Войтов Н. М.

Основы работы с Linux

Учебный курс

ДМК Пресс
издательство
softline®
Academy Alliance
Москва, 2011

УДК 32.973.26-018.2
ББК 004.4
В65

Войтов Н. М.

В65 Основы работы с Linux. Учебный курс. – М.: ДМК Пресс, 2010. – 216 с.: ил.

ISBN 978-5-94074-148-0

Эта книга знакомит читателей с основами работы в операционной системе Linux на примере Red Hat Enterprise Linux 5 (RHEL). Она предназначена для людей, которые только начинают осваивать систему Linux. Книга состоит из теоретической и практической частей, которые вместе позволяют получить систематизированные знания о системе и умения решать практические задачи. Основной упор сделан на полноту раскрытия темы, при сохранении лаконичности изложения. Теоретическая часть раскрывает принципы работы системы, нюансы настройки различных компонентов и позволяет подготовиться к экзамену RHCT, комплексной программы сертификации Red Hat. При создании практической части, было уделено внимание сбалансированности практических заданий. Задания довольно разнообразны – от простых для «новичков», с подробным описанием всех шагов, до более сложных, с возможностью самостоятельного выполнения различными способами для людей, обладающих представлением и опытом работы с конкретной технологией.

Курс состоит из 14-ти модулей, последовательно раскрывающих основы работы с ОС Linux и знакомящих читателей:

- с графическим и командным интерфейсами;
- с файловой системой и командами;
- с обработкой текста и написанием сценариев командного интерпретатора;
- с работой с учетными записями и процессами;
- с основами работы с сетевыми приложениями.

Данное пособие – первое в серии книг по Linux, издаваемых совместно издательством ДМК Пресс и Softline Academy Alliance.

Курс разработан в Учебном центре ВМК МГУ & Softline Academy (www.it-university.ru) при участии специалистов, которые не один год работают в области администрирования систем на базе Linux. Кроме того, книга базируется на 6-летнем опыте проведения курсов и построена таким образом, что читатель получает и закрепляет практические навыки для администрирования ОС Linux и получает базу для самостоятельного решения сложных проблем, за счет понимания принципов функционирования системы.

УДК 32.973.26-018.2
ББК 004.4

Материал, изложенный в данной книге, многократно проверен. Но поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-5-94074-148-0

Содержание

Введение

Данное пособие – первое в серии книг по Linux, издаваемых совместно ДМК Пресс (www.dmk-press.ru) и Softline Academy Alliance (www.it-academy.ru). «Основы работы с ОС Red Hat Enterprise Linux» – начальный курс в изучении ОС Linux. Курс содержит материал, который позволяет пользователям, начинающим работать с этой ОС, понять и закрепить принципы работы с данной системой. В качестве основного рассматриваемого в данном курсе дистрибутива используется дистрибутив ОС Red Hat Enterprise Linux 5 (RHEL)[1], являющийся коммерческим решением компании Red Hat. Знания, полученные в ходе изучения данного курса, применимы к любому другому дистрибутиву ОС Linux.

Курс предназначен для системных администраторов и инженеров, а также пользователей любого уровня знаний, стремящихся освоить ОС Linux и в дальнейшем успешно сдать сертификационные экзамены по программам Red Hat Certified Technician (RHCT) и Red Hat Certified Engineer (RHCE).

Предлагаемый вашему вниманию конспект лекций и практические работы — это основной учебный материал для проведения занятий по курсу «Основы работы с ОС Red Hat Enterprise Linux» в Учебных центрах Softline Academy (www.it-academy.ru). Эти учебные центры создаются в рамках инициативы Softline Academy Alliance, цель которой объединить учебные заведения и организации, заинтересованные в качественной и эффективной подготовке студентов и молодых специалистов для работы в области IT.

Курс разработан преподавателями Учебного центра ВМК МГУ & Softline Academy (www.it-university.ru), который является первым в России авторизованным учебным центром программы Microsoft IT Academy и первой Академией Softline.

Курс рассчитан на 32 академических часа и может быть освоен как самостоятельно, так и под руководством опытного преподавателя в любой из двадцати пяти Академий Softline, находящихся в восемнадцати регионах России.

Курс состоит из 14 модулей, последовательно раскрывающих основы работы с ОС Linux. Все модули условно разделены на пять групп:

- Основы работы с графическим и командным интерфейсами;
- Основы работы с файловой системой и командами;
- Основы обработки текста и написание сценариев командного интерпретатора;
- Основы работы с учетными записями и процессами;
- Основы работы с сетевыми приложениями.

1-й и 2-й модули познакомят вас с особенностями и характеристиками ОС Linux. В этих модулях даются основные понятия и определения, на основе которых строится весь последующий материал.

3-й, 4-й и 5-й модули рассказывают о принципах работы с файловой системой и командами. Данные модули являются основными для успешного понимания последующего материала.

---

[1] Здесь и далее по тексту под «ОС Linux» будет пониматься ОС Red Hat Enterprise Linux 5.

6-й, 7-й и 8-й модули посвящены работе с текстом. Так как большинство настроек системы и ее приложений находится в текстовых файлах, очень важно научиться эффективно работать с текстовыми данными.

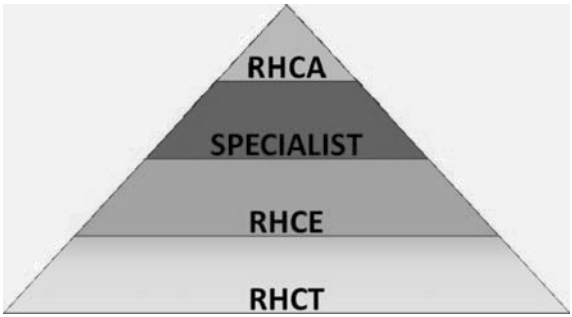
В последующих тёх модулях рассматриваются вопросы разделения прав пользователей системы, управления пользователями и процессами. Данные модули дадут четкое понимание модели управления пользователями, правами и пользовательскими процессами, что особенно важно для понимания взаимодействия различных служб системы между собой.

Заключительные три модуля помогут вам получить более углубленные знания по работе с командным интерпретатором и файловой системой, а также затронут вопросы взаимодействия ОС Linux с сетью передачи данных и сетевыми сервисами.

После завершения обучения данному курсу вы будете обладать необходимыми знаниями, и уметь:

- описывать особенности ОС Linux и понимать принципы работы системы;
- уверенно работать с файловой системой и моделью доступа к данным;
- пользоваться встроенной справочной системой;
- эффективно работать в командном интерпретаторе bash в интерактивном режиме;
- разрабатывать собственные сценарии командного интерпретатора bash, а также интерпретаторов sed и awk;
- редактировать текстовые файлы, обрабатывать массивы текстовых данных и использовать регулярные выражения;
- администрировать учетные записи пользователей и групп;
- понимать идеологию работы с процессами и многозадачности;

Предлагаемый курс RH-033 является базовым курсом, предназначенным для поготовки к сдаче экзаменов RHCT комплексной программы сертификации Red Hat, структура которой представлена на следующей диаграмме.



Аудитория, на которую ориентирована сертификация RHCT, – начинающие пользователи ОС Linux, а также пользователи, переходящие с других ОС. Полученный сертификат RHCT доказывает наличие навыков, необходимых для установки, настройки и включения рабочих станций Red Hat Enterprise Linux Desktop

в существующую сетевую инфраструктуру. Экзамен RHCT (RH202) длится 2 часа и содержит в себе как теоретические, так и практические задания.

Сертификация RHCE – стандартное требование[1] для большинства администраторов, системных инженеров и консультантов. В ней рассматриваются сетевые сервисы ОС Linux, вопросы их функционирования и безопасности. Экзамен RHCE (RH302) длится 3,5 часа и содержит в себе как теоретические, так и практические задания.

Заключительным шагом в развитии сертификации RHCE является специализация в одной из следующих областей:

- кластеры и управление хранилищами данных;
- администрирование политик SELinux;
- развертывание, виртуализация и управление системами;
- мониторинг и настройка производительности;
- корпоративные сервисы каталогов и аутентификации;
- безопасность сетевых сервисов.

В итоге специалист может подтвердить свою квалификацию, сдав один из экзаменов по указанным выше областям.

Сертификация SPECIALIST (Red Hat Certified Specialist) – следующий уровень комплексной программы сертификации Red Hat. На данном уровне существуют две сертификации – специалист в области безопасности (RHCSS) и специалист центра обработки данных (RHCDS). Для того чтобы стать специалистом, необходимо наличие сертификата RHCE и трех сданных экзаменов по темам RHCSS (EX333, EX423, EX429) или RHCDS (EX401, EX423, EX436).

Заключительный уровень RHCA является самой старшей ступенью сертификаций Red Hat. Она предназначена для специалистов, определяющих стратегию развития ИТ-инфраструктуры организации. Типичные функции RHCA – планирование, разработка и управление системами на основе ОС Linux. Для того чтобы получить сертификацию RHCA, необходимо сдать пять дополнительных экзаменов, каждый из которых длится от 2 до 8 часов в зависимости от темы экзамена.

Помимо рассмотренных выше программ сертификаций, существуют дополнительные программы, такие как Red Hat Certified Virtualization Administrator (RHCVA) и JBoss Certified Applications Administrator (JBCAA). Программа RHCVA[2] предназначена для администраторов, занимающихся администрированием и поддержкой виртуальных сред на основе ОС Red Hat Enterprise Linux. Программа JBCAA предназначена для администраторов платформы JBoss Application Platform[3].

---

[1] Общие требования к знаниям кандидатов, сертифицирующихся по программам RHCT и RHCE, представлены здесь: http://www.redhat.com/certification/rhce/prep_guide/

[2] Информация о платформе виртуализации Red Hat: http://www.redhat.com/virtualization/ rhev/

[3] Информация о платформе JBoss: http://www.jboss.com/products/platforms/application/

Условные обозначения

В данном пособии применяются следующие условные обозначения.

Имена файлов и папок начинаются со строчных букв (при работе в командной строке или графической оболочке регистр букв всегда имеет значение).

Аббревиатуры напечатаны ПРОПИСНЫМИ БУКВАМИ.

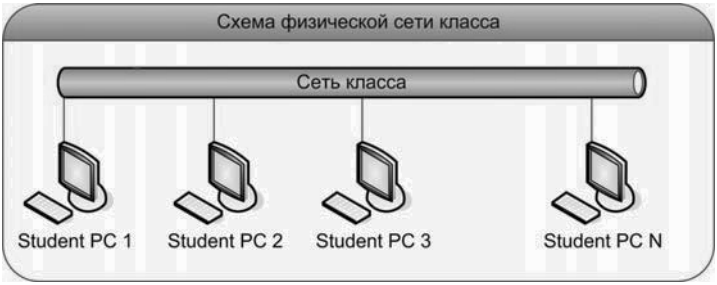
Листинги кода, примеры команд, а также текста, выводимого на экран, выделены `данным шрифтом`, причем ввод команды выделен **`жирным шрифтом`**.

Необязательные аргументы команд заключены в квадратные скобки (например: данный [`аргумент`] является необязательным).

Обязательные аргументы команд записываются без квадратных скобок.

Ключевые термины выделены ***полужирным курсивом***.

Схема класса и виртуальные машины



Класс, в котором выполняются практические работы, состоит из физических компьютеров, объединенных в локальную сеть с адресом 192.168.1.0/24 и имеющих динамическую IP-адресацию (DHCP). Все практические работы по данному курсу выполняются на виртуальных машинах, работающих под управлением ПО VMware Player.

Учетные данные для регистрации на физическом компьютере слушателя (Host PC):

Имя пользователя: _____
Пароль: _____
Домен (если есть): _____

В качестве операционной системы ***хостовой машины*** (физической рабочей станции слушателя, на которой работает ПО VMware Player) используется ОС Microsoft Windows XP. В качестве операционной системы ***гостевых машин***, в которых непосредственно выполняются все практические работы, используется ОС Red Hat Enterprise Linux 5 (RHEL5).

У каждого слушателя имеется одна виртуальная машина rhel5.linux.lab. Данная виртуальная машина имеет один сетевой интерфейс (eth0), используемый для связи между виртуальными машинами и выхода в физическую сеть класса. Все практические работы выполняются на виртуальной машине rhel5.linux.lab.

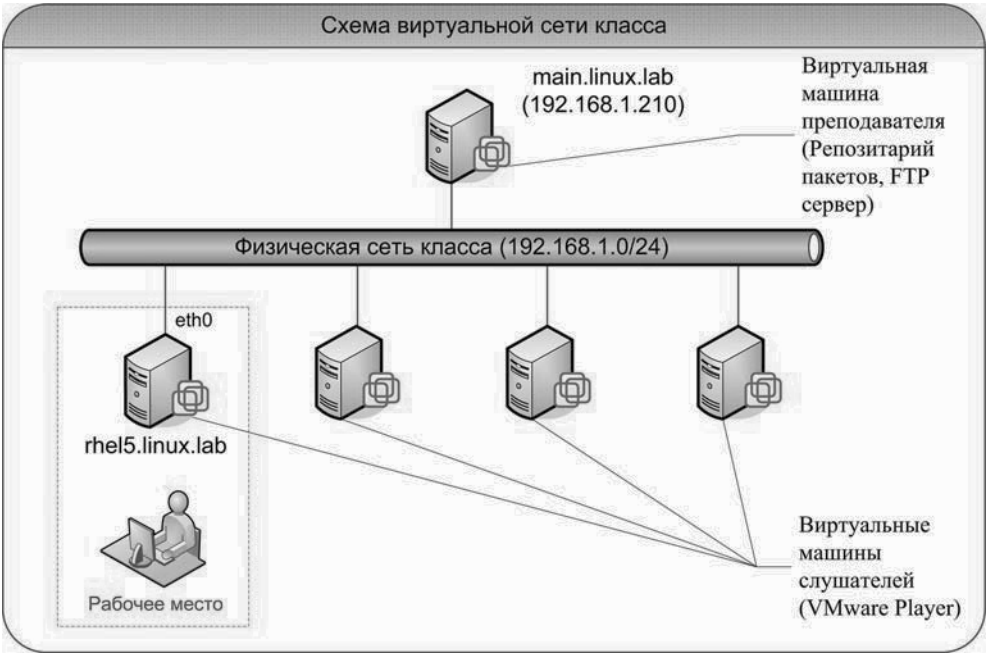


Схема виртуальной сети класса
main.linux.lab
(192.168.1.210)
Виртуальная машина преподавателя (Репозитарий пакетов, FTP сервер)
Физическая сеть класса (192.168.1.0/24)
eth0
rhel5.linux.lab
Рабочее место
Виртуальные машины слушателей (VMware Player)

Модуль 1. Обзор ОС Linux

После завершения изучения данного модуля вы научитесь:

- понимать идеологию открытого программного обеспечения;
- разбираться в дистрибутивах ОС Linux;
- описывать общую структуру ОС Linux и ее характеристики.

1.1. Открытое программное обеспечение

Прежде чем приступить к изучению ОС Linux, необходимо четко представлять идеологию самой системы, а точнее программного обеспечения, из которого она состоит. Под открытым программным обеспечением (далее Open Source[1]) подразумевается не только свободный доступ к исходному коду программ, но и способы распространения программ и принципы их разработки. Существуют четыре основных правила, которому должна удовлетворять любая программа, относящаяся к Open Source:

1. Предоставлять пользователю возможность использования программы вне зависимости от его цели.
2. Предоставлять пользователю возможность изучать принципы и алгоритмы работы программы, в том числе посредством доступа к исходному коду.
3. Не накладывать ограничений на свободное распространение программы.
4. Не накладывать ограничений на возможности изменения исходного кода программы и дальнейшего распространения.

Начало открытому программному обеспечению было положено в 1983 году разработчиком и инициатором идей открытого программного обеспечения Ричардом Столлменом, основателем первого open source проекта GNU (GNU's Not Unix). Этим же человеком была основана некоммерческая организация «Фонд свободного программного обеспечения» (Free Software Foundation), в задачи которой входила поддержка движения свободного программного обеспечения и в особенности проекта GNU.

Проект GNU, по сути, представляет собой набор отдельных программ (например, таких как редактор текста, компилятор, калькулятор и т. д.), выполняющих определенный функционал. Это еще не операционная система, а лишь программное обеспечение. Данное программное обеспечение и любое другое программное обеспечение, относящееся к open source, распространяется и лицензируется со-

[1] Более подробную информацию о сообществе Open Source можно узнать по адресу http://www.opensource.org.

гласно лицензии GPL (General Public License) и ее нескольким вариантам. Цель лицензии GPL заключается в предоставлении пользователю права копировать, изменять и распространять (в том числе и на коммерческой основе) программы, а также гарантировать, что и пользователи всех производных программ получат вышеперечисленные права.

В 1991 году финский программист Линус Торвальдс создал ядро операционной системы GNU/Linux. Таким образом появилась операционная система с ядром Linux и набором программ и библиотек, разработанных в рамках проекта GNU.

В отличие от большинства других операционных систем, система GNU/Linux не имеет единой «официальной» комплектации. Вместо этого GNU/Linux поставляется в большом количестве так называемых дистрибутивов, в которых программы GNU соединяются с ядром Linux и другими программами. Наиболее известными дистрибутивами GNU/Linux являются Slackware, Debian GNU/Linux, Red Hat, Fedora, Ubuntu. Некоторые из данных дистрибутивов, например Red Hat Linux, относятся к коммерческим программным продуктам.

1.2. Общая характеристика дистрибутивов ОС Linux

Под дистрибутивом понимается форма распространения программного обеспечения. Любой дистрибутив ОС, в том числе и GNU/Linux, можно выделить среди других дистрибутивов на основании его основных характеристик, среди которых содержатся:

- **политика дистрибутива.** По данной характеристике можно судить о том, каким образом в дистрибутив включается программное обеспечение, какие существуют требования к размещению файлов пакетов на файловой системе, какова периодичность обновления дистрибутива и т. д.;
- **программа-загрузчик.** Отвечает за инициализацию ядра операционной системы, начальных настроек аппаратного обеспечения;
- **используемое ядро.** Ключевой компонент дистрибутива, от которого зависит дальнейшая работоспособность системы. В каждом дистрибутиве содержится измененная версия оригинального[1] ядра Linux, что является следствием различных политик дистрибутивов и их программного окружения. Версия ядра дистрибутива зависит от типа дистрибутива (стабильный, тестовый и т. п.). У стабильных дистрибутивов версия ядра Linux не содержит самых последних обновлений драйверов и новейших возможностей. Поэтому при необходимости использования какого-нибудь нового функционала может понадобиться установка обновленной версии ядра;

[1] Оригинальное ядро (vanilla kernel) разрабатывается сообществом независимых разработчиков по всему миру. Загрузить самую последнюю версию ядра можно с сайта http://www.kernel.org/. Список изменений, внесенных в ту или иную версию ядра (или отдельных его компонентов), доступен на сайте http://git.kernel.org/.

- **программа управления пакетами.** В разных дистрибутивах используются разные программы управления пакетами, что влияет на способ установки, удаления и обновления программного обеспечения;
- **наборы пакетов.** Данная характеристика отражает специализированность дистрибутива, его ориентированность на решение конкретных задач – кластерные дистрибутивы, дистрибутивы для специфических областей науки и т. д.;
- **лицензирование.** Определяет политику использования программного обеспечения дистрибутива;
- **разработка.** Технические, административные, финансовые и другие решения, положенные в основу дистрибутива, наличие поддержки пользователей.

В настоящее время существуют три основных типа дистрибутивов ОС Linux, различающихся с точки зрения системы управления пакетами программ:

- **дистрибутивы, основанные на Debian GNU/Linux или использующие формат пакетов DEB.** Дистрибутивы данного типа отличаются наличием самого большого количества пакетов и поддержкой большинства существующих аппаратных архитектур;
- **дистрибутивы, основанные на Red Hat Linux или использующие формат пакетов RPM.** Некоторые из дистрибутивов данного типа, например Red Hat Enterprise Linux, являются коммерческими. Помимо возможностей, присутствующих в некоммерческих дистрибутивах, они предоставляют пользователям расширенную техническую поддержку, а также дополнительные виды сервисов, таких как RHN[1], облегчающих процесс администрирования, поддержки и управления множеством систем;
- **дистрибутивы, основанные на Slackware Linux.** Дистрибутивы Slackware Slackware основаны на принципе KISS (*Keep It Simple, Stupid*, или *Keep It Short and Simple* – процесс и принцип проектирования, при котором простота системы декларируется в качестве одной из основных целей или ценностей). Кроме того, менеджер пакетов не отслеживает зависимости в силу их очень незначительного количества.

В основе любого дистрибутива ОС Linux лежит ядро Linux. Поэтому основные системные характеристики ОС Linux диктуются особенностями ядра. ***Ядро*** – это ключевой компонент любой ОС, который обеспечивает взаимодействие пользовательских программ с аппаратурой компьютера, распределение времени между процессами, благодаря которому достигается многозадачность, и другие возможности системы.

В общем случае ОС Linux является полноценной многозадачной многопользовательской операционной системой с поддержкой почти всех имеющихся на сегодняшний день аппаратных архитектур процессоров, среди которых Intel x86,

[1] Более подробно о сервисе RHN можно узнать по адресу https://www.redhat.com/f/pdf/rhn/rhn101698_0705.pdf

AMD64, SPARC, IBM Power, IA64 и др. Поддержка сетевых соединений является одной из наиболее сильных сторон ОС Linux как в отношении поддерживаемых функций, так и в отношении производительности.

Изначально пользователю ОС Linux доступны следующие преимущества данной системы:

- **многозадачность.** Реальный приоритетный многозадачный режим дает возможность ядру ОС эффективно выполнять несколько программ одновременно, что крайне важно для конкурентной работы большого количества служб;
- **многоплатформенность.** Почти все имеющиеся на сегодняшний день аппаратные платформы поддерживаются ОС Linux;
- **мультисистемное взаимодействие.** ОС Linux хорошо взаимодействует с другими ОС посредством большинства используемых в настоящее время сетевых протоколов семейства TCP/IP и открытых стандартов (POSIX, LSB и прочее);
- **масштабируемость и переносимость.** Большая часть ОС Linux написана на языке C, что делает данную систему работоспособной и масштабируемой на любом оборудовании, включая мобильные устройства;
- **гибкость использования.** Пользователю дана возможность изначально определить использование ОС Linux под конкретную задачу и подобрать необходимое для решения данной задачи ПО;
- **надежность и производительность.** Ввиду особенностей ядра ОС Linux система обладает повышенной устойчивостью к программным сбоям и способна достаточно быстро обрабатывать большие объемы информации. Все системные процессы работы ОС доступны для просмотра и дальнейшего анализа (изучения).

1.3. Дистрибутив Red Hat Enterprise Linux

В настоящем пособии рассматривается дистрибутив Red Hat Enterprise Linux Server 5. Для освоения материала, изложенного в пособии, совсем не обязательно использовать именно этот дистрибутив, так как основная задача слушателя заключается в понимании основ работы с ОС Linux. Принципы, изложенные в данном пособии, применимы к любому дистрибутиву ОС Linux.

Компания Red Hat разделила разработку и развитие ОС Linux на два направления: Red Hat Enterprise Linux и Fedora Project. Red Hat Enterprise Linux лицензируется согласно лицензии GPL и предоставляет пользователю программные продукты промышленного уровня с двухгодичной периодичностью выпуска нового релиза. Fedora Project является Open Source проектом, предоставляющим пользователю наиболее свежие версии программных продуктов и выпускающим новый релиз с полугодичной периодичностью. Таким образом, коммерческий дистрибутив, помимо дополнительных коммерческих приложений и сервисов, обладает большей стабильностью и надежностью, чем ОС Fedora Linux. Несмотря на то что коммерческий дистрибутив Red Hat Enterprise Linux содержит в своем со-

ставе закрытые приложения, например такие, как кластерная система GFS, компания Red Hat предоставляет исходный код данных коммерческих продуктов в открытом доступе согласно GPL.

Если вы не хотите отказываться от надежности коммерческого дистрибутива и в то же время не готовы приобрести коммерческую лицензию, существует альтернативный проект – CentOS (Community Enterprise Operating System). Используя лицензионное соглашение GPL и общедоступный исходный код коммерческих приложений Red Hat, сообщество CentOS формирует свой собственный независимый от компании Red Hat дистрибутив, сочетающий в себе стабильность Red Hat Enterprise Linux и наличие коммерческих продуктов.

Дистрибутив Red Hat Enterprise Linux выпускается в двух вариантах: серверном (Red Hat Enterprise Linux Server) и персональном (Red Hat Enterprise Linux Desktop).

Системные требования к данным дистрибутивам существенно зависят от планируемой нагрузки и количества устанавливаемых пакетов. Можно сформулировать следующие минимальные системные требования к оборудованию на основе 32-битных процессоров Intel:

- процессор: Intel x86 200 МГц (текстовый интерфейс управления) или 400 МГц (графический интерфейс управления);
- память: 64 Мб (текстовый интерфейс управления) и 192 Мб (графический интерфейс управления);
- дисковое пространство: 700 Мб (текстовый интерфейс управления) или 3072 Мб (графический интерфейс управления);
- сетевой интерфейс (в случае необходимости работы с сетью);
- графический дисплей стандарта VGA.

Модуль 2. Знакомство с пользовательским интерфейсом

После завершения изучения данного модуля вы научитесь:

- выполнять базовые операции по управлению системой;
- работать в графической системе X Window;
- использовать виртуальные консоли.

