободить» модуль и удалить ваш рабочий каталог можно следующей последовательностью действий:

```
$ cd ..  #
```

Вы должны стать точно над вашим рабочим каталогом

```
$ cvs release -d tc #
```

В ответ вы получите такие сообщения

```
You have [0] altered files in this repository.
Are you sure you want to release (and delete) module `tc': y
```

Ответив y, вы «освободите» файлы рабочего каталога, и он может быть удален.

Подстановка ключевых слов

Пока вы редактируете исходные тексты внутри вашей рабочей копии модуля, вы в любой момент сможете определить состояние файлов посредством команд `cvs status` и `cvs log`. Однако, вскоре после того, как вы экспортируете файлы из вашего окружения и настроек, вы обнаружите, что стало труднее определить, с какой версией модуля вы работаете.

Чтобы как-то помочь идентифицировать исходные тексты, CVS позволяет использовать механизм, который называется *постановка ключевых слов*. Строки вида `$KEYWORD$` и `$KEYWORD: ... $` внутри файла будут заменены строками вида `$KEYWORD:VALUE$`, когда вы получаете новую версию файла.

Список ключевых слов

Ниже приведен список ключевых слов.

- `$Author$` — имя пользователя (login), который ввел данную версию в хранилище.
- `$Date$` — дата и время, когда данная версия была внесена в хранилище.
- `$Header$` — стандартный заголовок, который содержит полное имя файла RCS, номер версии, дату, имя пользователя, состояние, кто закрыл файл (если он закрыт). Обычно файлы не будут закрыты, если вы используете CVS.

- `$Id$` — то же самое, что `$Header$` исключая то, что имя файла будет неполным.
- `$Name$` — имя тега, который использовался, чтобы взять этот файл (то есть для выполнения команды `checkout`).
- `$Locker$` — — имя пользователя (login), который закрыл файл.
- `$Log$` — комментарий, который использовался во время выполнения команды `commit`, которому предшествует стандартный заголовок. Существующие комментарии не замещаются. Вместо этого новый комментарий вставляется после строки:

```
% $L o g: CVS_Key.tex,v $}
```

Обратите внимание, в строке слово `Log` написано с разрядкой, чтобы CVS не пыталась подставить значение в данном месте. Каждая новая строка начинается с одной и той же последовательности, которой предшествует ключевое слово

```
%$Log
```

Например, если файл содержит:

```
%  $Log: Chap_06.tex,v $
%  Revision 1.5  2000/09/13 07:57:12  shevel
%  corr for eps
%
%  Revision 1.4  2000/09/12 18:59:57  shevel
%  corr
%
%  Revision 1.3  2000/09/12 16:20:05  shevel
%  eps
%
%  Revision 1.2  2000/09/11 12:25:25  shevel
%  renamed
%
%  Revision 1.3  2000/08/30 14:31:54  shevel
%  corr shortage
%
%  Revision 1.2  2000/08/29 15:58:30  shevel
%  corr
%
%  Revision 1.27  2000/03/22 12:56:54  shevel
%  corr
```

```
%
%  Revision 1.26  2000/02/16 15:40:33  shevel
%  WordCorr
%
%  Revision 1.25  2000/02/16 14:54:22  shevel
%  corr_table
%
%  Revision 1.24  1999/12/21 16:20:53  shevel
%  corr
%
%  Revision 1.23  1999/12/17 17:02:31  shevel
%  corr
%
%  Revision 1.22  1999/12/15 16:57:00  shevel
%  corr
%
%  Revision 1.21  1999/12/15 12:13:33  shevel
%  corr
%
%  Revision 1.20  1999/12/13 11:11:44  shevel
%  removed description
%
%  Revision 1.19  1999/12/12 18:25:40  shevel
%  corr
%
%  Revision 1.18  1999/12/12 09:43:42  shevel
%  corr
%
%  Revision 1.17  1999/12/01 16:04:27  shevel
%  corr
%
%  Revision 1.16  1999/12/01 13:33:55  shevel
%  corr
%
%  Revision 1.15  1999/11/30 21:35:25  shevel
%  corr
%
%  Revision 1.14  1999/11/30 16:43:17  shevel
%  corr
%
%  Revision 1.13  1999/11/29 16:47:03  shevel
%  corr
%
%  Revision 1.12  1999/11/28 22:45:16  shevel
%  corr
%
```

```
%  Revision 1.11  1999/11/14 10:49:05  shevel
%  corr
%
%  Revision 1.10  1999/11/11 17:16:28  shevel
%  Updated with adding the longtables.
%
%  Revision 1.9  1999/11/09 14:01:08  shevel
%  corrections
%
%  Revision 1.8  1999/11/05 15:48:19  shevel
%  corr
%
%  Revision 1.7  1999/11/04 16:24:29  shevel
%  corr
%
%  Revision 1.6  1999/11/03 16:04:40  shevel
%  Added figures and changed minor .
%
%  Revision 1.5  1999/11/03 14:13:52  shevel
%  all pict
%
%  Revision 1.4  1999/11/03 11:22:30  shevel
%  Pict 07 i 08
%
%  Revision 1.3  1999/11/03 07:37:18  shevel
%  corr
%
%  Revision 1.2  1999/11/01 20:28:54  shevel
%  Corr
%
%  Revision 1.1  1999/10/21 12:31:40  shevel
%  New
%
%  Revision 1.10  1999/05/18 09:29:41  shevel
%  Minor
%
%  Revision 1.9  1999/05/18 09:07:21  shevel
%  Minor
```