2.1. Текстовый и графический режимы работы

Сегодня ОС Linux предоставляет конечным пользователям и администраторам возможность работы с системой не только в консольном (текстовом), но и в графическом режиме, используя графический интерфейс пользователя (GUI). Большинство, если не все, административных задач можно выполнять в консольном режиме работы, поэтому установка графической оболочки в случае использования ОС Linux в качестве сервера не имеет большого смысла из-за траты времени и ресурсов оборудования. Однако знание и понимание работы графической среды ОС Linux полезны для администрирования конечных пользователей, использующих ОС Linux в качестве клиента.

2.1.1. Графический интерфейс пользователя (GUI)

Изначально графическая система ОС Linux получила название **X Window System** (X). Впоследствии эта система регулярно изменялась, начиная от версии **XFree86** и заканчивая текущей версией **X.Org**. Архитектура графической системы ОС Linux состоит из следующих основных компонентов:

- **X-сервер** (X server) – ядро графической системы ОС Linux. X-сервер отвечает за прорисовку изображений окон и других графических объектов, управляет работой мыши и клавиатуры. Помимо отображения графической среды на локальном дисплее, X-сервер обслуживает подключение с удаленных хостов, а также все обращения к графическому оборудованию;
- **Диспетчер дисплеев** (display manager). Основной его задачей являются аутентификация пользователей и запуск X-сервера. По умолчанию в ОС Linux используется диспетчер дисплеев **GNOME Desktop Manager** (gdm)[1];
- **Графическое окружение** (desktop environment) является связующим звеном между диспетчером окон и конечным пользователем. Оно содержит

[1] В случае использования графического окружения **KDE** используется диспетчер **kdm**.

средства настройки отображения рабочего стола и различные программы, предназначенные для работы в графическом режиме. В ОС Linux имеются два графических окружения: GNOME и KDE, первое из которых используется по умолчанию при установке ОС Linux;

- **X-клиент** (X-client). Под X-клиентом принято понимать программу, общающуюся с X-сервером и посылающую ему запросы, необходимые для работы конкретного графического приложения.

В случае инсталляции по умолчанию ОС Linux загружается в графическом режиме. После завершения начального процесса загрузки отображается экран диспетчера дисплеев, в котором необходимо зарегистрироваться для дальнейшей работы с системой. Типовое окно диспетчера дисплеев в ОС Linux представлено на рис. 2.1.

Рис. 2.1. Окно регистрации в системе (диспетчер дисплеев)

В окне регистрации, помимо ввода учетных данных пользователя, существует возможность выбора языковых параметров системы (**Language**), а также осуществления базовых операций, таких как завершение работы системы (**Shut Down**), перезапуск системы (**Restart**) и выбор желаемого сеанса работы (**Session**[1]).

После регистрации в системе пользователь попадает в окружение рабочего стола, которым по умолчанию является графическая среда GNOME.

---

[1] Для настройки параметров сохранения текущего сеанса работы используется меню **System ⇒Preferences ⇒More Preferences ⇒Sessions**. Данное меню позволяет настраивать параметры сохранения окон приложений и запуск программ при повторном открытии графического сеанса.

Графическая среда GNOME предоставляет пользователю следующие возможности:

- отображать содержимое файлов в требуемой программе просмотра;
- перемещать текст и графические объекты между произвольными графическими окнами посредством буфера обмена;
- использовать персональное графическое окружение посредством профилей;
- настраивать графическую среду произвольным образом.

Первоочередная задача GNOME заключается в упрощении работы пользователя. Чтобы официально стать частью рабочего стола GNOME, приложения должны соответствовать достаточно обширным требованиям, предъявляемым к пользовательскому интерфейсу. Благодаря тому что GNOME представляет собой замечательную платформу для разработки программ на языках C, C++, Python, Java и C#, в последнее время появилось большое количество приложений третьих фирм, которые официально не входят в состав GNOME.

На рабочем столе GNOME (рис. 2.2) по умолчанию отображаются ярлык доступа к файловому менеджеру **Nautilus** (**Computer**), ярлык доступа к домашнему каталогу пользователя (**root's Home**) и корзина (**Trash**) для удаления файлов.

Рис. 2.2. Рабочий стол GNOME

Верхняя панель рабочего стола используется для доступа к основным меню, отображения пиктограмм приложений и программ-уведомлений, таких как часы. В состав основных меню входят следующие:

- **Applications** (Приложения). В данном меню присутствуют дополнительные категории меню, названные исходя из предназначения приложений, доступ к которым они предоставляют. Например, категория меню **Office and Internet** (Офис и Интернет) содержит ярлыки доступа к офисным программам и веб-обозревателям;
- **Places** (Места). Данное меню позволяет получить быстрый доступ к наиболее частым местам посещения, таким как домашний[1] каталог, сетевой ресурс, внешние устройства. Это меню также содержит возможности поиска файлов (**Search For Files**), доступа к недавно использованным документам (**Recent Documents**) и удаленный доступ к другим компьютерам по различным протоколам, например FTP или CIFS;
- **System** (Система). Данное меню включает два подменю – **Preferences** (Свойства) и **Administration** (Администрирование). Меню свойств используется для настроек графического окружения, таких как выбор заставки рабочего стола, параметры отображения окон и настройка параметров клавиш мыши. Меню администрирования содержит ярлыки доступа к графическим утилитам администрирования служб, присутствующих в системе. Кроме того, в данном меню содержится ярлык «завершения сеанса» (**Logout**) и блокировки системы (**Lock**).

В правой части верхней панели располагаются пиктограммы отображения времени и контроля громкости, а также системные оповещения (например, о доступных обновлениях) и информативная информация от разных дополнений[2] (**Applets**), например таких, как «использование процессора» и «сетевые соединения». Одним из полезных дополнений графической среды GNOME является переключатель пользователей (**User Switcher**), который позволяет, не завершая текущего сеанса работы, выполнять вход в графический режим работы для других пользователей.

Нижняя панель рабочего стола используется для работы с интерактивными задачами, такими как выбор и сворачивание окон. В нижнем правом углу рабочего стола присутствует переключатель рабочих столов, позволяющий переключаться на три дополнительных рабочих стола. В нижнем левом углу экрана располагается кнопка свертывания открытых окон.

Для того чтобы эффективно работать в графической среде GNOME, достаточно научиться выполнять несколько следующих простых операций:

- *открыть или активизировать элемент на панели*. Щелкнуть на элементе левой кнопкой мыши один раз;

---

[1] ***Домашним*** каталогом называется каталог пользователя, в который он попадает после входа в систему.

[2] ***Дополнения*** – это небольшие приложения, выполняемые внутри панели. Их можно поместить на панель через меню **Add to Panel** (Добавить на панель).

- *запустить программу*. Запуск программы производится щелчком левой кнопки мыши на запускающем объекте. В GNOME эти объекты обычно располагаются на обеих панелях и на поверхности самого рабочего стола. Кроме того, когда производится щелчок на файле, он открывается ассоциированной с ним программой, о чем вскоре будет сказано дополнительно;
- *переместить элемент по рабочему столу*. Перетащить левой кнопкой мыши;
- *переместить элемент по панели*. Перетаскивание левой кнопкой мыши действует для запускающих объектов, но в некоторых апплетах левая кнопка используется для управления апплетом. В таком случае перетаскивание выполняется средней кнопкой. То же касается перемещения окон захватом границы: щелчок левой кнопкой раскрывает окно, щелчок средней кнопкой перемещает его;
- *упорядочить элементы на рабочем столе*. Щелкнуть правой кнопкой на фоне рабочего стола и выбрать пункт контекстного меню **Clean Up by Name** (Выстроить по имени). Элементы будут отсортированы в алфавитном порядке с двумя исключениями: первый элемент в левом верхнем углу всегда будет ярлыком домашнего каталога пользователя, а последний элемент в списке – всегда мусорная корзина;
- *открыть или активизировать элемент на рабочем столе*. Дважды щелкнуть на элементе. Двойной щелчок на значке папки открывает ее в файловом менеджере **Nautilus**. Если дважды щелкнуть на документе электронной таблицы, запустится приложение электронной таблицы **Gnumeric**, в котором откроется документ. Двойной щелчок средней кнопкой мыши или щелчок с нажатой клавишей **<Shift>** на папке внутри открытого окна файлового менеджера приводит к закрытию текущего окна и открытию окна с содержимым выбранной папки;
- *получить список параметров или задать свойства любого объекта*. Щелкнуть правой кнопкой мыши, чтобы получить список параметров для любого объекта. Например, можно изменить фон рабочего стола, щелкнув на нем правой кнопкой и выбрав пункт контекстного меню **Change Desktop Background** (Изменить фон рабочего стола). Другие общие настройки можно выполнить с помощью центра управления GNOME, который вызывается через меню **System ⇒Settings** или вводом команды **gnome-control-center** в командной строке;
- *вставить текст в любую текстовую область*. Как и в любой другой операционной системе, копирование выполняется с помощью комбинации клавиш **<Ctrl+C>**, вырезание – **<Ctrt+X>**, вставка – **<Ctrl+V>**. Исключение составляют программы **Emacs** и **XChat**. Можно использовать и более традиционный для UNIX способ, при котором вставка выделенного текста производится щелчком средней кнопки мыши.

Расширенная настройка среды GNOME выполняется при помощи программы **Gconf**. Gconf – это централизованная система хранения настроек приложений рабочего стола, основанная на XML. Она позволяет приложениям совместно ис-

пользовать настройки комбинаций быстрых клавиш, тем и прочих личных предпочтений пользователя. Кроме того, GConf может использоваться для блокировки настроек рабочего стола с гораздо более высокой степенью избирательности, чем это возможно с традиционными блокировками файлов в UNIX. Администратор может открыть или закрыть доступ к тем или иным функциям некоторого приложения. Администраторы интерактивных терминалов, общественных компьютерных центров и других систем, где особое значение приобретает высокая степень защиты, понимают необходимость наложения определенных ограничений. Поэтому большинство приложений предоставляют возможность ограничения функциональности через свои файлы настроек **GConf**. Для более глубокого ознакомления с данной темой рекомендуется прочитать руководство **«GNOME System Administrator's Guide»**, которое можно найти на сайте http://www.gnome.org.

2.1.2. Интерфейс командной строки (CLI)

В ОС Linux основным рабочим инструментом администратора и разработчика является ***интерфейс командной строки***[1], позволяющий работать с системой намного быстрее и эффективнее. В основе работы в командной строке лежит понятие интерпретатора команд.

Командный интерпретатор (оболочка) – это программа, имеющая свои собственные ***встроенные команды*** (built-in commands), свое собственное ***переменное окружение***[2] (environment); кроме того, командный интерпретатор позволяет выполнять внешние команды программ, которые присутствуют в системе.

Существуют следующие варианты использования командного интерпретатора:

- интерактивное использование;
- настройка переменного окружения (сеанса);
- программирование.

При интерактивном использовании командного интерпретатора система ожидает ввод команды в приглашении командной строки. Команды могут включать специальные символы, позволяющие выполнять сокращение имен файлов и перенаправление ввода и вывода.

Командный интерпретатор определяет переменные, управляющие работой текущего ***сеанса***[3] командной строки. Значения некоторых переменных предвари-

---

[1] ***Интерфейс командной строки*** (консоль) – это разновидность текстового интерфейса человека и компьютера, в котором инструкции компьютеру даются только с клавиатуры. Интерфейс командной строки противопоставляется системам управления программой на основе меню, а также различным реализациям графического интерфейса.

[2] ***Переменное окружение*** (environment) – это набор системных переменных, доступных всем процессам системы. В данных переменных задаются основные параметры системы, такие как пути к исполняемым файлам, локализация терминала, идентификатор пользователя и т. д.

[3] Под ***сеансом*** командной строки понимается процесс работы пользователя с интерфейсом командной строки.

тельно устанавливаются системой; определение других можно выполнять в загрузочных файлах, которые считываются при входе в систему. Находясь в интерактивном режиме, также можно определять переменные и использовать их в дальнейшей работе.

Командные интерпретаторы предоставляют набор специальных встроенных команд, позволяющих создавать программы, называющиеся ***сценариями*** командного интерпретатора. Сценарии полезны для выполнения наборов отдельных команд и повторения выполнения команд.

В графической оболочке запуск командного интерпретатора **bash** осуществляется при открытии программы **Terminal** из общего меню программ или из контекстного меню **Open Terminal**.

Если графическая оболочка не установлена, то пользователь автоматически попадает в режим командной строки, а именно в консольную программу **login**, где ему необходимо аутентифицироваться.

После удачного завершения процесса аутентификации открывается сеанс командного интерпретатора и появляется первичное ***приглашение*** вида **[root@rhel5 ~]#**. После появления данного приглашения можно приступать к вводу команд.

2.2. Виртуальные консоли. Базовые операции с системой

Будучи многозадачной системой, ОС Linux предоставляет ряд способов одновременного выполнения нескольких задач. Большинство пользователей Linux, которым нужен такой асинхронный доступ, пользуются графической системой **X Window**. Но прежде чем ее запустить, можно сделать нечто подобное с помощью виртуальных консолей.

Чтобы переключиться в другую виртуальную консоль, необходимо нажать левую клавишу **<Alt>** и, удерживая ее нажатой, нажать одну из функциональных клавиш от **<F1>** до **<F8>**. При нажатии каждой функциональной клавиши будет открываться новый экран с приглашением к регистрации. Можно регистрироваться в различных виртуальных консолях под разными пользователями и переключаться между консолями для выполнения различных действий. Можно даже в каждой консоли запустить свой сеанс **X Window**. X Window по умолчанию использует виртуальную консоль с номером 7. Поэтому, запустив **X** и переключившись затем в одну из текстовых виртуальных консолей, можно вернуться в **X**, нажав комбинацию **<Alt+F7>**. Существует возможность запустить два экземпляра X-сервера, в этом случае вторая графическая консоль с **X Window** будет иметь номер 8. Для перехода на 1-ю текстовую консоль из графической оболочки используется комбинация клавиш **<Ctrl+Alt+F1>**. При переходе в текстовый режим отобразится виртуальный терминал – один из нескольких виртуальных терминалов, имеющихся в системе по умолчанию. В большинстве случаев нет необходимости переключаться на другие консоли, если у вас уже загрузилась по умолчанию установленная консоль, однако в процессе инсталляции ОС Linux для просмотра более подробных событий необходимо переключиться в системную

консоль с номером 4. В табл. 2.1 приведено описание клавиш для перехода между виртуальными консолями в процессе инсталляции ОС Linux.

Таблица 2.1. Описание клавиш переключения между виртуальными консолями в процессе инсталляции

№ консоли	Комбинация клавиш	Описание
1	<**Ctrl**+**Alt**+**F1**>	Диалог инсталляции
2	<**Ctrl**+**Alt**+**F2**>	Командная строка
3	<**Ctrl**+**Alt**+**F3**>	Выводы системных сообщений программы инсталлятора
4	<**Ctrl**+**Alt**+**F4**>	Сообщения уровня ядра и устройств
5	<**Ctrl**+**Alt**+**F5**>	Дополнительные сообщения
6	<**Ctrl**+**Alt**+**F6**>	Графический сеанс

Помимо использования указанных выше комбинаций клавиш, для перехода на другую виртуальную консоль используется команда **chvt N**, где **N** – номер виртуальной консоли.

Регистрация в системе и завершение сеанса

После перехода в текстовую виртуальную консоль отображается приглашение системы на ввод аутентификационных данных следующего вида:

```
Red Hat Enterpirse Linux Server release 5.1 (TIkanga)
Kernel 2.6.18-53.el5 on an i686
srv login:
```

Первые две строки, не содержащие приглашения программы **login**, содержат краткую информацию о системе. Информация, присутствующая в данных строках, может изменяться в целях повышения информативности системы и определяется в файле **/etc/issue**.

Здесь необходимо ввести имя пользователя, а затем, когда будет выдано соответствующее приглашение, – его пароль (**Password:**). После успешной регистрации в системе можно приступать к работе.

После регистрации отобразится приглашение вида **#**, если вход в систему был выполнен от административного пользователя **root**[1], или приглашение вида **$** для всех других случаев.

Типичное приглашение для пользователя **root** имеет следующий вид:

```
[root@srv ~]#
```

Запись, находящаяся в квадратных скобках, может содержать полезную информацию, такую как имя пользователя, который в настоящий момент работает в данной консоли, имя компьютера и рабочий каталог.

---

[1] Учетная запись пользователя root имеет почти неограниченные права на доступ к файлам системы и запуск программ. В целях недопущения ошибки в процессе работы с ОС Linux рекомендуется использовать любую другую учетную запись, а программы, которые доступны только для пользователя **root**, запускать при помощи программы **sudo**.

Для завершения текущего сеанса работы используется команда **logout** или комбинация клавиш **<Ctrl+D>**.

Завершение работы и перезапуск системы

Для завершения работы используется команда **shutdown** или **halt**. В первом случае возможно задавать время завершения работы системы, определять запуск утилиты проверки файловой системы после очередной загрузки, а также отменять ранее введенную команду завершения работы. Например, для запланированного завершения работы системы необходимо ввести следующую команду:

```
# shutdown -h 20:00 &
```

Для перезапуска системы без выключения питания используется команда **reboot**, которая является ссылкой[1] на команду **halt**.

Определение параметров пользователя и задание пароля

Часто при работе с системой под разными пользователями на разных графических и текстовых терминалах необходимо определить группы, членом которых является данный пользователь, и его идентификатор, уникально идентифицирующий пользователя в системе. Для этого в ОС Linux существует команда **id**, вывод которой имеет следующий вид:

```
# id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
```

В данном случае пользователь имеет нулевой идентификатор, что соответствует пользователю **root**, основная группа пользователя имеет также нулевой идентификатор. В последней строке через запятую перечислены дополнительные группы, членом которых является данный пользователь.

Для задания пароля пользователя используется команда **passwd**. Для смены своего пароля пользователь должен ввести свой текущий пароль и два раза новый, при этом будет выполнена проверка соответствия пароля принятым правилам безопасности паролей[2]:

```
$ passwd
Changing password for user1
(current) UNIX password:
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

Для определения параметров пользователя, таких как его домашний каталог, командный интерпретатор, а также информация о последних активных сеансах работы пользователя с системой, используется команда **finger**, вывод которой представлен ниже.

---

[1] Ссылка – ярлык на объект файловой системы, при обращении к которой происходит обращение к объекту ссылки.

[2] Для просмотра доступных правил безопасности паролей можно просмотреть справку по модулю pam_cracklib, используя команду man pam_cracklib.

```
# finger root
Login: root                             Name: root
Directory: /root                        Shell: /bin/bash
On since Wed Aug 19 22:47 (MSD) on tty1    1 hour 28 minutes idle
On since Wed Aug 19 23:46 (MSD) on tty2    2 days 22 hours idle
On since Sat Aug 22 20:23 (MSD) on pts/2 from :0.0
   1 hour 28 minutes idle
On since Sat Aug 22 20:48 (MSD) on pts/3 from :0.0
Mail last read Fri Aug 14 21:07 2009 (MSD)
No Plan.
```

Иногда необходимо определить не только параметры конкретного пользователя, но и понять, кто в настоящий момент работает с системой. Данная информация содержится в следующих файлах:

- **/var/log/wtmp** – содержит информацию о времени регистрации пользователя в системе;
- **/var/run/utmp** – содержит список текущих зарегистрированных пользователей;
- **/var/log/faillog** – содержит список пользователей, неудачно аутентифицировавшихся в системе;
- **/var/log/lastlog** – содержит информацию о наиболее последних регистрациях.

Непосредственно данные файлы просмотреть нельзя. Для этого используются специальные утилиты, такие как **w** (определение зарегистрированных пользователей на данный момент в системе), **ac** (получение данных о длительности сеансов пользователей), **lastlog**, **faillog**, **last**, **lastb**. В частности, команда **last** позволяет определить, когда были выполнены перезагрузки системы (записи псевдопользователь **reboot**).

2.3. Графический сервер Xorg. Приложения GNOME

Модульная архитектура графической системы ОС Linux изображена на рис. 2.3.

На рис. 2.3 пользователь работает на локальной рабочей станции, где запущен локальный X-сервер. Система **X Window** работает по принципу *клиент-сервер*. В качестве клиентов выступают графические приложения, работающие как на локальной рабочей станции, так и на удаленной; в качестве сервера выступает X-сервер, работающий на локальной рабочей станции.

Диспетчер дисплеев, изображенный на рис. 2.3, может работать как на локальной, так и на удаленной рабочей станции. В первом случае диспетчер дисплеев осуществляет запуск локального X-сервера, отображает начальное окно входа в систему и требует ввести учетные данные пользователя для аутентификации. Во втором случае X-сервер выступает в качестве клиента по отношению к удаленному диспетчеру дисплеев. В этом случае обмен данными между локальным X-сервером и удаленным диспетчером дисплеев осуществляется по протоколу **XDMCP**; пользователь получает возможность работать с несколькими приложе-

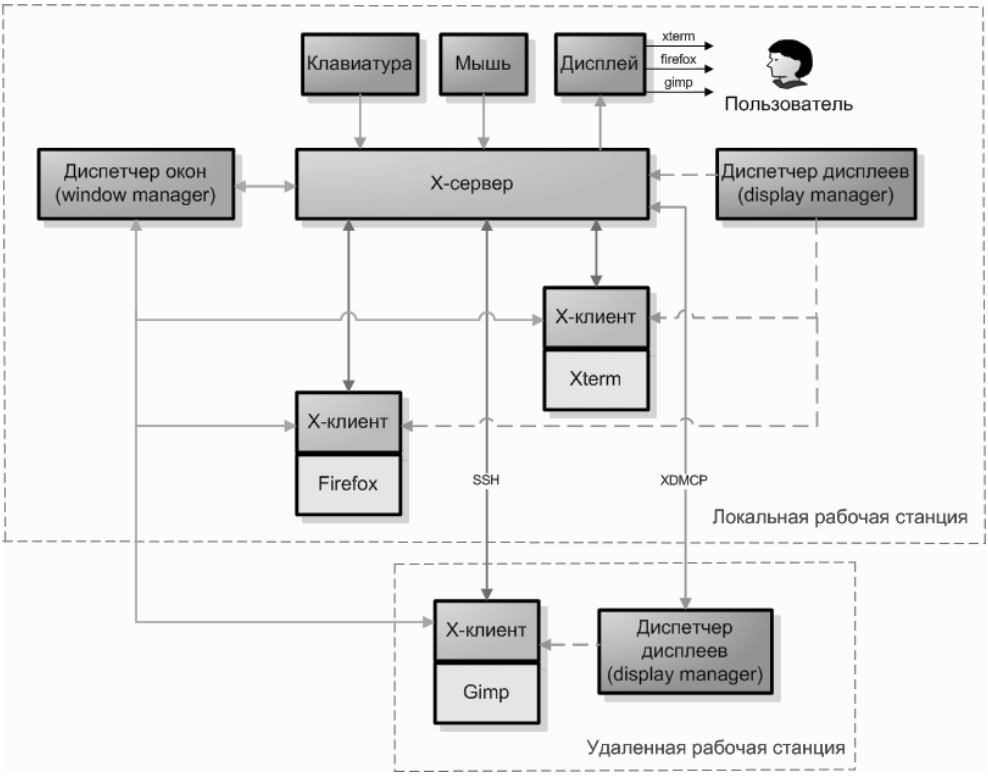


Рис. 2.3. Архитектура системы X Window

ниями, находящимися на разных компьютерах. Сам по себе протокол XDMCP использует UDP в качестве транспорта и не является защищенным. Для защиты протокола используется его туннелирование при помощи **SSH** (Secure Shell protocol).

По умолчанию запуск X-сервера осуществляется процессом **init** в процессе загрузки системы. Пользователь может выполнять запуск X-сервера вручную, непосредственно введя команду

```
X :<N> vt10,
```

где **<N>** – номер дисплея, а цифра 10 обозначает номер виртуального терминала. В результате выполнения данной команды на 10-м виртуальном терминале откроется новый сеанс X-сервера, однако для его использования необходимо запустить графическую программу-клиента, например терминал **xterm**:

```
# X :1 vt10 & sleep 2 ; DISPLAY=:1 xterm
```

В результате выполнения данной команды на 10-м виртуальном терминале будет запущена программа **xterm** (рис. 2.4).

Приведенная выше команда состоит из двух простых команд: первая запускает процесс X-сервера, а вторая запускает терминал **xterm** на этом сервере. Следует заметить, что второй команде передается значение переменной **DISPLAY**.

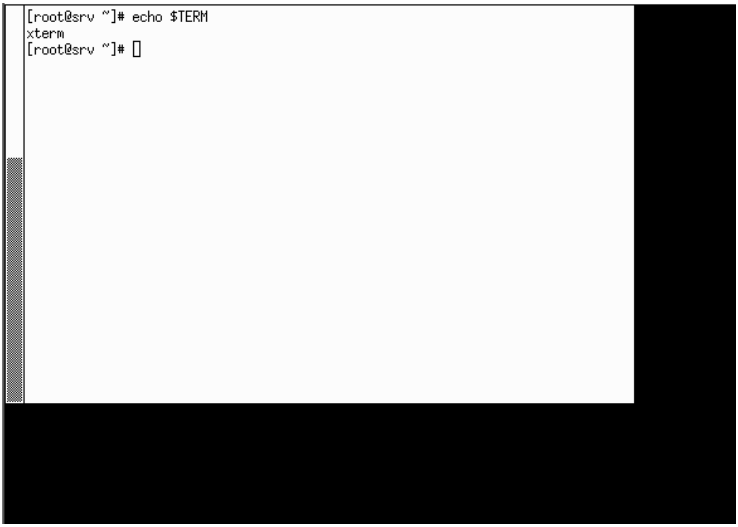


Рис. 2.4. Непосредственный запуск X-сервера и X-клиента

С точки зрения пользователя, каждый экземпляр X-сервера имеет собственное имя_дисплея, представленное в следующем формате:

```
имя_хоста:номер_дисплея.номер_экрана
```

Здесь в параметре *имя_хоста* указан хост, к которому физически подключен дисплей. Если этот параметр пропущен, значит, подключение будет выполняться к локальному X-серверу. Обязательный параметр *номер_дисплея* обозначает порядковый номер X-сервера, к которому необходимо подключиться. Нумерация дисплеев начинается с нуля. Параметр *номер_экрана* определяет, на каком физическом мониторе следует отображать графику. Нумерация экранов начинается с нуля. Таким образом, для запуска графического приложения на удаленном хосте **foo.bar.host** на 5-м экземпляре X-сервера и его отображения на 2-м мониторе необходимо указать дисплей в следующем формате:

```
DISPLAY=foo.bar.host:5:2
```

Запуск дополнительного экземпляра X-сервера удобно выполнять посредством обращения к менеджеру дисплеев **gdm**. Для этого используется команда **gdmflexiserver**. При запуске данной команды будет предложено выполнить запуск диспетчера дисплеев в текущем сеансе X-сервера или новом сеансе. Кроме того, существует возможность запуска вложенного **(nested)** сеанса X-сервера в текущем графическом сеансе (рис. 2.5).

Основные операции с X-сервером

Большинство операций по работе с X-сервером, такие как изменение разрешения экрана, запуск X-клиентов, изменение параметров окон, выполняются через графический интерфейс GUI. Для изменения разрешения экрана используется меню **System ⇒Administration ⇒Display**. Однако для применения введенных на-

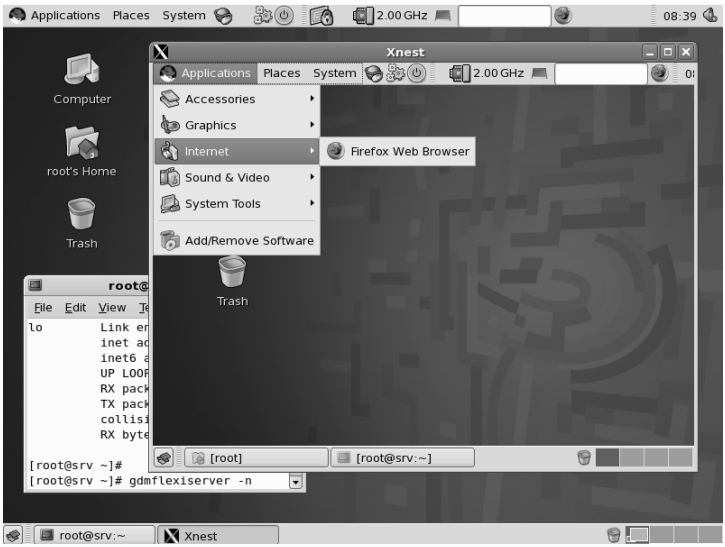


Рис. 2.5. Запуск вложенного сеанса X-сервера

строек необходимо перезапустить текущий сеанс X-сервера. Существует возможность динамически изменять разрешение экрана без завершения текущего сеанса, используя команду **xrandr -s <N>x<M>**, где **<N>** – горизонтальное разрешение экрана (в пикселах), а **<M>** – вертикальное разрешение экрана (в пикселах).

Как и в любой системе, в ОС Linux для выполнения операций копирования и вставки текста или любых других объектов существует ***буфер обмена***. Для копирования объекта используется комбинация **<Ctrl+C>**, для вставки объекта используется комбинация **<Ctrl+V>**. При работе с текстом данные операции можно осуществлять при помощи мыши. Для копирования фрагмента текста достаточно выделить текст, нажав левую кнопку мыши. Для вставки текста используется **средняя** кнопка мыши.

Наиболее часто использующейся операцией при работе с X-сервером является запуск произвольной графической программы (X-клиента). Для этого можно использовать имеющийся ярлык программы на рабочем столе или в меню программ, а также из командной строки, используя символ запуска программы в фоновом режиме (**&**):

```
$ xclock &
```

При работе с большим количеством графических окон и приложений возможно появление графических артефактов (например, часть свернутого окна может отображаться на рабочем столе после закрытия самого приложения), связанных с неправильной работой приложения. Для этого не обязательно перезапускать сеанс X-сервера, достаточно выполнить перерисовку всех графических объектов, используя команду **xrefresh**.

На серверах, а также на клиентах с ограниченными аппаратными ресурсами нет необходимости регулярно работать в графическом режиме, однако периодически, например для поиска какой-либо информации в Интернете, необходимо запустить графическое приложение, установленное на данном сервере. Существуют несколько вариантов решения данной задачи:

- запуск графического приложения с использованием другого X-сервера другого хоста;
- запуск графического приложения посредством выполнения команды **startx**;
- запуск графического приложения путем перезапуска системы в графический режим работы.

В первом варианте необходимо наличие на другой машине работающего X-сервера. Для того чтобы запустить приложение, используя удаленный X-сервер, необходимо выполнить следующую команду:

```
$ DISPLAY=10.31.5.120:0.1 /usr/bin/firefox
```

В результате выполнения данной команды на удаленном сервере с IP-адресом 10.31.5.120 на дисплее с номером 1 будет отображен вывод локальной графической программы firefox. Существует возможность использовать не только X-сервер, присутствующий в ОС семейства UNIX, но и X-сервер, работающий под управлением ОС Windows[1].

В третьем случае для запуска локального графического приложения необходимо запустить полноценный графический сеанс, включая запуск графической среды GNOME и дополнительных приложений, а также потратить некоторое время на ожидание перезапуска системы.

Наиболее оптимальным вариантом является запуск графического приложения в новом сеансе X-сервера посредством команды **startx**:

```
$ startx /usr/bin/firefox -geometry 1024x768 -- :1
```

В результате выполнения данной программы на 8-м виртуальном терминале будет запущен сеанс X-сервера с запущенной программой **Firefox**. Команда **startx** является сценарием командной оболочки и использует для запуска X-сервера утилиту **xinit**.

Приложения Gnome

В первом модуле уже было дано описание рабочей среды GNOME, ее основных возможностей и базовых операций. В данном разделе будет приведено описание приложений, являющихся частью среды GNOME.

Окружение рабочего стола GNOME тесно интегрировано с пакетом офисных приложений **OpenOffice**, который дает возможность единообразной работы с текстовыми документами, электронными таблицами и презентациями. Пакет OpenOffice обладает превосходной совместимостью с файлами, созданными с помощью Microsoft Office, и предлагает богатый набор функциональных возможностей, необходимых для повседневного использования.

---

[1] Xming – бесплатный вариант X-сервера для ОС Windows. Более подробная информация доступна на сайте http://www.straightrunning.com/XmingNotes/.

В состав пакета OpenOffice входят следующие программы:

- Writer – используется для обработки текста;
- Calc – используется для работы с электронными таблицами;
- Impress – используется для работы с презентациями;
- Draw – используется для рисования;
- Math – используется для работы с математическими формулами.

Существуют и другие офисные приложения. Например, приложение электронной таблицы **Gnumeric** умеет обращаться с некоторыми файлами, которые оказываются не под силу OpenOffice, и позволяет выполнять более сложные финансовые вычисления. Приложение **AbiWord** – превосходный текстовый редактор, способный решать значительную часть задач, и к тому же более простой, чем OpenOffice. Оба этих приложения занимают гораздо меньше дискового пространства, работают быстрее и подходят для установки в системы с ограниченными ресурсами. Для просмотра документов формата PDF можно использовать программу **Evince**, входящую в состав офисного пакета GNOME, или бесплатный вариант Adobe Acrobat Reader.

Для работы с базами формата Microsoft Access и другими в состав пакета OpenOffice входит программа **OpenOffice Base**, имеющая штатную возможность взаимодействия с другими базами данных, такими как ADO, MySQL, PostgreSQL, и позволяющая посредством драйверов ODBC и JDBC взаимодействовать с любыми приложениями. Также данное приложение поддерживает стандарт LDAP и стандарты Microsoft Outlook, Microsoft Windows и Mozilla.

Для прослушивания медиа-контента в состав среды GNOME входит встроенный проигрыватель компакт-дисков, а также аудиоплееры **Rhythmbox** и **Amarok**, поддерживающие все известные форматы файлов, включая mp3, ogg, wav, wma, mid. Для просмотра видеофайлов в среду GNOME входит программа Totem, поддерживающая форматы DVD, AVI, MP4, WMV, MOV и др.

Пользователи, которым необходим мощный инструмент обработки фотоматериалов и любых других графических изображений, также получат для этого все необходимые наборы инструментов, включая графический редактор изображений **Gimp**, схожий по многообразию возможностей с графическим инструментом **Photoshop**.

Для работы в сети Интернет, с электронной почтой и обмена мгновенными сообщениями среда GNOME предоставляет все необходимое. Веб-обозреватель **Mozilla Firefox**, почтовый клиент **Evolution** и клиент обмена сообщениями **Pidgin** поддерживают самые современные технологии и возможности, позволяя пользователю работать с максимальной эффективностью.

Одна из ключевых возможностей ОС Linux – это полноценная среда разработки и отладки приложений. В состав ОС Linux входит большое количество средств разработки, включая компилятор **GCC**[1], отладчик программ **GDB**[2] и универсальную среду разработки **Eclipse**[3].

---

[1] http://gcc.gnu.org/
[2] http://www.gnu.org/software/gdb/
[3] http://www.eclipse.org/

Модуль 3. Знакомство с файловой системой

После завершения изучения данного модуля вы научитесь:

- понимать иерархию файловой системы;
- выполнять базовые операции по управлению файлами;
- понимать принцип расположения различных типов файлов в иерархии файловой системы;
- использовать возможности одновременной работы с группами файлов.

3.1. Предназначение файловой системы

Одним из наиболее важных компонентов ОС является файловая система. В данном разделе будут рассмотрены понятие файловой системы и ее предназначение; иерархия файловой системы ОС Linux; типы файлов ОС Linux. Остальные вопросы, касающиеся администрирования файловой системы, более подробно будут рассмотрены в модуле 12.

В ОС Linux, как и в любой другой ОС семейства UNIX, любой объект является файлом, хранящимся на ***файловой системе***. Файловая система физически представляет собой некоторое устройство (например, жесткий диск), отформатированное для хранения файлов. Файловые системы могут находиться на жестких дисках, гибких дисках, CD-ROM или других носителях, которые позволяют осуществлять произвольный или последовательный доступ к данным.

Основными задачами файловой системы являются:

- упорядочивание хранимых данных;
- простой и быстрый доступ к хранимым данным;
- обеспечение целостности хранимых данных.

Точный формат и способы хранения файлов в ОС Linux не имеют значения, так как система обеспечивает общий интерфейс для всех распознаваемых ею типов файловых систем. В ОС Linux файловой системой, устанавливаемой по умолчанию, является **ext3fs**. При доступе к любой файловой системе из ОС Linux данные представляются в виде ***иерархии каталогов*** с находящимися в них файлами вместе с идентификаторами владельцев и групп, битами прав доступа и прочими атрибутами, о которых будет сказано позже.

3.2. Иерархия файловой системы

Иерархия каталогов файловой системы ОС Linux соответствует общепринятому в мире UNIX стандарту **Filesystem Hierarchy System**[1] (FHS) (табл. 3.1). Основное преимущество данного стандарта заключается в том, что определенные типы файлов расположены в соответствующих каталогах. Например, большинство конфигурационных файлов располагаются в каталоге **/etc**, а файлы журналов различных сервисов – в каталоге **/var/log**.

Таблица 3.1. Описание каталогов файловой системы ОС Linux

Каталог	Описание
/boot	Содержит файлы загрузчика ОС Linux, образ памяти и файл ядра. Содержимое данного каталога необходимо для нормальной загрузки системы
/bin	Содержит основные программы ОС Linux, включая командный интерпретатор bash и многие другие основные команды, такие как **cp**, **mv** и **rm**
/sbin	Содержит важные программы системного администрирования ОС Linux, включая программы **fdisk**, **fschk**, **ifconfig**, **mkfs**, и **init**
/dev	Содержит специальные файлы, которые используются для обращения к различным аппаратным устройствам оборудования
/proc	Содержит в себе ***псевдофайловую*** систему **procfs**, используемую ОС Linux для организации взаимодействия процессов с пространством ядра. В данном каталоге содержится ряд подкаталогов, именуемых согласно **идентификаторам процессов** (PID) и содержащих информацию по каждому запущенному процессу
/etc	Содержит большинство конфигурационных файлов сервисов и файлы инициализационных скриптов
/home	Содержит набор подкаталогов, являющихся домашними каталогами пользователей ОС Linux
/lib	Содержит библиотеки, используемые программами, находящимися в каталогах **/bin** и **/sbin**
/mnt	В данный каталог осуществляется монтирование файловых систем внешних устройств – CD- и DVD-дисков, USB-накопителей и дискет
/opt	Содержит бинарные и конфигурационные файлы, а также принадлежащие им библиотеки. Объекты данного каталога, как правило, являются частью дополнительно установленного ПО
/usr	Содержит бинарные и конфигурационные файлы программ, установленных из исходных кодов, страницы руководства man, а также все остальные файлы программ, установленных в системе
/tmp	Временный каталог, в котором содержатся временные файлы, создаваемые различными приложениями в процессе работы системы
/var	Содержит различные изменяемые данные, в том числе журнальные файлы сервисов и очередей

[1] Более подробная информация по стандарту FHS представлена на сайте http://www.pathname.com/fhs/.

Каталоги **/bin**, **/usr/bin**, **/usr/locl/bin**, **/sbin**, **/usr/sbin** и **/usr/local/sbin** содержат установленные в системе команды. При работе в качестве обычного пользователя[1] вам будут доступны только команды каталогов **/bin, /usr/bin** и **/usr/locl/bin**, так как в стандарте FHS определено, что в каталогах **sbin** должны содержаться лишь административные команды. Общая структура каталогов отражена на рис. 3.1

Основным каталогом файловой системы ОС Linux является корневой каталог

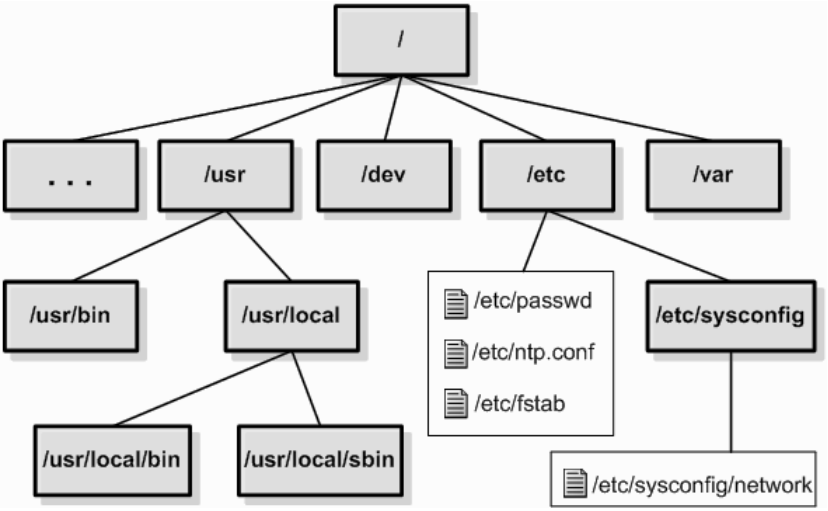


Рис. 3.1. Структура каталогов ОС Linux

/. Ниже располагаются все остальные каталоги, созданные в локальной дисковой подсистеме или ***смонтированные*** с внешних устройств. Процедура монтирования файловой системы означает связывание каталога существующей файловой системы, называемого ***точкой монтирования***, с корневым каталогом новой файловой системы.

Монтирование файловой системы к точке монтирования осуществляется при помощи команды **mount**. В следующем листинге приведен пример монтирования DVD-привода, содержащего дистрибутив ОС Red Hat Enterprise Linux 5.

```
    # mount /dev/hdc /mnt/
mount: block device /dev/hdc is write-protected, mounting read-only
```

В результате выполнения команды **mount** система вывела информацию о том, что блочное устройство (в данном случае DVD-привод) было смонтировано в режиме чтения.

Список смонтированных файловых систем хранится в файле **/etc/fstab**. Бла-

---

[1] В ОС Linux всех пользователей условно можно разделить на три группы: привилегированный, или суперпользователь (**root**), служебные пользователи и все остальные (обычные) пользователи.

годаря этому возможны автоматическая проверка целостности файловой системы при помощи команды **fsck** и монтирование файловых систем на этапе начальной загрузки, а также выполнение сокращенных команд наподобие **mount /var/spool**. Информация, содержащаяся в этом файле, отражает расположение файловых систем на диске. Подробнее файл **/etc/fstab** будет рассмотрен в модуле 12.

Размонтирование файловых систем осуществляется командой **umount**. «Заблокированную» файловую систему размонтировать невозможно. В ней не должно быть ни открытых файлов, ни текущих каталогов выполняющихся процессов. Если размонтируемая файловая система содержит исполняемые программы, они не должны быть запущены. В следующем листинге приводится пример размонтирования ранее подмонтированной файловой системы в каталоге **/mnt**.

```
# umount /mnt
```

Для того чтобы узнать, какие устройства смонтированы к системе, необходимо выполнить команду **mount** без параметров. В следующем листинге приведен пример определения смонтированных устройств.

```
# mount
/dev/sda2 on / type ext3 (rw)
proc on /proc type proc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
/dev/hdc on /mnt type iso9660 (ro)
```

Как видно, в выводе команды **mount** последовательно отображаются смонтированное устройство, точка монтирования, тип файловой системы и дополнительные опции монтирования.

Помимо монтирования удаленных файловых систем, существует возможность смонтировать локальный каталог к другому локальному каталогу (`mount --bind`). В результате содержимое смонтированного каталога становится доступным в двух местах, что равнозначно по смыслу символьной ссылке, с той лишь разницей, что ее нельзя удалить, в отличие от обычной символьной ссылки.

3.3. Типы файлов

При работе с ОС Linux важно понимать, что любой объект ОС ***является файлом***. Это ключевая особенность ОС Linux, по сравнению с операционными системами семейства Windows.

Файлы различаются как по своей структуре, так и по своему предназначению. В ОС Linux определены семь типов файлов:

- регулярные файлы (обычные файлы);
- каталоги;
- ссылки;
- сокеты;
- именованные каналы;
- файлы блочных устройств;
- файлы символьных устройств.

Определить тип файла можно с помощью команды **ls -ld**. Первый символ

в строке вывода обозначает тип файла. В следующем листинге выдается информация о файле **/dev/hdc**.

```
# ls -ld /dev/hdc
brw-rw---- 1 root disk 22, 0 Dec 22 13:30 /dev/hdc
```

Регулярные файлы

К регулярным файлам относятся бинарные файлы, библиотеки, текстовые файлы и файлы различных приложений. ОС Linux не накладывает никаких ограничений на структуру данных файлов. К их содержимому возможен как последовательный, так и прямой доступ.

Каталоги

Файлы данного типа хранят именованные ссылки на другие файлы. Имя файла, находящегося в каталоге, в действительности хранится в родительском каталоге, а не в самом файле.

В ОС Linux имеются так называемые специальные каталоги. Специальные каталоги, такие как «**.**» и «**..**», обозначают, соответственно, текущий рабочий каталог и его родительский каталог.

Ссылки

В ОС Linux принято различать символьные и жесткие ссылки, каждая из которых имеет особенное значение.

Символьная ссылка – это специальный файл, для которого в файловой системе не хранится никакой информации, кроме текстовой строки, содержащей путь к файлу, который должен быть открыт при обращении к данной ссылке. ***Жесткая ссылка*** является, по сути, дополнительным именем файла, то есть указывает непосредственно на ***индексный дескриптор файла***[1]. Таким образом, понятия «жесткая ссылка на файл» и «имя файла» являются синонимами. Файл считается удаленным после удаления его последней жесткой ссылки, однако место на файловой системе освободится только тогда, когда его индексный дескриптор перестанет использоваться.

Символьные ссылки могут содержать произвольное имя, то есть в них разрешается указывать на файлы, хранящиеся в других файловых системах, и даже на несуществующие файлы. Жесткие ссылки не могут указывать на файл, находящийся в другой файловой системе.

ОС Linux подсчитывает количество ссылок на каждый файл и при удалении файла не освобождает блоки данных до тех пор, пока не будет удалена последняя ссылка на него.

---

[1] ***Индексный дескриптор файла (инод)*** – это структура данных, уникально характеризующая каждый файл в рамках одной файловой системы. Иноды хранят информацию о файлах, такую как принадлежность владельцу (пользователю и группе), режим доступа (чтение, запись, запуск на выполнение) и тип файла. Каждый инод имеет уникальный идентификатор в файловой системе.

Сокеты

Сокет – это специальный тип файла, используемый процессами для взаимодействия друг с другом. Установленные посредством сокетов соединения позволяют процессам взаимодействовать, не подвергаясь влиянию других процессов. В ОС Linux выделяются несколько видов сокетов, использование которых предполагает наличие сетевой инфраструктуры. Локальные сокеты доступны только на локальном компьютере, обращение к ним осуществляется через специальные объекты файловой системы, а не через сетевые порты. Такие сокеты принято называть ***сокетами домена UNIX*** (UNIX domain socket). Помимо локальных сокетов, существуют сетевые сокеты, позволяющие процессам обмениваться данными по сети.

Несмотря на то что другие процессы распознают файлы сокетов как элементы каталога, чтение и запись файлов сокета могут осуществлять только те процессы, между которыми установлено соответствующее соединение. С локальными сокетами работают различные сервисы ОС Linux – **CUPS**[1], **X Window** и **Syslog**[2].

Именованные каналы (FIFO)

Файлы данного типа подобны сокетам, поскольку тоже используются для взаимодействия между процессами, однако, в отличие от сокетов, в именованных каналах данные передаются только в одном направлении.

Файлы блочных и символьных устройств

Файлы блочных и символьных устройств позволяют приложениям получать доступ к аппаратным средствам и периферийному оборудованию системы. На этапе конфигурирования к ядру ОС Linux динамически подгружаются необходимые модули, предназначенные для управления аппаратными средствами системы. За управление конкретным устройством отвечает специальный модуль, называемый ***драйвером устройства***.

Драйверы устройств образуют стандартный интерфейс взаимодействия, который воспринимается пользователем как набор обычных файлов. Получив запрос к файлу символьного или блочного устройства, ядро передает данный запрос соответствующему драйверу. Файлы блочных и символьных устройств сами по себе не являются драйверами; их можно рассматривать как шлюзы, через которые драйвер принимает запросы на выполнение предусмотренных операций.

Файлы символьных устройств не используют буферизацию в процессе операций ввода-вывода: они осуществляются немедленно по мере поступления. К символьным устройствам относятся виртуальные терминалы, модемы и другие устройства, не поддерживающие произвольного доступа к данным.

Файлы блочных устройств обрабатываются драйверами, которые осуществляют ввод-вывод данных цельными блоками. В данном случае на уровне ядра выполняется буферизация данных ввода-вывода. К блочным устройствам относятся

---

[1] Система печати в UNIX.

[2] Демон журналирования событий в UNIX.

жесткие диски, DVD-приводы, модули памяти, а также все остальные устройства, поддерживающие произвольный доступ к данным.

Файлы блочных и символьных устройств характеризуются двумя номерами: **старшим** (major) и **младшим** (minor). Старший номер устройства позволяет ядру определить, к какому драйверу относится файл, а младший номер идентифицирует конкретное физическое устройство.

Просмотреть номера файлов устройств можно, используя команду **ls**:

```
# ls -l /dev
brw-r----- 1 root disk     8,   0 Dec 22 13:30 sda
brw-r----- 1 root disk     8,   2 Dec 22 13:30 sda2
brw-rw---- 1 root disk    22,   0 Dec 22 13:30 hdc
crw-rw---- 1 root root     4,   0 Dec 22 16:29 tty0
crw--w---- 1 root tty      4,   1 Dec 23 00:05 tty1
crw------- 1 root root     5,   1 Dec 22 16:43 console
```

Из данного листинга видно, что блочному устройству **sda2** соответствуют старший номер 8 (номер драйвера) и младший номер 2 (номер устройства). Номер драйвера 8 в ОС Linux соответствует драйверу SCSI дисков **sd**. Младший номер устройства 2 является порядковым номером данного устройства, то есть, по сути, файл **/dev/sda2** представляет второй ***раздел***[1], созданный на жестком диске SCSI **sda**.

Информацию о старших и младших номерах, используемых конкретным драйвером устройства, можно найти в четвертом разделе справочного руководства **man** для данного драйвера.

3.4. Имена файлов и каталогов: группировка и использование подстановок

Часто пользователю приходится работать с большим количеством файлов, имеющих повторяющиеся символы в названии или другие схожие атрибуты. Для обращения к группе файлов, сгруппированных по какому-то признаку, используются специальные символы.

К ***специальным*** относятся символы «*****», «**?**», «**{}**», «**[]**», «**^**», «**~**». Каждый раз при вводе строки в командной строке командный интерпретатор проверяет введенные слова на наличие символов «*****», «**?**» и «**[**». В случае нахождения одного из указанных символов данное слово рассматривается интерпретатором как ***шаблон*** и заменяется отсортированным по алфавиту списком имен файлов, удовлетворяющим шаблону.

Любой символ, входящий в состав шаблона и не являющийся специальным символом, интерпретируется обычным образом, то есть обозначает то, чем является. Если необходимо составить шаблон, включающий специальный символ, то шаблон должен заключаться в кавычки. Ниже приводится описание доступных шаблонов подстановок.

---

[1] Под ***разделом*** понимается независимая область данных, расположенная на жестком диске. Совокупность таких областей называется разметкой диска.

***** Соответствует любой строке, включая строку нулевой длинны. Если данный символ указывается после символа / два раза, то данному шаблону будут соответствовать все каталоги и подкаталоги, а если данный символ встречается в шаблоне имени файла два раза подряд, то данному шаблону будут удовлетворять все файлы, каталоги и подкаталоги.

Например, для отображения всех файлов в рабочем каталоге, оканчивающихся на **.jpg**, можно использовать следующий шаблон подстановки:

```
# ls *.jpg
```

? Соответствует любому единичному символу.

Например, для отображения всех файлов в рабочем каталоге, оканчивающихся на **.jpg** и имеющих только один символ до точки, можно использовать следующий шаблон подстановки:

```
# ls ?.jpg
```

[...] Соответствует одному любому из указанных символов. Если между символами указан знак «-», то шаблону будет соответствовать диапазон отсортированных[1] символов, включая два указанных (первый и третий). Если перед первым символом шаблона указан символ «!» или «^», то шаблону будут соответствовать любые символы, кроме указанных в образце. Образцу вида [*.символ.*] соответствует указанный в образце *символ*.

Например, для отображения всех файлов в рабочем каталоге, оканчивающихся на .jpg и начинающихся только с заглавной буквы, можно использовать следующий шаблон подстановки:

```
# ls [A-Z]*.jpg
```

{} Соответствует каждому из приведенных образцов.

Например, для отображения всех файлов, оканчивающихся на символы **c**, **o** и **h**, можно использовать следующий шаблон подстановки:

```
# ls *.{c,o,h}
```

В фигурных скобках допустимо использование диапазонов символов посредством использования конструкции **{x..y}**, где **x** и **y** – начальный и конечный символы диапазона.

?(список_образцов[2]) Соответствует нулевому или единичному вхождению образца, указанного в списке образцов.

Например, для отображения всех файлов в рабочем каталоге, оканчивающихся на **.jpg** и **.gif**, начинающихся только с символов «**ab**» или «**cd**», можно использовать следующий шаблон подстановки:

```
# ls +(ad|cd)*+(.jpg|.gif)
```

---

[1] Порядок сортировки определяется текущей системной локализацией (переменная **LANG**) или переменной окружения **LC_COLLATE** (способ сортировки), если предыдущая не определена.

[2] Под списком образцов понимаются наборы символов, разделенных знаком «|».

***(список_образцов)** Соответствует нулевому и более вхождениям образца, указанного в списке образцов.

+(список_образцов) Соответствует одному и более вхождениям образца, указанного в списке образцов.

@(список_образцов) Соответствует одному из указанных в списке образцов.

!(список_образцов) Соответствует любому, кроме указанных в списке образцов.

Например, для отображения всех файлов в рабочем каталоге, не оканчивающихся на **.jpg** и **.gif**, можно использовать следующий шаблон подстановки:

```
# ls !(*.jpg|*.gif)
```

Следует отметить, что первые четыре шаблона подстановки являются стандартными. Остальные шаблоны становятся доступными только после активации опции расширенных подстановок командного интерпретатора **bash**. Для их активации необходимо ввести команду

```
shopt -s extglob
```

Кроме того, в подстановке вида [...] можно указывать специальные классы образцов (POSIX-стандарт), заключая их в двоеточия с обеих сторон. Для использования доступны следующие классы стандарта POSIX:

- alnum – соответствует буквенным и цифровым символам (эквивалентно A-Za-z0-9);
- alpha – соответствует буквенным символам (эквивалентно A-Za-z);
- blank – соответствует символу пробела или табуляции;
- cntrl – соответствует символу, получаемому при нажатии клавиши <**Ctrl**>;
- digit – соответствует только цифровым символам (эквивалентно 0–9);
- graph – соответствует символам с 33 по 126 таблицы кодов ASCII;
- print – соответствует аналогичному выше типу, включая символ пробела;
- lower – соответствует строчным буквенным символам (эквивалентно a–z);
- upper – соответствует прописным буквенным символам (эквивалентно a–z);
- punct – соответствует знакам пунктуации;
- space – соответствует символу пробела или горизонтальной табуляции;
- xdigit – соответствует символам шестнадцатеричных цифр (эквивалентно 0-9A-Fa-f).

Для использования данных шаблонов необходимо заключать их в квадратные скобки. Например, для отображения всех файлов, состоящих из четырех символов, второй и третий из которых являются цифрами, можно использовать следующий шаблон подстановки:

```
# ls ?[[:digit:]][[:digit:]]?
```

Для того чтобы интерпретатор корректно обработал введенную команду, включающую специальные символы в именах файлов или каталогов, существует понятие ***экранирования***. Для экранирования символов используется специальный символ «\». В следующем листинге приведен пример экранирования специального символа «**$**»:

```
# echo $#
0
# echo \$#
$#
```

Специальная переменная **$#** содержит количество аргументов команды. В первом случае при помощи команды **echo** выводится значение данной переменной. Во втором случае символ «**$**» экранируется и в результате команда echo выводит обычный текст. Тот же самый принцип экранирования действует для символов, содержащихся в подстановках имен файлов.

Модуль 4. Основы работы с командной строкой

Изучив данный модуль, вы будете иметь возможность:

- понимать принципы работы с командами;
- работать в командном интерпретаторе bash;
- осуществлять поиск необходимой справочной информации;
- настраивать переменные окружения командного интерпретатора;
- настраивать командный интерпретатор bash;
- работать с псевдонимами команд.

4.1. Командные интерпретаторы

Командный интерпретатор (shell) – самая важная часть ***интерфейса командной строки***[1] и ОС Linux в целом. Командный интерпретатор – это программа, имеющая свои собственные ***встроенные команды*** (built-in commands), свое собственное ***переменное окружение*** (environment); кроме того, командный интерпретатор позволяет выполнять внешние команды программ, которые присутствуют в системе.

Существуют следующие варианты использования командного интерпретатора:

- интерактивное использование;
- настройка переменного окружения (сеанса);
- программирование.

При интерактивном использовании командного интерпретатора система ожидает ввод команды в приглашении командной строки. Команды могут включать специальные символы, позволяющие выполнять сокращение имен файлов и перенаправление ввода и вывода.

Командный интерпретатор определяет переменные, управляющие работой текущего ***сеанса***[2] командной строки. Настройка данных переменных сообщает системе, например, о том, какой каталог следует применять в качестве домашнего или какой файл нужно использовать для хранения почты. Значения некоторых

---

[1] ***Интерфейс командной строки*** (консоль) – это разновидность текстового интерфейса человека и компьютера, в котором инструкции компьютеру даются только с клавиатуры. Интерфейс командной строки противопоставляется системам управления программой на основе меню, а также различным реализациям графического интерфейса.

[2] Под ***сеансом*** командной строки понимается процесс работы пользователя с интерфейсом командной строки.

переменных предварительно устанавливаются системой; определение других можно выполнять в загрузочных файлах, которые считываются при входе в систему. Находясь в интерактивном режиме, также можно определять переменные и использовать их в дальнейшей работе.

Командные интерпретаторы предоставляют набор специальных встроенных команд, позволяющих создавать программы, называющиеся ***сценариями*** командного интерпретатора. Сценарии полезны для выполнения наборов отдельных команд и повторения выполнения команд.

В состав ОС Linux входят несколько командных интерпретаторов, каждый из которых обладает своими возможностями, однако основным командным интерпретатором является **Bash** (Bourne Again Shell). Это наиболее распространенный командный интерпретатор, используемый в ОС Linux по умолчанию. С помощью Bash можно редактировать командную строку, вести историю команд и вести целочисленные вычислительные операции. В данном курсе все практические работы и упражнения будут выполняться именно в этом интерпретаторе. Определение используемого командного интерпретатора возможно посредством обращения к переменной окружения **$SHELL**, которая содержит абсолютный путь к бинарному исполняемому файлу командного интерпретатора. Для этого можно воспользоваться командой **echo**, которая выводит значение переменной, или командой **env**, отображающей все переменные окружения.

В графической оболочке запуск командного интерпретатора **bash** осуществляется при открытии программы **terminal** из общего меню программ. Если графическая оболочка не установлена, то пользователь автоматически попадает в консольную программу **login**, где ему необходимо аутентифицироваться, а затем приступить к работе в командной строке.

После удачного завершения процесса аутентификации открывается сеанс командного интерпретатора и появляется первичное ***приглашение***.

Приглашением командного интерпретатора принято называть весь текст, отображаемый слева от курсора, при условии, что курсор находится в самом начале командной строки. Данный текст несет полезную информацию и может настраиваться согласно предпочтениям пользователя.

В командном интерпретаторе bash существуют следующие приглашения.

- **PS1 (первичное приглашение).** В данной переменной содержится строка вида **[\u@\h \W]\$**, в которой каждому символу соответствует определенное значение. Например, символу **\u** соответствует имя текущего пользователя. Допустимые символы, используемые для формирования первичного приглашения, приведены в табл. 4.1.
- **PS2 (вторичное приглашение).** Данная переменная содержит второстепенное приглашение, которое возникает при многострочном редактировании текста или незавершенном вводе команды. По умолчанию оно обозначается как **>**. Пример вторичного приглашения приведен в следующем примере запуска команды получения информации DNS:

```
# nslookup
>
```

- **PS3**. Данная переменная содержит приглашение, присутствующее в операторе select, используемом для организации интерактивных консольных меню. Приглашению **PS3** в операторе select по умолчанию соответствует значение **#?**. В следующем примере приведено произвольное консольное меню, в котором пользователю предлагается сделать определенный выбор:

```
1) one
2) two
#? 1
One
#? 2
Two
```

- **PS4.** Данная переменная используется в основном при отладке сценариев командного интерпретатора и по умолчанию содержит строковое значение «++». Например, отладочный вывод shell-скрипта, содержащего единственную команду date, будет иметь следующий вид:

```
# bash -x /tmp/date.sh
++ date
+ echo Wed Dec 24 14:51:37 MSK 2008
Wed Dec 24 14:51:37 MSK 2008
```

Таблица 4.1. Формирующие символы первичного приглашения командного интерпретатора

Символ	Значение
\A	Текущее время в формате ЧЧ:ММ
\h	Имя хоста до первой точки
\u	Имя пользователя
\w	Текущий рабочий каталог
\$	Если эффективный идентификатор пользователя (UID) равен 0, то отображается #, в остальных случаях отображается $
\s	Название командного интерпретатора

При работе с ОС Linux в данном курсе слушатели будут в основном сталкиваться с первичным и вторичным приглашениями командного интерпретатора.

4.2. Идеология работы с командами: структура и использование

Основная операция, которая выполняется при работе с командным интерпретатором, – это ввод команд, их ключей и аргументов. К примеру, чтобы просмотреть список содержимого домашнего каталога пользователя в расширенном формате, включая и скрытые файлы, имена которых начинаются с точки (обычно это файлы настроек и каталоги файлов настроек), необходимо ввести команду **ls –la ~**[1].

[1] В оболочке bash тильда (~) используется для быстрого доступа к домашнему каталогу текущего пользователя.

По сути, команда **ls** является исполняемым бинарным файлом, расположенным в каталоге **bin**. Поэтому вместо команды **ls** можно ввести полное имя исполняемого файла, иными словами – указать абсолютный путь к файлу:

```
# /bin/ls
```

При этом часто возникает проблема: вы вводите команду, которая, как вы рассчитываете, есть в системе, но получаете сообщение, подобное «**Not found**» («Не найдена»). Проблема может быть в том, что команда размещается в таком каталоге, который интерпретатор просто не просматривает. Список каталогов, где интерпретатор отыскивает команды, называется «***путь поиска***». Текущий путь поиска содержится в переменной окружения **PATH**, для отображения которой можно использовать команду **echo**. Результатом работы команды **echo** будет вывод набора путей, разделенных двоеточиями.

Каждую команду можно условно разделить на три компонента:

- название команды;
- параметры;
- аргументы.

В предыдущем примере название команды соответствует **ls**, параметром команды является пара параметров **l** и **a**, а аргументом является символ **~**. Для того чтобы просмотреть список всех имеющихся параметров и аргументов команды, существуют специальные параметры – **-help**, **-h** и **—usage**, которые выводят на экран краткую справку по использованию команды.

Командная строка в ОС Linux обладает очень полезными средствами – автоматическим завершением названий команд и имен файлов, а также мощными возможностями редактирования команд. Все это реализовано в ОС Linux для обеспечения наибольшего быстродействия при работе с командной строкой.

При работе с командами ОС Linux необходимо придерживаться следующих общих требований.

1. Каждая команда имеет следующий ***синтаксис***:

   команда [опции] опции [аргументы] аргументы

   В квадратных скобках обозначаются необязательные опции и аргументы, без которых команда будет выполнена. Обязательные опции и аргументы обозначаются без квадратных скобок. Следует иметь в виду, что некоторые команды могут вообще не содержать обязательных опций или аргументов.
2. Каждое слово в строке отделяется от предыдущего пробелом.
3. Опции, состоящие из одного символа, начинаются, как правило, со знака «**-**».
4. Опции, состоящие из целого слова, начинаются, как правило, со знака «—».
5. Как правило, каждой опции, состоящей из целого слова, соответствует аналогичная опция, состоящая из единичного символа, например команда **df -k** равнозначна команде **df —kilobytes**. Для экономии времени рекомендуется использовать сокращенный вариант опций.
6. Единичные опции можно группировать вместе, например набор опций «`-a -b -c`» соответствует следующей комбинации опций: «`-abc`».

7. При написании нескольких команд на одной строке их необходимо разделять знаком «;».
8. Для переноса ввода команды на следующую строку, например при написании достаточно длинных команд, используется символ «\». После ввода символа «\» и нажатия клавиши ввода **<Enter>** отображается вторичное приглашение командного интерпретатора, после которого можно продолжать ввод команды.
9. Существуют специальные метасимволы, которые имеют особое значение в процессе интерпретации:
   – *Перенаправление ввода и вывода (<, >)*. Вывод любой команды может быть перенаправлен в файл. Каждая команда считывает данные из стандартного канала ввода или из файла:

```
# echo $SHELL > /tmp/shell.name
# cat < /tmp/shell.name
/bin/bash
```

   В первой строке листинга осуществляется перенаправление вывода команды **echo** в файл **/tmp/shell.name**. Встроенная команда **echo** используется для вывода всех своих аргументов на экран терминала. В качестве аргумента команды **echo** используется подстановка значения переменной **SHELL**, содержащей путь к программе текущего командного интерпретатора. Во второй строке листинга на вход команде **cat** подается содержимое файла **/tmp/shell.name**. Команда **cat**, получив данные на вход из файла **/tmp/shell.name**, выводит их на экран терминала.
   – Перенаправление стандартного вывода одной команды на стандартный ввод другой команды (|).

Основная идеология работы с командами ОС Linux заключается в том, что для решения сложной задачи используются более простые команды, объединенные между собой каналами перенаправления |. Рассмотрим следующий листинг.

```
# echo $SHELL | tr [a-z] [A-Z]
/BIN/BASH
```

В первой строке вывод команды **echo** перенаправляется на вход команды **tr**, которая осуществляет преобразование всех строчных букв в прописные.

Основные команды ОС Linux, как встроенные в командный интерпретатор bash, так и выполненные в виде отдельных бинарных файлов, условно можно разделить на группы, представленные в приложении 4.1.

4.3. Приемы работы с командной строкой: поиск команд и специальные клавиши

Командный интерпретатор предоставляет пользователю мощный инструмент работы с командами. Для удобства и быстроты работы с командами существуют специальные сочетания клавиш (**«горячие» клавиши**), которые позволяют вводить команды и перемещаться строке намного быстрее, чем при использовании

стандартных методов ввода данных. В приложении 4.2 представлены основные сочетания клавиш, используемые в командном интерпретаторе bash.

Таким образом, можно выделить следующие приемы работы с командной строкой:

- автодополнение команд;
- редактирование командной строки;
- использование «горячих» клавиш;
- поиск ранее введенных команд;
- использование псевдонимов.

Все это в совокупности определяет производительность и качество работы пользователя с системой.

Автодополнение

Для автоматического завершения названия команды используется клавиша **<TAB>**. Достаточно набрать первых 1–3 символа начала названия команды и нажать на клавишу **<TAB>** – и система автоматически подставит корректное название команды. В случае если имеются несколько команд, названия которых начинаются с одинаковых символов, система отобразит список всех доступных для автозавершения команд:

```
# ifc<TAB><TAB>
ifcfg     ifconfig
```

Автоматическое завершение имен файлов действует по тому же принципу с использованием клавиши **<TAB>**. Причем если после указания команды нажать клавишу **<TAB>** два раза, то система отобразит содержимое текущего рабочего каталога или содержимое каталога, указанного после команды:

```
# vi /etc/host<TAB><TAB>
host.conf    hosts        hosts.allow  hosts.deny
```

Следует отметить, что в списке доступных для автозавершения команд будут указаны только те команды, исполняемые файлы которых присутствуют в путях файловой системы, указанных в переменной окружения **PATH**.

Поиск ранее введенных команд

При нажатии на клавишу **<стрелка вверх>** появится предыдущая введенная команда. Клавиша **<стрелка вверх>** выполняет перемещение по буферу истории команд назад, в то время как клавиша **<стрелка вниз>** выполняет перемещение по буферу вперед, то есть к наиболее поздним введенным командам. Если нужно изменить символ в текущей строке, используется левая или правая стрелка.

Редактирование командной строки

Для редактирования командной строки есть много комбинаций «горячих» клавиш, однако иногда бывает проще отредактировать командную строку, как если бы это был текст в текстовом редакторе. Для этого в командном интерпретаторе необходимо определить режим редактирования команд:

```
# set -o vi
```

Для редактирования команд используются команды редакторов **vi** или **emacs** в зависимости от указанного параметра команды **set**. Например, если в командной строке введена следующая команда:

```
# echo arg1 arg2 arg3
```

то для удаления второго аргумента команды необходимо перейти в режим редактора **vi**, нажав клавишу **<ESC>**, а затем последовательно нажать комбинации клавиш **2b** и **dw**. После удаления второго аргумента необходимо выполнить команду, нажав на клавишу **<Enter>**.

Поиск ранее введенных команд

Еще одной приятной особенностью командного интерпретатора является ***история команд***. Под историей команд понимается список ранее введенных команд. В случае использования интерпретатора bash данный список будет содержаться в файле **~/.bash_history**. Для выполнения команд из истории команд пользователю доступны несколько вариантов, включая комбинацию клавиш **<Ctrl+R>**, команды **history** и **fc**. В первом случае поиск команд выполняется автоматически в процессе ввода первых символов команды в строке поиска (**reverse-i-search**).

Команда history выводит на экран список всех[1] введенных ранее команд. Для запуска команд из списка существуют специальные команды:

- **!!** – выполнить последнюю команду повторно;
- **!-<N>** – выполнить **<N>** последних команд;
- **!<N>** – выполнить команду **<N>** из списка команд;
- **!<string>** – выполнить последнюю команду, начинающуюся со строки **<string>**;
- N – порядковый номер команды в списке команд.

Команду **fc** удобно использовать для быстрого редактирования последней введенной команды. Данная команда имеет две формы ввода:

```
fc [-e редактор] [first] [last]
fc -s [образец=замена] [номер команды]
```

В первом случае выполняется редактирование диапазона команд в указанном редакторе[2]. Во втором случае выполняется повторный запуск команды с изменением всех **образцов** на указанную **замену**. Если номер команды явно не указан, то будет выполнена самая последняя команда с найденным **образцом**.

Использование псевдонимов

Благодаря использованию псевдонимов команд можно создать новую команду, которая будет носить сокращенное имя или выполнять схожие действия. К примеру, чтобы создать псевдоним для команды, выводящей список содержи-

---

[1] Количество запоминаемых команд в истории команд по умолчанию задается в переменной окружения **HISTSIZE**.

[2] Если редактор явно не указан, то используется редактор, указанный в переменных **FCEDIT** и **EDITOR**.

мого каталога в расширенном формате, необходимо выполнить следующую команду:

```
$ alias dir='ls -l'
```

Теперь, выполнив команду **dir**, на экране отобразится список содержимого текущего каталога в расширенном формате.

Наличие псевдонимов позволяет назначить короткие имена длинным, часто используемым командам. Чтобы узнать, какие псевдонимы были созданы в ОС Linux, необходимо выполнить команду **alias** без аргументов. Полученный список будет включать в себя несколько стандартных псевдонимов, а также только что созданный псевдоним **dir**:

```
# alias
alias cp='cp -i'
alias dir='ls -l'
alias ls='ls --color=tty'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-
tilde'
```

Следует отметить, что при выходе из текущего сеанса интерпретатора или запуска нового созданные псевдонимы уничтожаются. Во избежание данной ситуации используются конфигурационные файлы интерпретатора, о которых будет рассказано далее.

4.4. Получение справки

Одним из основных преимуществ ОС Linux является наличие встроенной справочной системы, которая содержит почти всю информацию обо всех командах, библиотеках и конфигурационных файлах.

4.4.1. Команды man и info

Основную часть справочной системы ОС Linux занимают страницы руководства **man**. Они могут быть просмотрены как в графическом, так и в текстовом режиме работы, что делает их самым быстрым источником получения необходимой справочной информации.

Страницы руководства man сгруппированы в несколько тематических разделов (табл. 4.2).

Таблица 4.2. Разделы справочного руководства man

Номер раздела	Описание
1	Команды пользовательского уровня
2	Системные вызовы и коды ошибок ядра
3	Библиотечные функции
4	Драйверы устройств и сетевые протоколы
5	Форматы файлов

Таблица 4.2. Разделы справочного руководства man (окончание)

Номер раздела	Описание
6	Игры и демонстрационные программы
7	Различные файлы и документы
8	Команды системного администрирования
9	Внутренние интерфейсы и спецификации ядра

В процессе администрирования ОС Linux и использования системы в качестве клиента наиболее полезными разделами руководства **man** являются 1 и 5 разделы.

Просмотр страниц руководства **man** осуществляется в программе просмотра текстовых файлов, заданного в конфигурационном файле **/etc/man.config**. По умолчанию в качестве программы просмотра страниц руководства **man** используется программа **less**.

Поиск справочной информации по страницам руководства **man** предельно прост. Необходимо ввести команду **man** и в качестве ее аргумента указать название команды или файла, которое необходимо найти в руководстве. В следующем листинге представлен частичный вывод справочной информации по команде **man**.

```
# man man
man(1)
NAME
    man - format and display the on-line manual pages
SYNOPSIS
    man  [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file] [-M
pathlist] [-P pager] [-B browser] [-H htmlpager] [-S section_list] [section]
name ...
```

Все действия со страницей руководства, такие как перемещение, поиск необходимого текста, форматирование страницы, осуществляются согласно установленным комбинациям клавиш программы просмотра **less**. Для просмотра допустимых комбинаций клавиш программы просмотра **less** можно воспользоваться встроенной справкой, нажав на клавишу **<h>** в режиме просмотра произвольной страницы руководства **man**.

При поиске необходимой команды может возникнуть ситуация, когда одно и то же слово присутствует в нескольких разделах. По умолчанию отображается информация из раздела с наименьшим номером. Для отображения справки из желаемого раздела необходимо после ввода команды man указать номер раздела, в котором будет осуществляться поиск. Например, для отображения справки по конфигурационному файлу **/etc/passwd** необходимо запустить команду `man 5 passwd`. Иначе будет отображена справка по команде управления паролем пользователя из первого раздела.

Как и команды, страницы руководства **man** иногда располагаются в самых неожиданных местах. Например, некоторые специфические программы могут быть установлены в каталог **/usr/local**, а их страницы справочного руководства помещены в **/usr/local/man**. Команда man не станет автоматически осуществлять поиск в **/usr/local/man**, поэтому при запросе страницы руководства можно полу-

чить сообщение **«No manual entry for...»**. Чтобы исправить положение, нужно задать все каталоги верхнего уровня в переменной окружения **MANPATH**.

```
$ export MANPATH=/usr/man:/usr/local/man
```

Синтаксис такой же, как в случае с переменной **PATH**, описанной выше: каталоги разделены двоеточиями. Есть еще одна переменная окружения, которую, возможно, придется установить, – **MANSECT**. Она определяет порядок просмотра разделов при поиске статей руководства. Например, следующая строка указывает, что поиск должен начинаться с раздела 2.