Дополнительные строки, которые добавляются, когда подставляется значение `$Log`, будут начинаться со знака процента. В противоположность прежним версиям CVS и RCS, начало комментария из файла RCS не используется. Ключевое слово `$Log` очень удобно, чтобы аккумулировать протокол изменений исходного текста. Однако в некоторых случаях это может оказаться непросто.

- `$RCSfile$` — имя файла RCS без имени пути (то есть не абсолютное имя).
- `$Revision$` — номер версии, присвоенный данному варианту.
- `$Source$` — абсолютное имя файла RCS.
- `$State$` — состояние, присвоенное данной версии. Состояние может быть присвоено командой:

```
$ cvs admin -s
```

Использование ключевых слов

Чтобы включить ключевое слово, вы просто включаете соответствующий текст. Например, вставляете строку `$Id$` в ваш файл и выполняете команду `commit`. Система CVS автоматически выполнит соответствующую подстановку как часть выполнения `commit`.

Обычно, включив строку `$Id$` в исходные файлы, вы получите в результате то, что подставленное значение будет попадать в сгенерированные файлы (готовые к исполнению двоичные коды). Например, если вам необходимо поддерживать исходные тексты компьютерных программ, то вы могли бы использовать какую-то переменную, которая содержала бы эту строку.

Команда `ident`, которая является частью RCS, можете быть использована, чтобы извлечь ключевые слова и их значение из файла. Это безусловно удобно для текстовых файлов, но еще важнее для двоичных файлов.

```
$ ident samp.c
samp.c:
   $Id: Chap_06.tex,v 1.5 2000/09/13 07:57:12 shevel Exp $
$ gcc samp.c
$ ident a.out
a.out:
   $Id: Chap_06.tex,v 1.5 2000/09/13 07:57:12 shevel Exp $
```

Существует другая система поддержки версий SCCS, которая имеет команду `what`. Команда `what` очень похожа на команду `ident` и используется для тех же целей. На многих компьютерах нет системы RCS, но есть система SCCS. Поскольку `what` ищет символы `@(#)`, то легко использовать ключевые слова, которые распознаются обеими системами.

Просто используйте ключевые слова RCS вместе с магическими фразами SCCS, как, например, показано ниже:

```
static char *id="\@(\#) $Id: Chap_06.tex,v 1.5 2000/09/
13 07:57:12 shevel Exp $"
```

Обход подстановок

Механизм подстановки ключевых слов не всегда удобен. Так, если вы желаете, чтобы в тексте появилась строка, литерально совпадающая с `$Author$`, без того чтобы CVS заменила ее на что-то вроде `$Author: shevel $`. К сожалению нет возможности селективно отключить часть подстановок. Вы можете отключить только все подстановки сразу, используя параметр `-ko`.

Во многих случаях вы можете избежать использования ключевого слова в исходном тексте, даже если оно должно появиться в финальном варианте текста. Например, исходный вид ключевого слова `$Author$` в данном тексте таков: `\$Author\$`.

Режимы подстановки

Для каждого файла имеется режим подстановки по умолчанию. То же самое верно и для копии файла в рабочем каталоге. Первый (параметр `-kkv`) устанавливается посредством параметра `-k` в командах `cvs add` и `cvs admin`. Последний (параметр `-kv`) устанавливается посредством параметров `-k` или `-A` в командах `cvs checkout` или `cvs update`. Команда `cvs diff` также имеет команду `-k`.

Имеются следующие режимы подстановки:

- `-kkv` — генерировать подстановку с использованием форм по умолчанию, то есть `$Revision: 1.5 $` для ключевого слова `Revision`.
- `-kkvl` — почти то же, что `-kkv`, но будет добавляться имя того, кто закрыл файл, если файл закрыт. Обычно такой режим не используется с CVS.
- `-kk` — генерировать только имена ключевых слов, опуская их значения. Например, для ключевого слова `Revision` будет сгенерировано `$Revision$` вместо `$Revision: 1.5 $`. Эта возможность полезна, когда необходимо

сравнить две версии одного файла, поскольку это позволит исключить влияние различия в значениях ключевых слов.

- `-ko` — генерировать старую ключевую строку, представленную в рабочем файле непосредственно перед помещением файла в хранилище. Например, для ключевого слова `Revision` будет сгенерировано `$Revision: 1.1 $` вместо `$Revision: 1.5 $`, так как будет использовано значение, при котором файл был помещен в хранилище.
- `-kb` — то же, что `-ko`, но для некоторых операционных систем (не типа Unix/Linux) может отличаться от `-ko`.
- `-kv` — генерировать только значения ключевых слов. Например, для ключевого слова `Revision` генерируется строка `1.9` вместо `$Revision: 1.9$`. Следует помнить, что этот параметр не может быть корректно использован при экспорте двоичных файлов.

Переменные окружения, которые использует система CVS

В табл. 5.3 приведен список переменных окружения, которые оказывают влияние на функционирование системы CVS.

Таблица 5.3. Переменные окружения

Переменная	Значение
$CVSIGNORE	Список разделенных пробелами шаблонов имен файлов, которые CVS должна игнорировать
$CVSWRAPPERS	Список разделенных пробелами шаблонов имен файлов, которые CVS должна рассматривать в качестве фильтров (wrappers)
$CVSREAD	Если эта переменная установлена, то команды checkout и update попробуют записать файлы в ваш рабочий каталог с правом доступа только чтение. Если переменная $CVSREAD не установлена, то по умолчанию файлы будут записаны с правами доступа чтение и запись
$CVSUMASK	Права доступа по умолчанию
$CVSROOT	Эта переменная должна содержать абсолютное имя каталога, в котором находится хранилище CVS. Если переменная не установлена, то вы должны во всех командах указывать это имя как значение параметра -d. Если использован параметр -d, то значение аргумента используется в качестве абсолютного имени каталога, где находится хранилище, даже если установлено значение переменной $CVSROOT

Переменная	Значение
$EDITOR, $CVSEDITOR	Определяют программу редактора, которая вызывается во время работы команды commit. Переменная $CVSEDITOR главнее, чем переменная $EDITOR
$PATH	Если переменная $RCSBIN не установлена, то используется переменная $PATH
$HOME, $HOMEPATH, $HOMEDRIVE	Эти переменные используются, чтобы найти каталог, в котором находится файл .cvsrc. В ОС UNIX система CVS использует только HOME
$CVS_RSH	Эта переменная определяет имя программы, которая используется для соединения с удаленным сервером, если использован метод доступа :ext:
$CVS_SERVER	Используется в режиме клиент-сервер, когда требуется получить доступ к удаленному хранилищу с использованием rsh. Значение по умолчанию есть cvs
$CVS_PASSFILE	Используется в режиме клиент-сервер во время доступа с использованием cvs login server. Значение по умолчанию есть $HOME/.cvspass
$CVS_CLIENT_PORT &	Используется в режиме клиент-сервер, когда доступ производится с использованием системы аутентификации Kerberos
$CVS_RCMD_PORT	Используется в режиме клиент-сервер. Если она установлена, то используется как номер порта в демоне RCMD на стороне сервера
$CVS_CLIENT_LOG	Используется для отладки только в режиме клиент-сервер. Если переменная установлена, то все, что посылается на сервер, записывается в файл $CVS_CLIENT_LOG.in, а все, что посылается клиенту, записывается в файл $CVS_CLIENT_LOG.out
$CVS_SERVER_SLEEP	Используется для отладки режима клиент-сервер. Если установлена, то воспринимается как число секунд, на которые должен быть задержан запуск процессов на серверной стороне, чтобы вы успели запустить отладчик
$CVS_IGNORE_ REMOTE_ROOT	Для CVS 1.10 и более старых версий установка данной переменной предотвращает изменение файла CVS/Root, когда используется глобальный параметр -d. В более поздних версиях не планируется изменять файл CVS/Root, таким образом, значение данной переменной не будет оказывать никакого влияния
$TMPDIR, $TMP, $TEMP	Каталоги, в которых CVS располагает временные файлы

Алфавитный указатель

A

B

C

D

E

F

M

N

O

P

Q

R

S

T

U

V

W

Y

Z

Б

В

Г

Д

З

И

К

Л

М

У

Ф

Х

Ч

Ш

Я

Ан[illegible]ль

LINUX ОБРАБОТКА ТЕКСТОВ

Специальный справочник

С использованием операционной платформы Linux можно выполнять любую работу, однако в этой книге рассказывается в основном о работе с текстами: описаниями, исходными текстами программ на различных языках, письмами, романами, руководствами, рефератами, наконец, просто заметками.

Этой теме в литературе обычно не уделяется должного внимания, в то время как в любой деятельности, связанной с компьютером, практически всегда имеет место работа с каким-либо текстом.

Уровень пользователя

Начинающий Опытный Профессионал

Категория книги: Справочник
Операционные системы

ISBN 5-272-00039-0

Информация, которую вы найдете в справочнике:

- оболочки, команды и файловая система Linux
- редакторы vi, vim, emacs и xemacs
- потоковый редактор sed
- поиск по шаблону и регулярные выражения
- подсистема печати a2ps
- подсистема анализа и обработки текстов awk
- система контроля версий CVS

BEST!

На каждый вопрос — готов ответ!

ПИТЕР®
WWW.PITER.COM