```
$ export MANSECT=«2:3:1:5:4:6:7:8:9»
```

Альтернативой страницам руководства man являются страницы руководства **info**. Основное различие данных систем заключается в том, что в системе **info** используется гипертекст, который позволяет перемещаться между страницами в интерактивном режиме. Кроме того, справочная система **info** содержит более подробное описание команд, чем руководство man. Например, для просмотра полного руководства по программе изменения параметров терминала (stty) необходимо ввести следующую команду:

```
$ info stty
```

Программа **info** сложна в восприятии и имеет массу функций для навигации. Чтобы разобраться в ней, лучше всего, находясь в программе, нажать **<Ctrl+H>** и прочесть весь список команд. Есть программы, которые упрощают чтение страниц info, например **pinfo**, работающая сходным с текстовым веб-обозревателем **lynx** образом.

В последнее время документация к различным программам ОС Linux все чаще поставляется в виде страниц HTML. Их можно читать любым веб-обозревателем. Иногда документация может распространяться в виде файлов формата PDF, которые можно просматривать либо с помощью коммерческой программы Adobe Acrobat Reader, которая входит в состав большинства дистрибутивов и которую можно загрузить с сайта разработчика, либо с помощью свободно распространяемой программы **evince**.

4.4.2. Поиск названий и описаний команд

Может так случиться, что названия некоторых команд не запоминаются или запоминаются не полностью. Существуют несколько команд, помогающих определить корректное название программы и дать краткое ее описание.

Для поиска команды, в описании которой присутствует некоторая строка текста, используется команда **apropos**. В следующем листинге приведен пример поиска всех команд, в описании которых встречается строка **download**:

```
# apropos download
ascii-xfr          (1)   - upload/download files using the ASCII protocol
CPAN               (3pm)  - query, download and build perl modules from CPAN sites
lwp-download       (1)   - Fetch large files from the web
update-pciids      (8)   - download new version of the PCI ID list
Wget [wget]        (1)   - The non-interactive network downloader
```

Если необходимо понять, зная название команды, ее использование, то используется команда **whatis**, выполняющая поиск по разделам руководства man и выдающая совпавшие результаты. В следующем листинге приведен вывод краткого описания команды **ifconfig**:

```
# whatis ifconfig
ifconfig              (8)  - configure a network interface
```

Иногда бывает необходимо просмотреть описание всех команд в некотором каталоге файловой системы, например **/usr/sbin**. Для этого можно использовать следующую комбинацию команд:

```
# cd /usr/sbin/ && ls | xargs whatis | less
```

В данном листинге представлена составная команда, состоящая из четырех команд, связанных между собой каналами перенаправления, а также условным оператором **&&**. Первая команда **cd** выполняет смену рабочего каталога на **/usr/sbin**. Далее в случае успешной смены каталога выполняется команда **ls**, выводящая имена всех файлов в каталоге. Затем вывод команды **ls** передается команде **xargs**, которая для каждого полученного имени файла выполняет команду **whatis**. Одновременно с выводом на терминал информации происходит ее перенаправление на вход команды **less**, которая просто просматривает информацию поэкранно, ожидая при этом действий пользователя для перехода к следующему экрану.

4.5. Настройка командного интерпретатора. Переменные окружения

Командный интерпретатор определяет переменные окружения, которые используются в текущем сеансе пользователя. Каждая новая введенная команда фактически запускается как ***дочерний*** процесс ***родительского*** процесса, в качестве которого в данном случае выступает командный интерпретатор **bash**.

Переменные окружения отличаются от обычных переменных тем, что они доступны как для родительских, так и для дочерних процессов.

Автоматическое определение переменных окружения происходит после аутентификации пользователя в системе. Программа **login** в случае успешного завершения процесса аутентификации на основе конфигурационного файла **/etc/passwd** определяет, какой командный интерпретатор будет использоваться в сеансе данного пользователя. После определения командного интерпретатора происходит настройка сеанса согласно конфигурационным файлам, представленным в табл. 4.3.

Все конфигурационные файлы интерпретатора **bash** являются сценариями, то есть написаны на языке интерпретатора.

Таблица 4.3. Конфигурационные файлы командного интерпретатора bash

Конфигурационный файл	Описание
/etc/profile	Определяет переменные окружения для всех пользователей системы. Данный файл выполняется при первом входе в систему и содержит основные переменные окружения, такие как

Таблица 4.3. Конфигурационные файлы командного интерпретатора bash (окончание)

Конфигурационный файл	Описание
/etc/profile	переменная поиска расположения команд **PATH**, переменная имени хоста **HOSTNAME**, переменная, определяющая размер истории команд **HISTSIZE**. Кроме того, данный файл генерирует дополнительные переменные окружения из конфигурационных файлов, находящихся в каталоге **/etc/profile.d**
/etc/bashrc	Выполняется для всех пользователей при каждом запуске командного интерпретатора **bash**. В данном файле определяется значение переменной **PS1**, а также дополнительные ***псевдонимы*** команд (alias). ***Псевдонимом*** называется сокращенное произвольно заданное название команды или последовательности команд, позволяющее выполнять сложные последовательности команд, не вводя их с клавиатуры, а вызывая через обращение к соответствующему псевдониму. Переменные, определенные в данном файле, могут быть переназначены аналогичным пользовательским файлом **~/.bashrc**, который имеет более высокий приоритет
~/.bash_profile	Содержит индивидуальные настройки пользователя. Выполняется только один раз при входе пользователя в систему. Кроме того, данный файл осуществляет запуск файла **~/.bashrc**
~/.bashrc	Содержит переменные окружения и псевдонимы, установленные пользователем. Он выполняется каждый раз при входе пользователя в систему или при открытии нового сеанса **bash**. Данный файл лучше всего подходит для определения пользовательских переменных и псевдонимов
~/.bash_logout	Выполняется каждый раз при выходе из системы или завершении последнего сеанса интерпретатора **bash**. По умолчанию в данном файле содержится команда очистки экрана терминала
/etc/inputrc	Содержит описание интерпретации различных сочетаний клавиш, а также специальные комбинации клавиш, нажатие которых вызывает выполнение заданных команд

Конфигурационные файлы, приведенные в табл. 4.3, считываются в строго определенном порядке. Этот порядок в первую очередь зависит от того, каким образом выполняется запуск оболочки: либо происходит первичный вход в систему, используя программу **login**, либо же происходит просто открытие дополнительного сеанса интерпретатора. В этих случаях принято говорить, что оболочка запускается как ***login shell*** и ***non-login shell*** соответственно. В случае использования **non-login shell** считывается только файл **~/.bashrc** из домашнего каталога пользователя. В случае использования **login-shell** сначала выполняется считывание файла **/etc/profile**, а затем файлов **~/.bash_profile**, **~/.bash_login** и **~/.profile**. После завершения сеанса оболочки выполняется файл **~/.bash_logout**.

Для изменения конфигурационных файлов **/etc/profile** и **/etc/bashrc** необходимо быть суперпользователем **root**. Обычные пользователи могут изменять

конфигурационные файлы **~/.bash_profile**, **~/.bashrc** и **~/.bash_logout**, находящиеся в их домашних каталогах.

При работе с конфигурационными файлами интерпретатора **bash** очень часто необходимо определять переменные окружения. Переменная окружения отличается от обычной переменной тем, что она доступна любой программе текущего сеанса. Как правило, переменные окружения пишутся заглавными буквами.

Интерпретатор **bash** использует большое количество переменных окружения. Список основных переменных окружения интерпретатора **bash** приведен в приложении 4.3.

Для просмотра значений переменных окружения текущего сеанса командного интерпретатора **bash** используются команды **printenv, env** и **set**. Первая из них выводит только переменные окружения. Вторая команда при вызове без параметров также показывает переменные окружения, однако используется в основном для запуска команд в измененном окружении, переопределяя текущие переменные окружения для указанной команды. Третья команда отображает все переменные текущего сеанса, включая переменные окружения, а также используется для установки опций командного интерпретатора.

Для задания дополнительных переменных окружения или перевода обычных переменных в окружение используется команда **export**. В следующем листинге приводится пример использования данной команды для экспорта переменной адреса прокси-сервера:

```
# export HTTP_PROXY=http://192.168.10.84:8080
# env | grep HTTP_PROXY
HTTP_PROXY=http://192.168.10.84:8080
```

В первой строке кода одновременно происходят определение новой переменной **HTTP_PROXY** и ее экспорт в окружение командного интерпретатора. Во второй строке кода осуществляется проверка того, что переменная **HTTP_PROXY** действительно определилась в окружении командного интерпретатора.

Примечание

Если включен автоматический режим экспорта переменных (активируется включением опции командного интерпретатора **set -a**), каждая новая определенная переменная автоматически становится переменной окружения командного интерпретатора.

Модуль 5. Работа с файловой системой ОС Linux

Изучив данный модуль, вы научитесь:

- выполнять базовые операции с файлами и каталогами;
- просматривать содержимое текстовых файлов;
- выполнять поиск файлов согласно заданным условиям поиска;
- работать с архивными файлами.

5.1. Основные операции при работе с файлами и каталогами

В ОС Linux имена файлов могут содержать символы как нижнего, так и верхнего регистров, причем регистр символов *имеет значение*. Два одинаковых имени файла, записанных в разных регистрах, будут являться физически разными файлами. В имени файла допускается использовать символы «.», «-», «_», «~» и др., однако их использование может иметь специальное значение для ОС Linux, о котором будет сказано далее.

Символ «/» недопустимо использовать в качестве имени файла, поскольку он предназначен для разделения каталогов при указании пути к файлу. Символ «\» в основном используется для переноса ввода текста на следующую строку. Использование данного символа в имени файла, как правило, заключается в экранировании пробелов между отдельными словами имени файла. В ОС Linux имена файлов могут содержать пробелы, однако их необходимо экранировать при помощи символа «\» или записывать имя файла в кавычки. Единичный символ «~» имеет особое значение и используется как ссылка на домашний каталог пользователя.

В предыдущем разделе уже упоминались специальные каталоги «.» и «..», которые используются для обращения к текущему рабочему каталогу и родительскому каталогу соответственно. При использовании символа «.» в качестве первого символа имени файла данный файл будет скрыт для обычного просмотра, а также для команды удаления, если при запуске данной команды не была указана соответствующая опция.

Длина имени файла в ОС Linux зависит от типа файловой системы, на которой располагается данный файл. В файловой системе **ext3**, используемой в ОС Linux по умолчанию, максимальная длина имени файла не может превышать 255 символов.

Следующие команды являются базовыми при проведении повседневных операций администрирования ОС. Эти команды позволяют просматривать, копировать, перемещать, переименовывать и удалять файлы.

5.1.1. Команды управления файлами

Для просмотра файлов в ОС Linux используется команда **ls**, имеющая следующий синтаксис:

```
ls [опции]... [имя]...
```

опции – параметры, позволяющие форматировать вывод команды;
имя – имя каталога, содержимое которого необходимо отобразить.

Чаще всего используются, при работе с командой **ls**, следующие опции:

- **-l (--format=long)** – расширенный вывод содержимого каталога, в котором отображаются параметры разрешений, имена владельца и группы, дата создания, размер и прочие важные параметры файла;
- **-a (--all)** – отображение всех файлов, включая скрытые файлы, начинающиеся с символа «.»;
- **-d (--directory)** – вывод информации о каталоге без вывода его содержимого;
- **-F (--classify)** – отображение типа файла. При использовании данной опции к каждому отображаемому объекту файловой системы добавляется соответствующий суффикс: / (для каталога), @ (для символьных ссылок), = (для сокетов), | (для именованных каналов), * (для исполняемых файлов);
- **-t (--sort=time)** – отсортированный вывод файлов в соответствии со временем последнего изменения.

Опции и аргументы команды **ls** являются необязательными, при запуске команды **ls** без параметров на экран будет выведено содержимое текущего каталога в кратком формате.

В качестве аргументов команде **ls** можно указывать сразу несколько каталогов, тогда на экран будет осуществлен вывод содержимого каждого указанного каталога.

Для копирования файлов в ОС Linux используется команда **cp**, имеющая следующий синтаксис:

```
cp [опции] источник место_назначения
```

источник – объект файловой системы, который необходимо скопировать;
место_назначения – целевой объект (например, каталог) файловой системы, куда осуществляется копирование.

Обычно в качестве источника используются один или несколько файлов, а в качестве приемника – каталог, однако возможно копирование одного файла в другой при указании в качестве источника и приемника соответствующих имен файлов.

Команда **cp** имеет достаточно большое количество опций. Для просмотра всех опций можно воспользоваться справочным руководством **man**, запустив команду **man cp**.

Наиболее часто используемыми при работе с командой **cp** являются следующие опции:

- **f (--force)** – для принудительной замены всех существующих файлов в месте назначения без предупреждения пользователя;
- **i (--interactive)** – для подачи запроса пользователю перед перезаписью существующего файла в месте назначения;
- **p (--preserve)** – для сохранения разрешений, прав владения и отметок времени оригинального файла источника;
- **r (--recursive)** – для рекурсивного копирования каталогов;
- **u (--update)** – для отмены копирования файлов в место назначения с таким же или более поздним временем изменения.

Если при запуске команды **cp** не указать опцию **-p**, то атрибуты файлов в месте назначения будут заново созданы в соответствии с атрибутами пользователя, который запустил команду **cp**.

Для перемещения и переименования файлов в ОС Linux используется команда **mv**. Следует заметить, что в ОС Linux *операции перемещения и переименования не различаются*.

Команда **mv** имеет следующий синтаксис:

```
mv [опции] исходное_размещение целевое_размещение,
```

`опции` – разнообразные параметры, позволяющие управлять процессом копирования;
`исходное_размещение` – исходное размещение объекта файловой системы, который необходимо переместить или переименовать;
`целевое_размещение` – целевой объект файловой системы, куда осуществляется перемещение.

Значения опций команды **mv**, таких как **-u**, **-f** и **–i**, аналогичны значениям таких же опций команды **cp**.

Таблица 5.1. Описание работы команды mv

Исходное размещение	Целевое размещение	Результат выполнения команды
Файл	*Имя*	Переименовать файл, назначив ему заданное *имя*
Файл	Существующий файл	Заменить существующий файл исходным файлом
Каталог	*Имя*	Переименовать каталог, назначив ему заданное *имя*
Каталог	Существующий каталог	Переместить каталог таким образом, чтобы он стал подкаталогом существующего каталога
Несколько файлов	Существующий каталог	Переместить файлы в каталог

В следующем листинге приведены примеры использования команды **mv**.

```
# mv -v 1.txt 2.txt
'1.txt' -> '2.txt'
# mv -v 2.txt /tmp/
'2.txt' -> '/tmp/2.txt'
```

В первой команде производится простое переименование файла **1.txt** в файл **2.txt**. Опция **-v** указана для вывода дополнительной служебной информации. Во второй команде производится перемещение файла **2.txt** в каталог **/tmp**. Обратите внимание на то, что в конце каталога **/tmp** указан символ «/». Это означает, что под целевым размещением подразумевается именно каталог, а не файл. Если конечный символ «/» не указан и целевой каталог назначения был по каким-то причинам указан неверно, команда **mv** выполнит переименование файла **2.txt** в неверно указанное имя файла.

Удаление файлов в ОС Linux осуществляется при помощи команды **rm**, имеющей следующий синтаксис:

```
rm [опции] файлы
```

файлы – объекты файловой системы, которые необходимо удалить.

Чтобы удалить файл при помощи команды **rm**, пользователь должен иметь право работать с каталогом, содержащим данный файл, однако ему не обязательно иметь разрешение на работу с самим файлом, который он хочет удалить.

Наиболее часто используются при работе с командой **rm** следующие опции:

- **f (--force)** – для принудительного удаления файлов без вывода запросов на подтверждение пользователя;
- **i (--interactive)** – для подачи запроса подтверждения пользователю перед удалением файлов;
- **r (--recursive)** – для рекурсивного удаления содержимого каталога и самого каталога.

Работать с командой **rm** надо *предельно внимательно*, поскольку при работе в командном режиме не существует понятия «корзины», и файлы, удаленные данной командой, можно восстановить только из резервных копий (если таковые были своевременно сделаны) или с помощью низкоуровневых утилит восстановления файловой системы, таких как **debugfs**.

В файловой системе **ext3**, используемой в ОС Linux по умолчанию, каждый файл имеет следующие временные атрибуты:

- время создания файла;
- время последнего изменения файла;
- время последнего доступа к файлу.

Иногда может возникнуть необходимость изменения временных атрибутов файлов или просто создания регулярных файлов. Для таких целей в ОС Linux существует команда **touch**, которая имеет следующий синтаксис:

```
touch [опции] [дата] файлы
```

дата – дата и время, которую необходимо присвоить файлу;
файлы – файлы, которые необходимо изменить.

Если параметр дата не указан, то доступ будет использовать текущее значение. Изменение временных атрибутов файлов может быть использовано, например, при компилировании программ командой **make** из исходных кодов, поскольку данная команда использует временные атрибуты файлов исходных кодов для определения того, какие файлы будут перекомпилированы, в случае наличия соответствующего объектного файла для данного файла, содержащего исходный код.

Если указанный файл не существует, команда **touch** создаст пустой файл с таким же именем. Этот механизм используется для организации блокировок, например при одновременном доступе процессов к одному файлу.

Чаще всего используются при работе с командой **touch** следующие опции:

- **a (--time=atime)** – для указания времени доступа;
- **m (--time=mtime)** – для указания времени изменения;
- **c (--no-create)** – для указания не создавать файл, если он не существует;
- **t** – для указания значения времени в формате [[гг]гг]ммддччмм[.сс];
- **r *файл* (--reference=файл)** – для указания значения времени из указанного ***файла***.

5.1.2. Команды управления символьными ссылками

Как было сказано ранее, в ОС Linux существуют символьные и жесткие ссылки. Для создания ссылок используется команда **ln**, имеющая два варианта синтаксиса:

```
ln [опции] файл имя_ссылки
```

либо

```
ln [опции] файлы каталог
```

файл, файлы – целевой файл (или файлы), на который необходимо создать ссылку;
имя_ссылки – имя (псевдоним) ссылки;
каталог – каталог, в котором необходимо создать ссылку.

В первом варианте команда **ln** создает ссылку с именем имя_ссылки на указанный файл. Во втором варианте, если указан существующий каталог, программа **ln** создает ссылку на каждый файл в указанном каталоге.

Наиболее часто используются при работе с командой **ln** следующие опции:

- **s (--symbolic)** – создать символьную ссылку;
- **f (--force)** – принудительное создание ссылки без запроса на перезапись;
- **d (--directory)** – используется создание жестких ссылок на каталоги (работает только для привилегированных пользователей);
- **b (--backup)** – создать резервные копии файлов;
- **n (--no-dereference)** – заменять символьные ссылки на каталоги вместо ликвидации ссылок.

При использовании команды **ln** без опций по умолчанию будет создаваться жесткая ссылка на указанный файл.

5.1.3. Команды управления каталогами

Большинство команд, которые применимы к файлам, подходят и для управления каталогами. В частности, команды **ls**, **mv**, **touch** и **ln** работают с каталогами так же, как и с файлами. Команда **cp** работает с каталогами при указании опции **-r** для рекурсивного копирования каталогов.

Однако в ОС Linux, помимо описанных выше команд, существуют команды **cd**, **pwd**, **mkdir** и **rmdir**, работающие только с каталогами.

Команда **cd** является встроенной[1] командой командного интерпретатора **bash** и используется для смены текущего рабочего каталога. Синтаксис данной команды следующий:

```
cd [опции] [каталог]
```

Команда **pwd** является также встроенной командой и используется для отображения текущего рабочего каталога. Синтаксис данной команды следующий:

```
pwd [опции]
```

Для создания каталогов в ОС Linux существует команда **mkdir**, которая имеет следующий синтаксис:

```
mkdir [опции] каталоги
```

Для создания каталогов необходимо иметь соответствующие разрешения в родительском каталоге. Наиболее часто используемыми при работе с командой **mkdir** являются следующие опции:

- **m (--mode режим)** – установить режим доступа к новым каталогам;
- **p (--parents)** – создать промежуточные каталоги, если они не существуют;
- **v (--verbose)** – выводить имена каталогов при их создании.

Для удаления пустых каталогов в ОС Linux существует команда **rmdir**, которая имеет следующий синтаксис:

```
rmdir [опции] каталоги
```

Для удаления каталогов необходимо иметь соответствующие разрешения в родительском каталоге. Чаще всего при работе с данной командой **rmdir** используются следующие опции:

- **p (--parents)** – удалять заданные каталоги и все промежуточные пустые каталоги;
- **--ignore-fail-on-non-empty** – игнорировать отказ удаления непустых каталогов;
- **v (--verbose)** – выводить сообщения о каждом удаленному каталоге.

---

[1] Встроенные команды работают быстрее, чем внешние, поскольку их код содержится непосредственно в коде командного интерпретатора.

5.1.4. Команды определения типов файлов и дополнительной информации

Одной из самых часто используемых команд определения типа файла является команда **ls**, о которой речь шла выше. Для определения типа файла достаточно выполнить команду **ls -l**, в результате выполнения которой на терминале первый символ каждой строки будет указывать на тип файла. В следующем листинге приведен вывод первых трех строк команды **ls**:

```
# ls -l|head -3
total 22864
-rwxr-xr-x 1 root root     9684 Aug  6  2007 accept
lrwxrwxrwx 1 root root       17 May 14 19:46 accton -> ../../sbin/accton
```

Как видно, первым символом во второй строке листинга является «–», что говорит о том, что файл accept является регулярным. В четвертой строке листинга первым символом строки является «**l**», следовательно, файл **accton** является ссылкой.

Иногда необходимо более детально определить тип файла, например если это регулярный файл. Для этого используется команда **file**, имеющая следующий синтаксис:

```
file [опции] -f [файл] -m [файл] файлы ...
```

Программа **file** тестирует каждый заданный `файл`, пытаясь классифицировать его. Прежде всего устанавливается тип файла. Часто для этого приходится проверять **magic numbers** (магические числа): первые шестнадцать бит файла содержат кодированную информацию о типе этого файла. С помощью файла **/etc/magic** или **/usr/share/magic** ядро системы интерпретирует кодированную информацию, пытаясь определить тип файла.

Если был обнаружен ASCII-файл, команда также пытается определить язык текстового файла, анализируя строки нескольких первых блоков файла.

Наиболее часто используемыми при работе с командой **file** являются следующие опции:

- **b** – не выводить имена проверяемых файлов;
- **i** – отображает mime-тип файла (text/plain и прочее);
- **z** – проверка сжатых файлов;
- **s** – проверка специальных файлов;
- **L** – следует по символьным ссылкам.

В параметре ключа **–f** указывается файл, содержащий список файлов для проверки. В параметре ключа **–f** указывается файл, содержащий магические числа. Далее следуют непосредственно аргументы, содержащие набор файлов для проверки. В следующем листинге приведены примеры определения типа нескольких файлов и степени детализации выводимой информации:

```
# file -s /dev/sda
/dev/sda: x86 boot sector; partition 1: ID=0x83, active, starthead 1,
```

```
startsector 63, 208782 sectors; partition 2: ID=0x8e, starthead 0,
startsector 208845, 10265535 sectors, code offset 0x48
# file /dev/sda
/dev/sda: block special (8/0)
# file -i /etc/resolv.conf
/etc/resolv.conf: text/plain; charset=us-ascii
# file /etc/resolv.conf
/etc/resolv.conf: ASCII text
# file -b /usr/include/stdio.h /bin/bash /var/log/messages
ASCII C program text
ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.6.9,
dynamically linked (uses shared libs), for GNU/Linux 2.6.9, stripped
ASCII English text
# file -z  /usr/share/man/man1/perl.1.gz
/usr/share/man/man1/perl.1.gz: troff or preprocessor input text (gzip
compressed data, from Unix, max compression)
```

В первом вызове команды **file** используется ключ **–s**, позволяющий детализировать информацию по файлам устройств. В третьем вызове команды **file** приведен пример определения mime-типа файла и его кодировки. В пятом вызове команды **file** выполняется проверка трех файлов без указания файлов в выводе команды. В последнем вызове команды **find** выполняется проверка содержимого архива **perl.1.gz**.

Помимо определения типа файла, иногда бывает необходимо определить такие параметры, как время последнего доступа к файлу, владельца и группу файла, индексный дескриптор (инод) и устройство, на котором располагается файл. Для этих задач можно использовать команду **ls**, однако более удобно применять команду **stat**, характеризующую файл с точки зрения файловой системы. Команда stat имеет следующий синтаксис:

```
stat [опции] файлы ...
```

Наиболее часто используемыми при работе с командой **sat** являются следующие опции:

- **f** – вывести информацию о файловой системе;
- **c** – использовать специальный формат отображения;
- **L** – следует по символьным ссылкам.

В следующем листинге приведен пример использования команды **stat**:

```
# stat /root/.bashrc
  File: '/root/.bashrc'
  Size: 176             Blocks: 16         IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 704006      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2009-08-28 11:12:52.000000000 +0400
Modify: 2006-07-13 04:06:42.000000000 +0400
Change: 2009-05-14 19:41:59.000000000 +0400
```

Из листинга можно определить индексный дескриптор файла **(Inode)**, размер **(Size)**, а также количество жестких ссылок на данный файл **(Links)**.

5.2. Просмотр содержимого файлов: утилиты more и less

Для просмотра файлов или большого объема тестовых данных в ОС Linux существуют утилиты, относящиеся к классу ***пейджеров*** (от лат. *page* – страница). Довольно часто данных, выводимых определенной командой, оказывается слишком много, чтобы уместиться на одном экране. Отдельные команды не знают, как уместить свои выходные данные на нескольких страницах. Эту работу они оставляют утилитам-пейджерам. Основными утилитами данного класса являются **more** и **less**.

Команда **more** разбивает выходные данные на отдельные экраны и ожидает нажатия на клавишу **<Space>**, перед тем как вывести следующую порцию. Нажатие на клавишу **<Enter>** сдвинет вывод только на одну следующую строку, а не на целую страницу. Опции команды **more** могут извлекаться из переменной окружения **MORE**, однако опции командной строки имеют превосходство над этой переменной. Команда **more**, как и другие команды этого класса, используются в интерактивном режиме работы. Синтаксис команды прост:

```
more [опции] файлы ...
```

При указании нескольких файлов на экран будут последовательно выводиться фрагменты всех файлов. Наиболее часто используемыми при работе с командой **more** являются следующие команды:

- **<Space>** – прокрутка следующего экрана;
- **<Enter>** – прокрутка на одну строку вниз;
- **v** – запуск редактора vi в текущей строке текста;
- **:n** – переход к просмотру n-го файла;
- **:k** – переход к просмотру k-го предшествующего файла;
- **:f** – отображение имени текущего файла и номера строки;
- **.** – повтор предыдущей команды;
- **q** – завершение работы программы **more**.

В следующем листинге приведен пример работы с командой more для поэкранного отображения содержимого каталога **/usr/sbin**:

```
# cd /usr/bin/
# ls -l|more
total 156288
-rwxr-xr-x 1 root root        8104 Oct  2  2006 411toppm
-rwxr-xr-x 1 root root      111148 Oct  3  2006 a2p
-rwxr-xr-x 1 root root       20484 Jul 13  2006 ac
...
-rwxr-xr-x 1 root root        3100 Aug  9  2006 amuFormat.sh
--More--
```

Экраном в данном случае называется весь текст, отображаемый вертикально от начала до конца видимой области терминала (до надписи **--More--**).

Следует отметить, что команда **more** не предоставляет возможности прокрутки файла назад.

Помимо команды more, в ОС Linux существует более мощная команда прокрутки файлов – **less**.

Программа **less** не считывает вначале весь входной файл полностью, как это делают другие текстовые редакторы, например **vi**, поэтому даже в случае больших входных файлов она стартует значительно быстрее.

С помощью команды – может быть изменено большинство опций, когда **less** уже запущена. Большинство опций может быть задано в двух форматах: дефис, за которым следует единственная буква, или же два дефиса, за которым следует длинное имя опции. Многие опции также выбираются из переменной окружения **LESS**, хотя опции командной строки имеют больший приоритет. Синтаксис команды **less** следующий:

```
less [опции] файлы ...
```

С помощью **less** можно просматривать несколько файлов одновременно, а также определять шаблоны файлов для открытия всех подходящих файлов:

```
# less /usr/share/doc/*.txt
```

В отличие от своего предшественника – программы **more**, **less** умеет просматривать файлы в обратном направлении. Кроме этого, программа **less** обладает удобными средствами для поиска и навигации по тексту, хорошо интегрирована с оболочкой, отлично справляется с большими файлами и очень гибка в настройке. Последние версии программы less позволяют просматривать не только текстовые файлы, но также файлы формата **pdf** и архивы.

Наиболее часто используемыми при работе с командой **less** являются следующие команды:

- **h** – вызов справки;
- **q** – выход;
- **<Space>** – на экран вперед;
- ← и → – горизонтальная прокрутка;
- ↑ и ↓ – вертикальная прокрутка (аналогично j и k);
- **F** – просмотр растущего файла (аналог tail -f);
- **<N>g** – перейти на строку N (по умолчанию 1);
- **<N>%** – перейти на позицию N% (к примеру, 50%);
- **/pattern** – поиск по шаблону вперед;
- **?pattern** – поиск по шаблону назад;
- **n** – следующее совпадение;
- **N** – предыдущее совпадение;
- **v** – запуск редактора (переменная окружения **EDITOR**);
- **!** – запуск shell-команды (**%** – имя текущего файла, к примеру **! cat % > /tmp/foobar.txt**).

Опции команды **less** задают различные режимы работы программы. Вот некоторые из них:

- **I** – игнорировать регистр при поиске;
- **J** – включить столбец статуса слева экрана;

- **N** – отображать номера строк в столбце статуса;
- **S** – усекать длинные строки.

Повторное указание опции в интерактивном режиме отключает ее действие.

5.3. Поиск файлов: утилиты find и locate

Поиск файлов в ОС Linux осуществляется при помощи утилит **find** и **locate**, входящих в состав пакета **findutils**. Часто бывает необходимо выполнить какое-либо действие над найденными файлами, для этого в пакете **findutils** присутствует утилита **xargs**. В дополнение к этим командам существует команда **updatedb**, позволяющая обновлять базу файлов, использующуюся утилитой **locate**.

Команда find выполняет поиск файлов в указанной иерархии каталогов и выводит информацию о найденных файлах. Команда имеет следующий синтаксис:

```
find [каталог...] [условие]
```

В следующем листинге приведен простой пример использования команды find. Для поиска файлов в иерархии каталогов **/usr/src**, оканчивающихся на **.c** и размер которых больше 100 Кб:

```
# find /usr/src -name '*.c' -size +100k -print
```

Отметим, что поиск утилитой **find** выполняется **рекурсивно** по всей указанной иерархии каталогов до самого последнего уровня вложенности.

Утилита **locate** выполняет поиск файлов в специальной базе данных согласно заданному образцу и имеет следующий синтаксис:

```
locate [опции...] [образец]
```

В следующем листинге приведен пример поиска файлов, оканчивающихся на **Makefile** или **makefile**.

```
# locate '*[Mm]akefile'
```

Утилита **xargs** предназначена для комбинирования аргументов команды, построения команды и ее выполнения. В большинстве случаев аргументами команды **xargs** является список файлов, полученный с помощью команды **find**. Команда **xargs** имеет следующий синтаксис:

```
xargs [опции...] [команда [начальные аргументы]]
```

В следующем листинге приведен пример поиска файлов в списке **list** и вывода на экран только тех, чьи строки содержат слово **log**:

```
# xargs grep log < list
```

В команде **find** существует большое количество условий поиска файлов, а также возможных вариантов действий над найденными файлами. Условия и действия можно группировать вместе при помощи логических операторов.

Наиболее часто используются следующие условия поиска файлов.

Общие условия поиска

- **L** – следовать по ссылкам при поиске файлов (по умолчанию следование по ссылкам не выполняется);

- **daytime** – производить поиск, используя временные условия поиска, с единицами измерения в днях, а не в часах;
- **(max,min)depth** <уровень> – производить поиск, в иерархии каталогов начиная с минимального уровня (**mindepth** <уровень>) и заканчивая максимальным (**maxdepth** <уровень>) уровнем;
- **mount** – выполнять поиск файлов только на одной файловой системе.

Условия поиска файлов по имени

- **(i)name**[1] <образец> – условие выполняется, если основное имя файла соответствует указанному образцу;
- **(i)path** <образец> – условие выполняется, если полное имя файла (без учета символов «.» и «/») соответствует указанному образцу.

Временные условия поиска

- **(a,c,m)time** <(+-)n> – условие выполняется, если доступ к файлу выполнялся n*24[2] часа назад, статус файла был изменен n*24 часов назад или же файл был изменен n*24 часов назад;
- **(a,c,m)min** <(+-)n> – условие выполняется, если доступ к файлу выполнялся n*24[3] минут назад, статус файла был изменен n*24 минут назад или же файл был изменен n*24 минут назад;
- **(a,c,m)newer** <файл> – условие выполняется, если доступ к файлу выполнялся ранее, чем доступ к указанному файлу, статус файла был изменен ранее, чем статус указанного файла, или же файл был изменен ранее, чем указанный файл;
- **used** <n> – условие выполняется, если к файлу был доступ n дней назад с момента последнего изменения его статуса.

Условия поиска по размеру

- **size** <n[bckwMG]> – условие выполняется, если файл занимает n указанных единиц дискового пространства (по умолчанию единицей является блок размером 512 байт);
- **empty** – условие выполняется, если файл имеет нулевой размер.

Условия поиска по типу файла

- **f** – условие выполняется, если файл является регулярным;
- **b** – условие выполняется, если файл является блочным устройством;
- **l** – условие выполняется, если файл является ссылкой.

Условия поиска по владельцу файла и коду доступа

- **user** <пользователь> – условие выполняется, если файл принадлежит указанному пользователю;

---

[1] Здесь и далее начальный символ **i** используется для отмены учета регистра символов.
[2] n=0 соответствует времени менее 24 часов.
[3] n=0 соответствует времени менее 24 часов.

- **group** <группа> – условие выполняется, если файл принадлежит указанной группе;
- **nouser** – условие выполняется, если файл принадлежит несуществующему пользователю;
- **nogroup** – условие выполняется, если файл принадлежит несуществующей группе;
- **uid(gid)** <n> – условие выполняется, если файл принадлежит указанным идентификаторам пользователя или группы;
- **perm <**[-/][xxx]> – условие выполняется, если в коде доступа файла (три символа xxx) содержатся все указанные три бита (знак **-**) или любые указанные из трех битов (знак /).

После того как будут выбраны необходимые условия, необходимо их сгруппировать, используя логические операторы, а затем указать действие, которое необходимо выполнить над найденными файлами. Для группировки условий используются операторы «И» (условие1 –**a** условие2), «ИЛИ» (условие1 –**o** условие2) и «НЕ» (**!** условие). Для указания приоритета выполнения операторов используются круглые скобки **(** условие1 **)**.

Среди стандартных действий используются следующие действия:

- **print** – вывести список полных имен файлов по каждому на строке;
- **ls** – вывести список файлов в расширенном формате команды **ls**;
- **exec** <команда, включающая комбинацию '{}'> – выполнить указанную команду последовательно над каждым найденным файлом. В качестве шаблона имени файла используется комбинация **'{}'**;
- **execdir <**команда> **{}**+ – выполнить указанную команду одновременно над всеми найденными файлами. Команда выполняется в поддиректории относительно найденных файлов;
- **delete** – удалить найденные файлы.

Добавление комплексных расширенных проверок возможно при помощи утилиты **xargs**, однако встроенные команды утилиты **find** работают быстрее.

В следующем листинге приведен пример выполнения комплексной проверки типа исполняемых файлов в каталоге **/usr/sbin** и вывода полных имен найденных файлов:

```
# find /usr/sbin -type f -perm /a=x | xargs file | grep 'not stripped' | cut
-d: -f1
/usr/sbin/tzdata-update
```

Как видно, вначале выполняется поиск всех регулярных исполняемых файлов, затем при помощи команды **xargs** формируется новая команда (**file** файл1 файл2 ...), вывод которой обрабатывается командой **grep** для поиска файлов, содержащих строку **'not stripped'**. Далее выполняется команда **cut**, которая выводит на терминал только первое поле (в качестве разделителя используется символ «**:**») каждой строки.

В отличие от утилиты **find**, утилита **locate** выполняет поиск файлов в заранее определенном месте – собственной базе данных (по умолчанию **/usr/local/var/locatedb**).

Наиболее часто используемыми при работе с командой **locate** являются следующие:

- **d** <путь> – использовать иную базу данных для поиска файлов;
- **i** – игнорировать регистр образца поиска;
- **l <N>** – ограничить вывод первыми N найденными файлами;
- **w** – точное совпадение с образцом поиска;
- **r** – использовать базовые регулярные выражения для указания образца поиска;
- **b** – использовать первое правое вхождение указанного образца до первого символа «/».

Периодически необходимо выполнять обновление базы данных **locate** в целях получения актуальных сведений о файлах. Для этого используется утилита **updatedb**.

В следующем листинге приведен пример поиска файлов **resolv.conf**:

```
# locate -r /resolv.conf$
/etc/resolv.conf
/opt/resolv.conf
/opt/restore/aivanov/resolv.conf
```

В данном случае выполняется поиск по регулярному выражению, то есть образцу будут удовлетворять все файлы, оканчивающиеся строго на **/resolv.conf$**.

Следует отметить, что поиск файлов утилитой **locate** выполняется быстрее, чем утилитой **find**, за счет отсутствия просмотра данных на файловой системе.

5.4. Работа с архивами

При установке или обновлении программного обеспечения в ОС Linux необходимо уметь хорошо работать со средствами сжатия и архивирования файлов. В системе имеются десятки утилит данного класса. Некоторые из них (такие как **tar** и **compress**) относятся по времени создания к самым ранним версиям UNIX. Другие, такие как **gzip** и более новая **bzip2**, являются относительно новыми. Основной задачей данных утилит являются ***архивирование файлов*** (то есть упаковка группы файлов в один для удобства перемещения и создания резервных копий) и сжатие файлов для уменьшения дискового пространства, занимаемого файлом или группой файлов.

В данном разделе рассматриваются наиболее часто встречающиеся форматы файлов и утилиты, с которыми предстоит работать. В среде UNIX общепринято перемещать файлы или программное обеспечение в виде **tar-архивов**, сжатых с помощью утилит **compress**, **gzip** или **bzip2**. Для того чтобы иметь возможность самостоятельно создавать или распаковывать такие файлы, нужно уметь пользоваться соответствующими инструментами. Эти инструменты чаще всего применяются при установке нового программного обеспечения или создании резервных копий. Пакеты, пришедшие из других сред, таких как Windows или Java, часто архивируются и сжимаются с помощью утилит **zip** или **rar**; такие архивы распаковываются с помощью команд **unzip** и **unrar**, которые имеется в большинстве инсталляций ОС Linux.

5.4.1. Утилиты bzip и gzip2

Утилита **gzip** упаковывает перечисленные файлы в командной строке и существенно уменьшает их размер. При выполнении упаковки каждый файл замещается сжатой версией с таким же именем, как у оригинального, к которому добавляется расширение **.gz**; сохраняются владелец, группа, полномочия, а также временные метки оригинала. Если имена файлов не указаны или вместо имени стоит **дефис**, содержимое стандартного канала ввода упаковывается и пересылается на стандартный канал вывода. Команда **gzip** только сжимает обычные файлы, игнорируя символьные ссылки.

Следует обратить внимание на то, что в случае удачного создания архивного файла оригинальный файл удаляется.

Команда **gzi**p имеет следующий синтаксис:

```
gzip [опции...] [файлы ...]
```

Наиболее часто используемыми при работе с командой **gzip** являются следующие опции:

- **l** – просмотр содержимого архива и степени сжатия;
- **d** – использовать режим распаковки (аналог команды **gunzip**);
- **N** – сохранять оригинальные имена файлов при сжатии или распаковке;
- **r** – выполнять рекурсивную обработку каталогов;
- **t** – выполнить проверку файла на наличие программных ошибок сжатия;
- **1–9** – степень сжатия файлов;
- **c** – вывести содержимое файла в стандартный канал вывода.

В следующем листинге приведен пример сжатия текстового файла **master.cf**, а затем его просмотр при помощи утилиты **less**. Содержимое и степень сжатия отображены в последней строке листинга:

```
# gzip master.cf
# gzip -dc master.cf.gz |less
...
# gzip -l master.cf.gz
  compressed        uncompressed  ratio uncompressed_name
        1547                5377  71.8% master.cf
```

В ОС Linux существует утилита **zless**, позволяющая просматривать архивные файлы непосредственно.

bzip2 – это новая программа, сжимающая в среднем на 10–20% лучше, чем **gzip**, за счет большей продолжительности процесса сжатия. Нельзя использовать утилиту **bunzip2** для декомпрессии файлов, сжатых **gzip**, и наоборот. Узнать файлы, сжатые при помощи утилиты **bzip2**, можно по расширению **.bz2**. Опции утилиты **bzip2** схожи с опциями утилиты **gzip**:

- **c** – вывести содержимое файла в стандартный канал вывода;
- **d** – использовать режим распаковки (аналог команды **gunzip**);
- **k** – не удалять оригинальные файлы после завершения создания архива;
- **t** – выполнить проверку файла на наличие программных ошибок сжатия;
- **1–9** – степень сжатия файлов.

5.4.2. Использование утилиты tar

Tar является универсальной утилитой архивирования, способной упаковать несколько файлов в один архивный файл, сохраняя при этом данные, необходимые для полного восстановления, такие как права доступа и владения. Название **tar** происходит от «tape archive», поскольку первоначально эта утилита предназначалась для архивирования файлов в виде резервных копий на магнитных лентах.

Команда tar имеет следующий синтаксис:

```
tar <действие> [опции...] файлы...
```

Здесь <действие> является буквой, указывающей выполняемую операцию, опции является списком однобуквенных параметров этого действия, а файлы – списком упаковываемых или распаковываемых файлов в архиве. Допускается опции указывать слитно с действием.

Параметр <действие> может принимать следующие значения:

- **c** – создать новый архив;
- **x** – извлечь файлы из архива;
- **t** – перечислить содержание архива;
- **r** – дописать файлы в конец архива;
- **u** – заменить файлы в архиве более новыми;
- **d** – сравнить архивированные файлы с файлами файловой системы.

Чаще других используются действия **c**, **x** и **t**.

Обычно используются следующие значения опций:

- **k** – сохранить при разархивации существующие файлы, то есть не заменять имеющиеся файлы извлекаемыми из архива;
- **f** <имя_файла> – задать имя архивного файла, который нужно прочесть или записать;
- **z** – указать, что записываемые в архив файлы или имеющиеся в нем сжимаются с помощью **gzip**;
- **j** – аналогично **z**, но вместо **gzip** используется **bzip2**;
- **v** – заставить **tar** показать имена файлов, помещаемых в архив или извлекаемых из него. Используется в основном для отладки сценариев, в которые входит утилита **tar**.

Помимо указанных параметров, утилита **tar** имеет еще несколько десятков, информацию о которых можно узнать на страницах руководства man.

В следующем листинге приведен пример создания архива каталога **log** утилитой **tar**:

```
# tar cvf log_arch.tar log/
log/
log/messages
...
log/httpd/
log/rpmpkgs.3
log/conman.old/
```

Как видно, в случае использования ключа **v** утилита **tar** отображает обрабатываемые файлы. Если использовать двойной ключ **vv**, то будет отображена дополнительная информация, включающая временные параметры файлов и разрешения.

Разархивировать созданный архив можно, используя следующую команду:

```
# tar xvf log_arch.tar
log/
log/messages
log/httpd/
log/rpmpkgs.3
log/conman.old/
```

В результате выполнения данной команды в текущем рабочем каталоге создастся каталог **log**, в который поместятся все исходные файлы с теми же правами доступа, которые были в исходной системе. Владельцем новых файлов будет пользователь, выполнивший команду **tar xvf**, если только разархивирование не выполнил **root**. В последнем случае сохраняется первоначальный владелец файла. Параметр **x** означает извлечение файлов, параметр **v** используется для перечисления всех извлекаемых файлов. Как можно заметить, **tar** сохраняет имя пути каждого файла относительно того расположения, где первоначально был создан архивный файл. Когда создавался архив командой **tar cvf log_arch.tar log/**, единственным заданным именем входного файла было **log** – имя содержащего файлы каталога. Поэтому **tar** записывает в архивный файл сам каталог и все находящиеся в нем файлы.

По умолчанию **tar** извлекает из архива все файлы относительно текущего каталога, в котором запускается **tar**.

Утилита **tar** позволяет извлекать из архива отдельные файлы. Для этого используется команда

```
#tar xvf file.tar файлы ...,
```

где аргумент файлы является списком извлекаемых из архива файлов. Если не указывать имен, **tar** распаковывает весь архив.

Утилита **tar** не осуществляет компрессии данных, сохраняемых в архиве. Если создается tar-архив из трех файлов по 500 Кбайт, то получите архив размером около 1500 Кбайт. Очень часто rar-архивы сжимаются с помощью утилит **gzip** или **bzip2**. Сжатый tar-архив можно создать, выполнив следующую команду:

```
#tar cvf - файлы ... | gzip -9 > файл.tgz
```

В данном листинге создается архив из файлов с перечисленными именами, и программа выводит его на стандартный канал вывода; затем утилита **gzip** считывает данные из стандартного канала ввода, сжимает их и выводит в стандартный канал вывода; далее выполняется перенаправление сжатого архивного файла в файл **.tgz**.

Для распаковки данного файла можно использовать следующую команду:

```
#tar zxvf файл.tgz
```

Также существует более упрощенный способ создания сжатого по алгоритму **bzip2** tar-архива:

```
#tar cvjf файл.tbz файлы
```

Модуль 6. Обработка текстовых данных

Изучив данный модуль, вы научитесь:

- форматировать текст в требуемом формате;
- просматривать и изменять содержимое текстовых файлов, используя утилиты sed и awk;
- выполнять поиск текстовых данных в файлах согласно заданным условиям;
- сравнивать текстовые файлы.

6.1. Базовые операции с текстом: утилиты обработки текста

ОС Linux предоставляет пользователю неограниченные возможности по способам обработки и форматирования текста. При этом для выполнения любой задачи используется основной принцип работы с командами в ОС Linux: любую задачу можно разбить на несколько более простых. Все утилиты, присутствующие в ОС Linux и предназначенные для обработки текста, условно можно разделить на следующие группы задач, которые они выполняют:

- конкатенация текста (команды **cat**, **join**);
- форматирование текста (команды **sort**, **split**, **tr**, **uniq**);
- просмотр текста (команды **head**, **tail**);
- работа с фрагментами текста (команды **cut**, **wc**).

6.1.1. Конкатенация текста: утилиты cat и join

Утилиты данной группы предназначены для объединения нескольких файлов в один. Существуют три важные команды в этой категории: **cat**, **join** и **paste**, – которые конкатенируют файлы на основе указанных полей или же соединяют несколько файлов построчно в один файл.

Команда **cat** (concatenate) используется для последовательного объединения произвольного количества файлов и вывода результирующего текста в стандартный канал вывода. В следующем листинге приведен пример объединения двух файлов в один:

```
# cat first.file second.file > third.file
```

Несмотря на свое предназначение, утилита **cat** используется в основном для вывода содержимого отдельного файла:

```
# cat first.file
```

Это очень удобно для отображения коротких файлов. Если файл длинный, то лучше воспользоваться утилитой-пейджером **less**.

Команда **cat** имеет следующий синтаксис:

```
cat [опции...] [файлы...]
```

Если файлы не заданы или в качестве одного из файлов указан символ «**-**», то команда считывает данные из стандартного канала ввода. Ввод завершается указанием конца файла (**EOF**).

Наиболее часто используемыми при работе с командой **cat** являются следующие опции:

- **e** – отображает конец строки символом **$**;
- **n** – отображает номер каждой строки;
- **T** – отображает символ табуляции как **^I**;
- **s** – сокращает повторяющиеся пустые строки при выводе.

Команда **join** используется для объединения двух файлов на основе указанных ***полей*** (аналог оператора JOIN языка SQL). ***Поля*** представляют собой части строки, разделенные произвольным повторяющимся разделителем (по умолчанию используется символ пробела). Для указания другого символа используется ключ **-t**. Если имеются два файла следующего содержания:

File1:

```
555-2397 Ivanov, Sergey
555-5116 Sidorov, Pavel
555-7929 Petrov, Ivan
555-9871 Ivanov, Ilja
```

File2:

```
555-2397 unlisted
555-5116 listed
555-7929 listed
555-9871 unlisted
```

то для их объединения по первому полю используется следующая команда:

```
# join File1 File2
555-2397 Ivanov, Sergey unlisted
555-5116 Sidorov, Pavel listed
555-7929 Petrov, Ivan listed
555-9871 Ivanov, Ilja unlisted
```

По умолчанию команда **join** использует первое поле для объединения. Можно указать, используя ключи **-1** <N> (для 1-го файла) и **-2** <M> (для 2-го файла), какое поле в каждом файле нужно использовать для объединения. Одним из ограничений команды **join** является то, что объединять файлы можно, только если строки каждого файла отсортированы согласно единому порядку сортировки. Поэтому предварительно рекомендуется отсортировать файлы при помощи команды **sort**.

Команда **paste** используется для объединения соответствующих строк одного или нескольких файлов в вертикальные столбцы, разделенные знаком табуляции. Синтаксис команды следующий:

```
paste [опции...] файлы...
```

Наиболее часто используемыми при работе с командой **paste** являются следующие опции:

- **«-»** – использовать вместо файла стандартный канал вывода;
- **d** *символ* – использовать в качестве разделителя указанный символ;
- **s** – объединять идущие подряд строки.

В следующем листинге приведен пример объединения строк файлов **File1** и **File2**:

```
# paste File1 File2
555-2397 Ivanov, Sergey 555-2397 unlisted
555-5116 Sidorov, Pavel 555-5116 listed
555-7929 Petrov, Ivan 555-7929 listed
555-9871 Ivanov, Ilja 555-9871 unlisted
```

Основным использованием команды **paste** является разбивка текста на две колонки:

```
# who
root     tty1         2009-09-03 20:19
root     tty2         2009-09-03 20:22
root     tty4         2009-09-03 20:18
root     pts/1        2009-09-03 20:24 (:0.0)
root     pts/2        2009-09-03 21:51 (:0.0)
# who | paste - -
root     tty1         2009-09-03 20:19  root   tty2    2009-09-03 20:22
root     tty4         2009-09-03 20:18  root   pts/1   2009-09-03 20:24 (:0.0)
root     pts/2        2009-09-03 21:51 (:0.0)
```

6.1.2. Форматирование текста: утилиты sort, split, uniq и tr

Утилита **sort** сортирует содержимое файла и часто используется как промежуточный фильтр в конвейерах. Эта команда сортирует поток текста в порядке убывания или возрастания, в зависимости от заданных опций. Команда может считывать данные из стандартного канала ввода. В следующем листинге приведен пример сортировки нескольких простых слов, введенных с клавиатуры:

```
# sort
bananas
carrots
apples
<Ctrl+D>
apples
bananas
carrots
```

Как видно из листинга, слова были отсортированы в алфавитном порядке. Команда имеет следующий синтаксис:

```
sort [опции...] [файлы...]
```

Наиболее часто используемыми при работе с командой **paste** являются следующие опции:

- **i** – игнорировать регистр символов;
- **n** – использовать числовую сортировку;
- **k** *поле* – использовать в качестве поля сортировки указанный номер поля.

В следующем листинге приведен пример сортировки файла **File1** по 3-му полю:

```
# sort -k 3 File1
555-9871 Ivanov, Ilja
555-7929 Petrov, Ivan
555-5116 Sidorov, Pavel
555-2397 Ivanov, Sergey
```

Утилита **sort** имеет большое количество дополнительных опций, описание которых можно посмотреть в руководстве **man**.

Команда **split** используется для разбивки файла на несколько более мелких файлов одинакового размера и имеет следующий синтаксис:

```
split [опции...] [входной_файл] [выходной_файл]
```

Исходный файл остается без изменений. Выходные данные записываются в выходной_файл**aa**, выходной_файл**ab** и т. д.

Наиболее часто используемыми при работе с командой **paste** являются следующие опции:

- **b <N>** – разделить входной файл на файлы размером N байт;
- **n <N>** – разделить входной файл на файлы длиной N строк;
- **d** – использовать числовые суффиксы вместо алфавитных.

В следующем листинге приведен пример разбивки файла на фрагменты размером 100 Кб:

```
# split -b 5120 messages
# ls |grep ^x
xaa
xab
xac
xad
xae
xaf
```

Команда **uniq** используется для удаления одинаковых строк и имеет следующий синтаксис:

```
uniq [опции...] [файл1 [файл2]]
```

Uniq удаляет повторяющиеся соседние строки из отсортированного файла файл1, направляя каждую строку в файл2.

Наиболее часто используемыми при работе с командой **uniq** являются следующие опции:

- **d** – вывести повторяющиеся строки;
- **u** – вывести только уникальные строки;
- **s <N>** – пропустить первые N символов;
- **w <N>** – сравнивать только по первым N символам;
- **f <N>** – пропустить первые N полей;
- **c** – вывести количество вхождений каждой строки.

В следующем листинге приведен пример удаления повторяющихся строк из отсортированного файла:

```
# cat /tmp/file3 | uniq
resolv.conf
hosts
ifconfig
who
sed
hosts
resolv.conf
# cat /tmp/file3 | sort | uniq
hosts
ifconfig
resolv.conf
sed
who
```

Команда **tr** используется для изменения символов, присутствующих в стандартном канале ввода. Команда имеет следующий синтаксис:

```
tr [опции...] [строка1 [строка2]]
```

Команда **tr** копирует входные данные из стандартного канала ввода в стандартный канал вывода, выполняя замену символов `строка1` на `строка2` или удаление символов `строка1`. Удаляет повторяющиеся соседние строки из отсортированного файла `файл1`, направляя каждую строку в `файл2`.

Наиболее часто используемыми при работе с командой **tr** являются следующие опции:

- **d** – удалить символы `строка1` из выходных данных;
- **s** – удалить повторяющиеся выходные символы в `строке2`;
- **t** – сократить `строку1` до длины `строки2` перед обработкой входных данных.

В следующем листинге приведен пример преобразования нижнего регистра на верхний во входном файле:

```
# cat /etc/resolv.conf
; generated by /sbin/dhclient-script
search linux.lab.
nameserver 172.16.0.5
# tr 'a-z' 'A-Z' < /etc/resolv.conf
; GENERATED BY /SBIN/DHCLIENT-SCRIPT
SEARCH LINUX.LAB.
NAMESERVER 172.16.0.5
```

6.1.3. Просмотр текста: утилиты head и tail

Команды **head** и **tail** используются для отображения строго определенного количества строк начала и конца файла соответственно. По умолчанию команды **head** и **tail** выводят 10 строк с начала и конца файла соответственно. Команда **head** имеет следующий синтаксис:

```
head [опции...] [файлы]
```

Наиболее часто используемыми при работе с командой **head** являются следующие опции:

- **<N>** – вывести первых N строк файла;
- **c <N[b|k|m]>** – вывести первые N байт (килобайт, мегабайт) файла.

Команда **tail** имеет следующий синтаксис:

```
tail [опции...] [файлы]
```

Наиболее часто используемыми при работе с командой **head** являются следующие опции:

- **<N>** – вывести последние N строк файла;
- **f** – вывести новые данные, записываемые в файл в стандартный канал вывода;
- **r** – вывести все строки файла в обратном порядке.

В следующем листинге приведен пример вывода первых 4 строк файла **/etc/services**:

```
# head -4 /etc/services
# /etc/services:
# $Id: services,v 1.42 2006/02/23 13:09:23 pknirsch Exp $
#
# Network services, Internet style
```

Утилиты **head** и **tail** очень удобны для быстрого просмотра частей длинного текстового файла или же для просмотра изменений файла.

6.1.4. Работа с элементами текста: утилиты cut и wc

Команда **cut** предназначена для извлечения отдельных полей текста из файлов. Команда имеет следующий синтаксис:

```
cut опции [файлы]
```

Наиболее часто используемыми при работе с командой **cut** являются следующие опции:

- **b <список>** – выбрать из файла указанные в списке байты;
- **c <список>** – выбрать из файла указанные в списке символы;
- **f <список>** – выбрать из файла указанные в списке поля;
- **d <символ>** – использовать указанный символ в качестве разделителя;
- **s** – выбрать только те строки, которые содержат указанный разделитель (используется совместно с ключом **-d**).

Почти во всех опциях команды cut указывается **список** тех или иных элементов текста. Список может представлять собой одно число, диапазон чисел (например, 2–4), интервал чисел (например, –4 или 4–). В последнем случае началом интервала считается начало и конец строки.

В основном команда cut используется в сценариях для обработки выходных данных какой-либо команды. В следующем листинге приведен пример получения аппаратного (MAC) адреса:

```
# ifconfig eth0
          eth0   Link encap:Ethernet  HWaddr 00:0C:29:07:FC:29
          inet   addr:192.168.137.4          Bcast:192.168.137.255
          Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:59 Base address:0x2024
# ifconfig eth0 |grep HWaddr|cut -d " " -f 11
00:0C:29:07:FC:29
```

Команда **wc** используется для вывода количества строк, символов, строк и байт. Данная команда в основном используется для получения информации о количестве строк файла или о количестве файлов, в случае использования в сценариях.

Пример запуска данной команды представлен в следующем листинге:

```
# wc /etc/redhat-release
 1  8 54 /etc/redhat-release
```

В листинге первое число соответствует количеству строк файла, второе – количеству слов, и третье – количеству занимаемых байт. Можно ограничить вывод информации только по строкам, словам, байтам и символам при помощи ключей **–l**, **–w**, **–c**, и **–m** соответственно.

В следующем листинге приведен пример подсчета количества файлов в каталоге **/var/log**:

```
# ls -l /var/log|wc -l
76
```

В каталоге **/var/log** в данном случае содержатся 75 файлов, так как первая строка вывода команды **ls** содержит общее количество блоков, занимаемых файлами и каталогами данного каталога.

6.2. Сравнение файлов и каталогов

Сравнение файлов в ОС Linux имеет большое значение, особенно это важно для разработчиков и системных администраторов, которые работают с разными версиями файлов, выполняя операции сравнения файлов с исходным кодом, конфигурационных, журнальных и других текстовых данных. В ОС Linux присутствуют несколько утилит, предназначенных для этого:

- **cmp** (побайтовое сравнение двух файлов);
- **comm** (построчное сравнение отсортированных файлов);
- **diff** (построчное сравнение файлов).

Команда cmp посимвольно сравнивает два файла и отображает изменения на экране. Команда имеет следующий синтаксис:

```
cmp [опции] файл1 файл2 [сдвиг1[сдвиг2]]
```

Необязательные параметры сдвиг1 и сдвиг2 задают смещение внутри файла, определяющее точку начала сравнения.

Без указания опций команда **cmp** выводит первое различие в файлах в виде «символ, строка». Опция **–l** выводит смещения и ASCII-коды различающихся байт. Команда **cmp** в основном подходит для простого определения того, что файлы разные. Если файлы одинаковые, то команда ничего не выведет на экран и код ее завершения будет равен нулю. Если файлы различны, то код завершения команды будет равен 1. Опция **–s** позволяет оценить файлы только по коду завершения, не выводя самих различий. В следующем листинге приведен пример сравнения двух тестовых файлов:

```
# cat hosts.old
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1       localhost.localdomain localhost
::1             localhost6.localdomain6 localhost6
# cat hosts.new
# Do not remove the following line, or various programs
# that require network functionality will fail
127.0.0.1       localhost.localdomain localhost
::1             localhost6.localdomain6
# cmp hosts.old hosts.new
hosts.old hosts.new differ: byte 103, line 2
```

Как видно, первое несовпадение было найдено в 103-м символе, расположенном на 2-й строке.

Для сравнения отсортированных файлов используется команда **comm**, имеющая следующий синтаксис:

```
comm [опции] файл1 файл2
```

Вывод данных этой команды выполняется в три колонки: строки, входящие только в 1-й файл, строки, входящие только во 2-й файл, строки, входящие в оба файла. Данная команда выбирает повторяющиеся и неповторяющиеся строки в двух отсортированных файлах. Ее работа схожа с командой **uniq**, которая выбирает повторяющиеся (или неповторяющиеся) строки в одном отсортированном файле.

Наиболее часто используемыми при работе с командой **comm** являются следующие опции:

- **1** – не выводить колонку 1;
- **2** – не выводить колонку 2;
- **3** – не выводить колонку 3;
- **12** – вывести только строки колонки 3;
- **13** – вывести только строки колонки 2;
- **23** – вывести только строки колонки 1.

Более удобный способ сравнения двух файлов предоставляет команда **diff**, которая выводит различия между строками первого и второго файлов. Команда имеет следующий синтаксис:

```
diff [опции] [опции для каталогов] файл1 файл2
```

Выходные данные команды состоят из строк контекста каждого файла. Текст из `файла1` обозначается символом <, текст из `файла2` – символом >. Строки контекста начинаются с команды редактора **ed** (**a**, **c** или **d**), преобразующей `файл1` в `файл2`. Символ «**a**» означает, что «добавлено», символ «**d**» – удалено, символ «**c**» – изменено. Перед символами **a**, **d** или **c** стоят номера строк `файла1`, после них – номера строк `файла2`. Каждая строка, которая была добавлена, удалена или изменена, предваряется символами **<** или **>**.

В следующем листинге приведен пример сравнения двух файлов из предыдущего примера:

```
# diff hosts.old hosts.new
2c2
< # that require network functionality will fail.
---
> # that require network functionality will fail
4c4
< ::1             localhost6.localdomain6 localhost6
---
> ::1             localhost6.localdomain6
```

Фактически вывод команды **diff** определяет, что нужно сделать с исходным файлом, чтобы получился итоговый файл. Первая команда **2c2** означает, что 2-ю строку первого файла (начинается с символа <) надо заменить на 2-ю строку из 2-го файла (начинается с символа >); аналогичная команда представлена для 4-й строки.

Более наглядную форму сравнения можно получить, используя опцию –u:

```
# diff -u hosts.old hosts.new
--- hosts.old   2009-09-06 18:34:18.000000000 +0400
+++ hosts.new   2009-09-06 18:34:34.000000000 +0400
@@ -1,4 +1,4 @@
 # Do not remove the following line, or various programs
-# that require network functionality will fail.
+# that require network functionality will fail
 127.0.0.1       localhost.localdomain localhost
-::1             localhost6.localdomain6 localhost6
+::1             localhost6.localdomain6
```

Первый файл в данном случае идентифицируется символом «**---**», а второй – символом «**+++**». Далее следует краткое описание изменяемых строк в виде «@@ -1,4 +1,4 @@». Далее символами «-» и «+» указывается принадлежность строк первого и второго файлов соответственно.

Очень часто команду **diff** используют для получения патчей к файлам исходных кодов или любых других файлов. Допустим, имеется следующий фрагмент кода:

```
/* hello.c */
#include <stdio.h>

    int main() {
         printf('Hello, World!');
         exit(0);
}
```

Предположим, данный фрагмент кода изменился и стал иметь следующий вид:

```
/* hello.c */
/* (c) 2009 Andrew Ivanov */

#include <stdio.h>

    int main() {
         printf('Hello, World!\n');
         return 0;
}
```

Тогда для того, чтобы создать патч-файл (то есть набор инструкций, который необходимо применить к исходному файлу, чтобы он стал итоговым файлом), необходимо выполнить команду **diff** с ключом **–c**:

```
# diff -c hello.c.old hello.c.new |tee hello.c.patch
*** hello.c.old 2009-09-06 19:49:52.000000000 +0400
– hello.c.new 2009-09-06 19:50:55.000000000 +0400
***************
*** 1,7 ****
  /* hello.c */
  #include <stdio.h>

    int main() {
!       printf('Hello, World!');
!       exit(0);
  }
--- 1,9 ----
  /* hello.c */
+ /* (c) 2009 Andrew Ivanov */
+
  #include <stdio.h>

    int main() {
!       printf('Hello, World!\n');
!       return 0;
  }
```

Для применения изменений, содержащихся в патч-файле **hello.c.patch**, используется утилита **patch**:

```
# patch -b –verbose hello.c.old helo.c.patch
Hmm...  Looks like a new-style context diff to me...
The text leading up to this was:
--------------------------
|*** hello.c.old        2009-09-06 19:49:52.000000000 +0400
|--- hello.c.new        2009-09-06 19:50:55.000000000 +0400
--------------------------
Patching file hello.c.old using Plan A...
Hunk #1 succeeded at 1.
done
```

Здесь ключ **–b** указан для создания резервной копии файла перед применением изменений, ключ **--verbose** указан для отображения наглядности работы утилиты **patch**.

6.3. Модификация файлов. Использование редакторов sed и awk

Иногда для изменения текста в файлах, анализа и обработки фрагментов текста использование стандартных утилит ОС Linux оказывается затруднительным. В процессе написания сценариев командной оболочки приходится очень часто находить и обрабатывать отдельные части текста, причем логика обработки текста может быть достаточно сложна. Для этих случаев в ОС Linux присутствуют более совершенные средства обработки текста – язык программирования **awk** и потоковый редактор **sed**.

6.3.1. Язык программирования awk

Язык программирования **awk** представляет собой программу сопоставления с образцами, предназначенную для обработки файлов, особенно в случаях, когда каждая строка имеет простую структуру в виде полей.

Использование **awk** позволяет:

- представлять текстовый файл как состоящий из записей и полей в текстовой базе данных;
- выполнять арифметические и текстовые операции;
- использовать программные конструкции, такие как циклы и условные операторы;
- создавать отформатированные отчеты;
- определять собственные функции;
- выполнять команды ОС из сценариев;
- использовать регулярные выражения для разделения записей и полей;
- работать с массивами.

Существуют два варианта синтаксиса вызова **awk**:

```
awk [опции] 'сценарий' переменная=значение файлы
awk [опции] -f файл-сценарий переменная=значение файлы
```

Можно задать `сценарий` непосредственно из командной строки либо предварительно сохранить его в `файл-сценарий`. Переменным, присутствующим в сценариях **awk**, можно задавать значения непосредственно из командной строки. Обязательным аргументом команды **awk** является указание одного или нескольких файлов, которые должен обработать указанный сценарий. Можно запускать сценарии **awk** и без указания файлов, тогда данные для обработки следует направлять из стандартного канала ввода.

Сценарий **awk** состоит из образцов и действий, записанных в формате:

```
образец {действие}
```

Любой из данных элементов можно опустить. Если отсутствует образец, то ко всем строкам применяется указанное действие. Если отсутствует действие, то выводятся все строки, удовлетворяющие указанному образцу.

Для того чтобы понять принцип работы awk, рассмотрим следующий листинг:

```
# awk '{ print }' /etc/passwd
root:x:0:0:test,w,1,w:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

В результате выполнения данной команды выводится содержимое файла **/etc/passwd**. Вызывая **awk**, мы указали **/etc/passwd** в качестве входного файла. После запуска **awk** обработал команду **print** для каждой строки в файле **/etc/passwd** по порядку (сверху вниз). Весь вывод отправлен в стандартный канал вывода, и мы получили результат, идентичный результату команды **cat /etc/passwd**. Теперь объясним блок **{ print }**. В фигурные скобки заключено выполняемое действие. В данном случае есть лишь одно действие – команда **print**. В **awk** команда **print** без дополнительных параметров печатает все содержимое текущей строки.

Вот еще один пример программы на **awk**, которая делает то же самое:

```
# awk '{ print $0 }' /etc/passwd
```

В **awk** переменная **$0** представляет всю текущую строку, поэтому **print** и **print $0** делают в точности одно и то же. Если угодно, можно создать программу на **awk**, которая будет выводить данные, совершенно не связанные с входными данными. Вот пример:

```
# awk '{ print "" }' /etc/passwd
```

При передаче строки "" команде **print** она всегда печатает пустую строку. Если протестировать этот скрипт, обнаружится, что **awk** выводит одну пустую строку на каждую строку в файле **/etc/passwd**. Это происходит потому, что **awk** исполняет скрипт **для каждой строки во входном файле**.

Awk хорошо подходит для обработки текста, разбитого на множество логических полей, и дает возможность без усилий обращаться к каждому отдельному полю из **awk**-сенария. Следующий скрипт распечатает список всех учетных записей в системе:

```
# awk -F":" '{ print $1 }' /etc/passwd
```

В вызове **awk** в вышеприведенном примере ключ **–F** задает символ «**:**» в качестве разделителя полей. Обрабатывая команду **print $1**, **awk** выводит первое поле, встреченное в каждой строке входного файла. Вот еще один пример:

```
# awk -F":" '{ print $1 $3 }' /etc/passwd
```

Вот фрагмент из вывода на экран данной команды:

```
root0
bin1
daemon2
...
```

Как видно, **awk** выводит первое и третье поля файла **/etc/passwd**, которые представляют собой соответственно поля имени пользователя и **uid**. При этом, хотя сценарий и работает, он не совершенен, так как нет пробелов между двумя

выходными полями. Следующая команда вставляет пробел между соответствующими полями:

```
# awk -F":" '{ print $1 " " $3 }' /etc/passwd
```

Когда команда **print** вызывается таким способом, она последовательно соединяет **$1**, " " и **$3**, создавая удобочитаемый вывод на экране.

Передача сценариев интерпретатору **awk** в виде аргументов командной строки может быть удобной для небольших однострочных текстов, но когда дело доходит до сложных многострочных программ, определенно будет лучше составить скрипт в виде внешнего файла и указать его, используя ключ **–f**.

Размещение скриптов в отдельных текстовых файлах также позволяет воспользоваться дополнительными преимуществами **awk**. Например, следующий многострочный скрипт делает то же самое, что и один из предыдущих однострочных, – распечатывает первое поле каждой строки из файла **/etc/passwd**:

```
BEGIN {
        FS=":"
}
{ print $1 }
```

Разница между этими двумя методами состоит в том, как задается разделитель полей. В этом сценарии разделитель полей указывается внутри самой программы (установкой переменной **FS**), тогда как в предыдущем примере **FS** настраивается путем передачи **awk** параметра **-F":"** в командной строке. Обычно лучше всего задавать разделитель полей внутри самого скрипта просто потому, что тогда не потребуется запоминать еще один аргумент командной строки.

Обычно **awk** выполняет каждый блок **{}** в тексте сценария один раз для каждой входной строки. Однако в программировании часто встречаются ситуации, когда требуется выполнить код инициализации перед тем, как **awk** начнет обрабатывать текст из входного файла. Для таких случаев **awk** дает возможность определять блок **BEGIN** (см. предыдущий пример). Поскольку блок **BEGIN** обрабатывается до того, как **awk** начинает обрабатывать входной файл, это отличное место для инициализации переменной **FS** (разделитель полей), вывода заголовка или инициализации других глобальных переменных, которые будут позже использоваться в сценарии.

Awk также предоставляет еще один специальный блок, называемый блоком **END**. **Awk** выполняет этот блок после того, как все строки во входном файле были обработаны. Обычно блок **END** используется для выполнения заключительных вычислений или вывода итогов, которые должны появиться в конце выходного потока.

Awk позволяет использовать регулярные выражения для избирательного выполнения отдельных блоков программы в зависимости от того, совпадает или нет регулярное выражение с текущей строкой. Вот пример сценария, выводящего только те строки, которые содержат символьную последовательность **foo**:

```
/foo/ { print }
```

Есть много других способов избирательно выполнять блок сценария. Можно поместить перед блоком программы любое логическое выражение для управле-

ния исполнением этого блока. **Awk** будет выполнять блок программы, только если предыдущее логическое выражение истинно. Следующий пример сценария будет выводить третье поле всех строк, в которых первое поле равно **tip**. Если первое поле текущей строки не равно **tip**, **awk** продолжит обработку файла и не выполнит оператор **print** для текущей строки:

```
$1 == "tip" { print $3 }
```

Awk предлагает полный набор операторов сравнения, в том числе обычные «==», «<», «>», «<=», «>=» и «!=». Кроме того, **awk** предоставляет операторы «~» и «!~», которые означают «совпадает» и «не совпадает». При их использовании переменная помещается слева от оператора, а регулярное выражение – справа от него. Вот пример, где выводится только третье поле строки, если пятое поле той же строки содержит символьную последовательность daemon:

```
$5 ~ /daemon/ { print $3 }
```

Awk предоставляет также стандартные логические операторы **if**. При желании можно переписать предыдущий пример с использованием оператора **if**:

```
{
  if ( $5 ~ /daemon/ ) {
          print $3
  }
}
```

Оба сценария работают идентично. В первом примере логическое выражение находится вне блока **{}**, в то время как во втором примере блок выполняется для каждой входной строки, и мы избирательно выполняем команду печати, используя оператор **if**. Оба метода работают, и выбрать можно тот, который наилучшим образом объединяется с другими частями скрипта.

Вот более сложный пример оператора **if** в **awk**. Как можно видеть, даже в случае сложных вложенных условных выражений операторы if выглядят достаточно стандартно и логично:

```
{
  if ( $1 == "foo" ) {
           if ( $2 == "foo" ) {
                    print "one"
           } else {
                    print "two"
           }
  } else if ($1 == "bar" ) {
           print "two"
  } else {
           print "three"
  }
}
```

Язык **awk** имеет достаточно большое количество встроенных переменных. Рассмотрим основные из них.

- Разделитель полей (**FS**). Как упоминалось ранее, эта переменная позволяет задать последовательность символов, которую **awk** будет считать разде-

лителем полей. Когда мы использовали в качестве ввода **/etc/passwd**, переменная **FS** была установлена в «**:**». Значение переменной **FS** не обязано быть одним символом; ей может быть присвоено регулярное выражение, задающее символьный шаблон любой длины. Если производится обработка полей, разделенных одним или несколькими символами табуляции, то **FS** нужно настроить таким образом:

```
FS="\t+"
```

- Номер записи (**NR**). Данная переменная всегда содержит номер текущей записи (**awk** считает первую запись записью номером 1). Переменную **NR** можно использовать для вывода только определенных строк ввода:

  ```
  (NR < 10 ) || (NR > 50)
  ```

  В данном примере будут рассматриваться только записи до 10-й или после 50-й.
- Число полей (**NF**). Переменная **NF** определяет число полей в каждой записи. **Awk** автоматически устанавливает значение этой переменной равным числу полей в текущей записи. Можно использовать переменную **N**F для отображения только определенных входных строк:

  ```
  NF == 5 { print "в этой записи пять полей: " $0 }
  ```

В настоящее время существует много разновидностей интерпретаторов языка **awk**, среди которых наиболее поздним и расширенным вариантом является **gawk**, имеющий встроенные возможности работы с сетью и множество дополнительных возможностей. Исчерпывающая информация по утилите **gawk** содержится в руководстве **info**.

6.3.2. Потоковый редактор sed

Утилита **sed** является одной из наиболее известных инструментов обработки текста и чаще всего используется для выполнения простых подстановок в потоках данных, проходящих через стандартные каналы, однако на этом возможности **sed** далеко не ограничиваются. Утилита **sed** является поточным редактором. Она, подобно **awk**, интерпретирует сценарий и выполняет действия, описанные в сценарии. Входные данные для **sed**, как правило, берутся из файла или стандартного канала, но также могут вводиться с клавиатуры.

К типовым задачам, к которым можно применить редактор **sed**, относятся:

- автоматическое редактирование файлов;
- упрощенное выполнение повторных операций для нескольких файлов;
- создание сценариев преобразования.

Редактор **sed** работает следующим образом:

- каждая строка входных данных копируется в «область образца» – внутренний буфер, в котором выполняются операции редактирования;
- все команды редактирования в сценарии **sed** применяются, в свою очередь, к каждой строке входных данных;

- команды редактирования применяются ко всем строкам (глобально), если только адресация строк жестко не указана;
- если команда изменяет входные данные, последующие команды и проверки адресов применяются к текущей строке в области образца, а не к оригинальной входной строке;
- оригинальный входной файл не изменяется (если не указан ключ **–i**), так как команды редактирования модифицируют только копию входного файла в памяти. Копия направляется в стандартный канал вывода;
- редактор **sed** также использует «область хранения» – отдельный буфер, который может использоваться для последующего извлечения данных.

Существуют два варианта синтаксиса вызова **sed**:

```
sed 'команды' файлы
sed -f файл-сценарий файлы
```

Первый вариант синтаксиса позволяет указать команду редактирования в командной строке, заключив ее в одинарные кавычки. Второй вариант синтаксиса позволяет создавать сценарии, содержащие команды **sed**. Если файлы не заданы, то выполняется чтение стандартного канала ввода.

Команды редактора **sed** имеют следующий общий синтаксис:

```
[адрес[,адрес]][!]команда[аргументы]
```

Параметр `команда` принимает односимвольные значения. `Аргументы` включают в себя символы, используемые в соответствующей команде. Редактор **sed** может содержать один или два адреса для поиска образцов. Адреса допускается указывать в следующем формате:

- **/образец/** – выбрать строки, соответствующие указанному образцу;
- **\;образец;** – подобно предыдущему, однако в качестве разделителя используется символ «;»;
- **N** – выбрать строку с номером N;
- **$** – выбрать последнюю строку вывода.

Редактор **sed** выполняет обработку текста согласно следующим правилам адресации:

- если в команде нет адреса, то команда применяется к каждой входной строке;
- если в команде один адрес, то команда применяется к любой строке, соответствующей указанному адресу;
- если в команде указаны два адреса, разделенные запятыми, то она применяется к первой соответствующей строке и ко всем строкам, до строки, соответствующей второму адресу;
- если в команде указан адрес с символом «!», то команда применяется ко всем строкам, не соответствующим указанному адресу.

Далее приведен список наиболее часто используемых команд редактора **sed**:

- a \текст – добавить текст, следующий за каждой строкой, соответствующей адресу;
- c \текст – заменить строки, указанные адресом, заданным текстом;
- d – удалить заданную адресом строку из области образца;

- g – вставить содержимое области хранения в область образца;
- h – вставить содержимое области образца в область хранения;
- i \текст – вставить текст перед каждой строкой, соответствующей адресу;
- p – вывести строки, заданные адресом;
- q – выполнить выход при обнаружении заданного адреса;
- s/регулярное_выражение/замена/флаги – выполнить подстановку текста, определенного в замене, в каждую строку, заданную адресом и удовлетворяющую указанному регулярному выражению;
- w файл – добавить содержимое области образца в файл (максимально можно использовать до 10 открытых файлов);
- y /строка1/строка2/ – заменить все вхождения символов из строки1 на соответствующие из строки2. Длины строк должны быть равными;
- = – вывести номер строки;
- # – комментарий.

Рассмотрим несколько примеров использования редактора **sed**. В следующем примере выполнено добавление текста после строки, указанной адресом:

```
# who
root     tty1         Mar 14 17:00
nik      tty2         Mar 11 18:21
root     tty6         Mar 11 17:11
jake     tty5         Mar 12 17:26
# who | sed '2a\
new_line
'
root     tty1         Mar 14 17:00
nik      tty2         Mar 11 18:21
new_line
root     tty6         Mar 11 17:11
jake     tty5         Mar 12 17:26
```

В данном примере при помощи команды «**a**» после 2-й строки был добавлен текст «**new_line**».

В следующем листинге приведен пример замены строк, соответствующих адресу:

```
# who
root     tty1         Mar 14 17:00
nik      tty2         Mar 11 18:21
root     tty6         Mar 11 17:11
jake     tty5         Mar 12 17:26
# who | sed '/root/ c\
new_line
'
new_line
nik      tty2         Mar 11 18:21
new_line
jake     tty5         Mar 12 17:26
```

Как видно из примера, была выполнена замена всех строк, содержащих текст **root**, на строку замены – «**new_line**».

В следующем примере выполняется удаление нескольких строк:

```
# who
root     tty1         Mar 14 17:00
nik      tty2         Mar 11 18:21
root     tty6         Mar 11 17:11
jake     tty5         Mar 12 17:26
# who | sed '2,4d'
root     tty1         Mar 14 17:00
```

В следующем листинге приведен пример выполнения замены указанного адреса:

```
# who
root     tty1         Mar 14 17:00
nik      tty2         Mar 11 18:21
root     tty6         Mar 11 17:11
jake     tty5         Mar 12 17:26
# who | sed  's/t/T/g'
rooT     TTy1         Mar 14 17:00
nik      TTy2         Mar 11 18:21
rooT     TTy6         Mar 11 17:11
jake     TTy5         Mar 12 17:26
```

Из примера видно, что прописная буква **t** была глобально заменена на заглавную.

Модуль 7. Регулярные выражения

Изучив данный модуль, вы научитесь:

- понимать принцип работы регулярных выражений;
- использовать утилиты семейства grep для поиска данных.

Во многих случаях при работе с ОС Linux часто приходится сталкиваться с необходимостью произвести выборку данных, осуществить отбор строк для редактирования, произвести поиск (замену) или исключить ненужное из списка данных. Для этого в ОС Linux используются ***регулярные выражения***.

В самом простом определении регулярное выражение – это просто набор букв и цифр, позволяющих описать некий шаблон, по которому требуется осуществлять выборку данных, или, другими словами, строка, по которой будет происходить поиск.

С другой стороны, регулярное выражение (**regular expression**) – это способ описания наборов символов. Простейшим набором является слово, но регулярное выражение может включать и глобальные символы, заменяющие другие символы и позволяющие более кратко записать всевозможные требования к поиску. Все утилиты ОС Linux, высокоуровневые и низкоуровневые языки программирования работают с регулярными выражениями. Рассмотренные ранее утилиты, такие как **awk** и **sed**, позволяют искать образцы на основе регулярных выражений. Однако следует понимать, что разные утилиты не обязательно могут использовать один и тот же стандарт регулярных выражений.

В третьем модуле мы уже сталкивались с так называемыми подстановками файлов, что, по сути, является регулярным выражением.

В данном модуле мы рассмотрим основные регулярные выражения, используемые утилитами семейства **grep**, включая регулярные выражения в стиле языка **Perl (grep –P)**.

Обычно регулярные выражения входят в состав утилиты **grep** следующим образом:

```
grep [опции] [регулярное_выражение] [файлы]
```

Все регулярные выражения состоят из двух типов символов: стандартных текстовых символов, называемых ***литералами***, и специальных символов, называемых ***метасимволами***. Кроме того, в состав регулярного выражения могут входить **escape-последовательности**, позволяющие использовать метасимволы в качестве литералов или обозначать специальные условия (например, границы слов).

Как правило, регулярные выражения заключаются в символы кавычек. От того, какие будут указаны кавычки, зависят смысл выражения и способ его интер-

претации. Обычно для регулярных выражений используют прямые одинарные кавычки, в первую очередь это сделано для того, чтобы интерпретатор не интерпретировал разделенные пробелом символы как отдельные аргументы. В случае использования одинарных кавычек подстановка переменных не выполняется. В случае использования одинарных наклонных кавычек командная оболочка выполняет весь текст, заключенный в эти кавычки, подобно обычной команде. Например, в команде

```
$ grep 'whoami' имя_файла
```

оболочка вместо слова '**whoami**' выполнит подстановку команды **whoami**, которая, в свою очередь, отобразит имя пользователя текущего сеанса. Таким образом, командой **grep** будет выполнен поиск строк в указанном файле, в состав которых входит вывод команды **whoami**.

Использование прямых двойных кавычек целесообразно в случае подстановок переменных, например в команде

```
$ grep "$HOME" имя_файла
```

выполняется поиск всех строк в указанном файле, содержащих значение переменной **$HOME**.

При составлении регулярных выражений используются метасимволы, представленные в табл. 7.1.

Таблица 7.1. Метасимволы и операторы регулярных выражений

Символ	Назначение	Пример	Описание примера
Поиск единичного символа			
.	Поиск одного любого символа	r.d	Соответствует словам red, rid, r2d и т. п.
[...]	Поиск символа из указанного диапазона	[a-f]	Соответствует словам a, b, c, d, e, f
[^...]	Поиск любого символа, не включенного в данный диапазон	..[^24680]	Соответствует любому слову из трех символов, последний из которых не является четным
\	Поиск специального символа (экранирование)		\^ Соответствует строке ^sky, sky^ и т. п.
Поиск символов согласно занимаемой позиции в строке			
^	Поиск с начала строки	^bag	Соответствует образцу, начинающемуся со слова bag в начале строки
$	Поиск с конца строки	^bag$	Соответствует единственному слову bag в строке
\<	Поиск с начала слова	\<the	Слова theater, the и т. п.
\>	Поиск с конца слова	\<the\>	Слово the
Квантификаторы (подсчет количества)			
?	Сопоставление с одним экземпляром предшествующего регулярного выражения или с его отсутствием (используется в egrep и awk)	80[2-4]?86	8086, 80286, 80386 или 80486

Таблица 7.1. Метасимволы и операторы регулярных выражений (окончание)

Символ	Назначение	Пример	Описание примера
*	Поиск любого числа символов (включая их отсутствие)	[A-Z].*	Буква в верхнем регистре, за которой следует несколько символов или ни одного
+	Сопоставление с одним или несколькими экземплярами предшествующего регулярного выражения	[[:upper:]]+	Одна или несколько букв в верхнем регистре
{N}	Сопоставление только при наличии N вхождений	150{3}\b	Соответствует 15000
{N,}	Определяет, сколько раз предшествующее выражение может встречаться. Общее количество повторений может быть от m и более вхождений	150{3,}\b	Соответствует 1500000, 150000000 и т. п.
{min, max}	Определяет, сколько раз предшествующее выражение может встречаться. Общее количество повторений может быть от min до max включительно	150{2,3}\b	Соответствует 1500 и 15000
{,N}	Определяет, сколько раз предшествующее выражение может встречаться. Общее количество повторений может быть до N вхождений включительно	150{,3}\b	Соответствует 1500, 150
Дополнительные условия поиска			
(nm)	Применить сопоставление к заключенной группе регулярных выражений. Используется в egrep и awk	(150){3}	Соответствует 150150150
|	Сопоставление с регулярным выражением, заданным до или после черты (альтернация). Используется в egrep и awk	yes|no	Одно из слов yes или no
\N	Воспроизводит N-й дополнительный образец, заключенный между **\(** и **\)** в образце на данный момент; N задает номер от 1 до 9, начиная слева. Используется в sed и grep	\(why\).*\1	Строка с двумя вхождениями слова why

Помимо указанных выше регулярных выражений, существуют классы POSIX, описанные в модуле 3. Если, используя оговоренные выше регулярные выражения, добиться необходимого варианта решения задачи не удается, то существует еще одна возможность достижения результата – использование регулярных выражений языка **Perl** (**PCRE**[1]). Справочная информация по регулярным выражениям языка Perl доступна посредством ввода команд **perldoc perlrequick** (вводная информация о PCRE) и **perldoc perlre** (описание синтаксиса выражений PCRE). Типовой вызов интерпретатора **perl** для поиска текста из потока входных данных, удовлетворяющего заданному регулярному выражению, имеет вид:

```
$ echo <строка> | perl -p -w -e /[\\bcr]at/
```

Данное регулярное выражение будет удовлетворять словам **\at**, **bat**, **cat**, или **rat**. Приведенный пример используется непосредственно из командной строки в интерактивном режиме. Информация по ключам запуска интерпретатора perl доступна из встроенной справки посредством ввода команды **perldoc runperl**.

GNU-версии стандартных утилит ОС Linux понимают дополнительные управляющие последовательности, представленные в табл. 7.2, и действующие подобно метасимволам.

Таблица 7.2. Управляющие последовательности

Символ	Назначение	Пример	Описание примера
\b и \B	Граница слова и обратный слеш (альтернатива \\)	'\bheart\b', 'c:\Bwindows'	Соответствует только слову heart Соответствует строке c:\windows
\w и \W	Поиск слов и неслов	[a-z0-9] и [^a-z0-9]	Соответствует только словам Соответствует всему, кроме слов

Основным инструментом для поиска текста при помощи регулярных выражений является семейство утилит **grep**, куда входят команды:

- **fgrep**;
- **grep**;
- **egrep**.

Команда **fgrep** используется для поиска строк, соответствующих **непосредственно** заданному текстовому образцу в одном или нескольких файлах. Данная команда не поддерживает регулярные выражения, поэтому является более быстрой по сравнению с другими утилитами семейства **grep**. Команда **fgrep** имеет следующий синтаксис:

---

[1] Perl compatible regular expressions (регулярные выражения, совместимые с Perl). URL: http://en.wikipedia.org/wiki/PCRE.

```
fgrep [опции] [образец] [файлы]
```

Опции команды **fgrep** аналогичны опциям команды **egrep**. В качестве примера в следующем листинге выполнен поиск строк файла, содержащих слова из списка **list**:

```
$ fgrep –f list имя_файла
```

Команда **grep** выполняет поиск строк, соответствующих заданному регулярному выражению в одном или нескольких файлах.

Команда **egrep** выполняет то же самое, что и команда **grep**, однако **egrep** не поддерживает метасимволы **\(**, **\)**, **\n**, **\<**, **/>/**. Утилита **egrep** поддерживает другие метасимволы, а также расширенный набор **+**,**?**,**|** и **()**. Следует учитывать то, что данные символы необходимо заключать в кавычки.

Наиболее часто используемыми при работе с командами **fgrep**, **grep** и **egrep** являются следующие опции:

- **c** – задает отображение только числового значения, указывающего, сколько строк соответствуют шаблону;
- **i** – дает указание игнорировать регистр символов;
- **h** – подавляет вывод имен файлов, включающих найденные строки (по умолчанию в выводе команды **grep** каждой строке предшествует имя файла, в котором она содержится);
- **l** – задает отображение только имен файлов, содержащих найденные строки;
- **n** – задает нумерацию выводимых строк;
- **s** – подавляет вывод сообщений о несуществующих или нетекстовых файлах;
- **v** – задает отображение строк, не соответствующих шаблону;
- **d** <действие> – применить заданное действие для обработки каталогов. Допустимые значения параметра действие: **skip** – пропустить обработку каталога, **recurse** – войти в каталог и выполнить рекурсивную обработку файлов;
- **E** – интерпретировать образец поиска как расширенное регулярное выражение (по умолчанию используется в команде **egrep**);
- **G** – интерпретировать образец поиска как базовое регулярное выражение (по умолчанию используется в команде **grep**);
- **F** – интерпретировать образец поиска как список фиксированных строк, разделенных символами новой строки (по умолчанию используется в команде **fgrep**);
- **P** – интерпретировать образец поиска как **PCRE**;
- **r (-R)** – рекурсивная обработка файлов во всех каталогах и подкаталогах;
- **o** – вывести только часть строки, совпадающей с регулярным выражением;
- **color[=когда]** – вывести совпадающий текст цветом, определенным в переменной окружения **GREP_COLOR**. Значения параметра «когда» следующие: **auto**, **always**, **never**;
- **m <число>** – прекратить чтение файла по достижении заданного количества совпадающих строк;
- **--binary-files=<mun>** – просмотреть первые байты в файле и идентифицировать файл.

При составлении сложных регулярных выражений при помощи команд семейства **grep** следует учитывать приоритетность обработки одних выражений по отношению к другим. Например, квантификация выполняется первее, чем конкатенация, а конкатенация выполняется первее, чем альтернация. Конкатенация строк выполняется простой вставкой текста между строками, которые необходимо конкатенировать. В следующем листинге приведен пример регулярного выражения:

```
$ egrep –r 'pat{2}ern|red' /etc
```

В данном случае операция квантирования (метасимвол **{}**) выполняется в первую очередь, что соответствует двум символам **t**. Затем выполняется конкатенация строк **pat**, **tt** и **ern**, что в итоге дает слово **pattern**. Далее следует применение операции альтернации. Таким образом, итоговому образцу поиска будут удовлетворять строки, содержащие слова **pattern** или **red**. Поиск совпадений будет выполняться рекурсивно в иерархии каталогов **/etc**.

Для того чтобы изменить принятый порядок приоритета, используются метасимволы **()**. В следующем примере показан способ изменения стандартного приоритета интерпретации метасимволов в регулярном выражении:

```
$ egrep –r 'pat{2}(ern|red)' /etc
```

Здесь сначала будут выполняться операции альтернации, таким образом, итоговое регулярное выражение будет выполнять поиск слов **pattern** или **patterd**.

Также следует отметить, что регулярное выражение может занимать более одной строки, пока не будет введен завершающий символ «**‘**» или «**’**».

Для составления комплексных регулярных выражений, где необходимо применить несколько способов заключения выражений, например для интерпретации переменных и выполнения команд, допускается использовать группы завершающих последовательностей ‘’ и “”. В следующем примере приведено комплексное регулярное выражение:

```
$ egrep  'username:'`whoami`' and home directory
is '"$HOME" /opt
```

В данном примере итоговое выражение будет иметь вид “**username:voitov and home directory is /home/voitov**”, при условии, что вывод команды **whoami** выведет пользователя **voitov**, а переменная окружения **HOME** будет иметь значение **/home/voitov**. Здесь же приведен пример переноса регулярного выражения на следующую строку.

Разберем несколько простых примеров использования команд семейства **grep**. Рассмотрим файл **data.f** следующего содержания:

```
data.f:
48    dec    3BC1997       LPSX       68.00      LVX2A      138
483   sept   5AP1996       USP        65.00      LVX2C      189
47    oct    3ZL1998       LPSX       43.00      KVM9D      512
219   dec    2CC1999       CAD        23.00      PLV2C      68
484   nov    7PL1996       CAD        49.00      PLV2C      234
483   may    5PAZ998       USP        37.00      KVM9D      644
216   sept   3ZL1998       USP        86.00      KVM9E      234
```

Для определения числа строк в файле data.f, соответствующих заданному шаблону, используется следующая команда:

```
$ grep -c "48" data.f
4
```

Команда **grep** возвращает число 4. Это означает, что в файле **data.f** обнаружены 4 строки, содержащие последовательность символов «48». Следующая команда отображает эти строки:

```
$ grep "48" data.f
48    dec     3BC1997        LPSX       68.00      LVX2A      138
483   sept    5AP1996        USP        65.00      LVX2C      189
484   nov     7PL1996        CAD        49.00      PLV2C      234
483   may     5PAZ998        USP        37.00      KVM9D      644
```

С помощью опции **-n** выводимые строки можно пронумеровать. В результате можно с легкостью устанавливать, в какой позиции файла находится требуемая строка. Например:

```
$ grep -n "48" data.f
1:48     dec    3BC1997      LPSX       68.00      LVX2A      138
2:483    sept   5AP1996      USP        65.00      LVX2C      189
5:484    nov    7PL1996      CAD        49.00      PLV2C      234
6:483    may    5PAZ998      USP        37.00      KVM9D      644
```

Предположим, требуется извлечь из файла строки заказов, сделанных в городах, код которых равен 483 или 484. Поставленную задачу решает следующая команда:

```
$ grep '48[34]' data.f
2:483    sept   5AP1996      USP        65.00      LVX2C      189
5:484    nov    7PL1996      CAD        49.00      PLV2C      234
6:483    may    5PAZ998      USP        37.00      KVM9D      644
```

Следующая команда находит в файле **data.cf** строки, не начинающиеся с цифры 4 или 8:

```
$ grep '^[^48]' data.f
219   dec     2CC1999        CAD        23.00      PLV2C      68
216   sept    3ZL1998        USP        86.00      KVM9E      234
```

Представленная ниже команда находит все заказы, которые были сделаны в 1996 или 1998 году и коды которых начинаются с цифры 5:

```
$ grep '5..199[68]' data.f
483   sept    5AP1996        USP        65.00      LVX2C      189
483   may     5PAZ998        USP        37.00      KVM9D      644
```

Структура используемого здесь шаблона такова: первым символом является цифра 5, за ней следуют два произвольных символа, затем число 199, а последним символом может быть либо цифра 6, либо цифра 8.

Поиск всех заказов, сделанных в 1998 году, выполняется посредством команды:

```
$ grep '[0-9]\{3\}8' data.f
47    oct     3ZL1998        LPSX       43.00      KVM9D      512
483   may     5PAZ998        USP        37.00      KVM9D      644
216   sept    3ZL1998        USP        86.00      KVM9E      234
```

Примененный в этом примере шаблон означает: найти любую последовательность из трех цифр, за которой идет цифра 8.

Модуль 8. Редактирование текста: редакторы vi и vim

Изучив данный модуль, вы научитесь:

- выполнять базовые операции по редактированию файлов;
- настраивать редакторы vi и vim.

Для редактирования текста в ОС Linux можно использовать различные текстовые редакторы, работающие как в графическом, так и в консольном режиме. Два основных текстовых редактора, доступных изначально, – **emacs** и **vi**. Оба они имеют встроенную проверку синтаксиса, автодополнение, «горячие» клавиши и предлагают неограниченные возможности настройки, которые увеличивают скорость редактирования текста.

Основным редактором, которым будет осуществляться редактирование файлов в практических работах и упражнениях по данному курсу, является редактор **vi** или его улучшенная версия **vim**.

Vim (сокр. от ***Vi Improved***) – свободный режимный текстовый редактор, созданный на основе более старого **vi**. Ныне это один из мощнейших текстовых редакторов с полной свободой настройки и автоматизации и созданным благодаря этому расширениям и надстройкам.

Существует и модификация для использования в графическом оконном интерфейсе – **GVim**. Многие пользовательские команды в **GVim** могут вызываться через соответствующие пункты меню.

В сравнении с классическим редактором **vi** редактор **Vim** отличается следующими основными улучшениями:

- работа с несколькими файлами одновременно. Разбиение окон редактирования может производиться многократно как по горизонтали, так и по вертикали;
- поддержка кодировки **Unicode**;
- поддержка визуального режима, который позволяет, например, выполнять операции над блоками текста;
- неограниченная глубина отмены (**undo**) и возврата (**redo**) действий;
- широкая файловая поддержка (файл со справкой и более 200 файлов с описанием синтаксиса);
- подсветка синтаксиса, автоматическое определение величины отступа для каждой строки в зависимости от языка программирования (поддерживает более 200 языков программирования и форматов конфигурационных файлов);

- интеграция с операционной системой, дающая возможности, близкие к интегрированным средам разработки, такие как поиск ошибки по сообщению компилятора, автодополнение идентификаторов и др.;
- поддержка языка сценариев; возможность написания модулей расширения – плагинов;
- автоматическое продолжение команд, слов, строк целиком и имен файлов.

8.1. Режимы работы редакторов vi и vim

В отличие от многих привычных редакторов, **vi** имеет **модальный** интерфейс. Это означает, что одни и те же клавиши в разных режимах работы выполняют разные действия. В редакторе **vi** есть три основных режима: командный режим, режим вставки и режим редактора **ex**. По умолчанию работа начинается в командном режиме. В режиме вставки клавиатура используется для набора текста. Для выхода в командный режим используется клавиша **<ESC>** или комбинация **<Ctrl+c>**. Для перехода в режим редактора **ex** используется символ «**:**».

В любой произвольный момент времени редактор **vi** находится в одном из следующих режимов.

1. **Режим ввода общих команд.** Загрузка в данный режим осуществляется сразу при открытии файла, если при запуске редактора не заданы специальные опции. Режим общих команд ожидает ввода специальных команд, обычно являющихся единичными символами. Например, символ **i** используется для перехода в режим вставки текста в текущее положение курсора; символ **a** используется для перехода в режим вставки текста в конец текущей строки; символ **o** используется для вставки пустой строки после текущей.
2. **Режим ввода системных команд (Ex)** используется для осуществления операций над файлами (сохранение, копирование, выполнение дополнительных действий, установка настроек редактора). Вход в данный режим происходит после ввода символа «**:**» (**двоеточие**); сразу после этого символа необходимо ввести соответствующую команду. Например, для сохранения файла необходимо ввести команду **w**; для выхода из редактора необходимо ввести команду **q**.
3. **Режим вставки текста.** В данном режиме осуществляется стандартное редактирование текста, включающее в себя ввод символов с клавиатуры.
4. **Визуальный режим (только в vim)**. Редактор vim включает в себя дополнительный режим работы. Он позволяет выделять блоки текста, которые затем можно использовать в качестве объекта для выполнения операций редактирования, таких как удаление или копирование. В графическом редакторе **GVim** для этих целей используется мышь.

8.2. Основные команды редакторов vi и vim

При работе с редактором **vi** используются команды, которые условно можно разделить на четыре группы:

- команды перемещения;

- команды вставки;
- команды редактирования;
- команды сохранения и выхода.

Данные группы команд редактора **vi** приведены в приложении 8.1.

Рассмотрим основные из этих команд более подробно.

8.2.1. Перемещение по тексту

Команд в командном режиме немало – почти все буквы и множество сочетаний вида <**Ctrl**+**буква**>. Даже для перемещения влево, вниз, вверх и вправо на одну позицию зарезервированы буквы **hjkl**. Это полезно в случае, если на клавиатуре отсутствуют стрелки. Клавиши **h** и **l** означают соответственно «влево» и «вправо» (вместо них можно использовать привычные Backspace и пробел), клавиша **j** предназначена для перемещения вниз; тогда k – для перемещения вверх. В **vi** очень много команд **перемещения** по тексту. ***Перемещение*** – это смена точки редактирования, которая чаще всего привязана к какому-нибудь элементу структуры текста: следующее слово, предложение, экран и т. п.

Например, перемещение по словам – **w** (word, переход на начало следующего слова), **b** (**b**ack word, на начало предыдущего слова) и **e** (end, на конец текущего слова) – стандартные команды перемещения в **vi**. ***Словами*** в этом случае считаются цепочки букв, цифр и знаков «_». Прописные **W**, **B** и **E** считают словами цепочки любых неразделенных записей.

Перемещать курсор можно по ***предложениям*** (команды «**)**» и «**(**»). Предложения заканчиваются соответствующим знаком препинания и разделителем после него. Перемещение по **абзацам** – команды «**}**» и «**{**» (абзацы разделяются пустой строкой; именно пустой: строка, содержащая одни только пробелы или табуляции, не подходит).

Поиск шаблона в тексте – тоже команда перемещения. Задать шаблон и поискать его можно с помощью команды «/». Для ввода шаблона **vi** использует командную строку, чтобы пользователю было видно, что он набирает. ***Шаблон*** – это, конечно, регулярное выражение, но можно **выключить** настройку **magic (:set nomagic)**, тогда любой символ в шаблоне будет считаться обычным. **Vi** ищет **вперед** по тексту; если шаблон не найден, выводится сообщение и поиск продолжается с **начала** файла. Поиск **назад** по тексту задается командой «**?**». Повторный поиск того же самого шаблона в ту же сторону – команда **n** (**n**ext), а поиск того же шаблона **в другую сторону** – **N**.

Любая команда **vi** может иметь множитель. ***Множитель*** – это **число**, которое пользователь набирает в командном режиме непосредственно перед командой, и в результате команда выполняется указанное число раз. Число это **не отображается** на экране: подобно имени команды, множитель – это часть **уже решенной** пользователем мини-задачи редактирования; ее внешний вид имеет значение только для редактора, пользователя же интересует результат. Например, команда **10x** удалит десять символов справа от курсора, а команда **7+** переместит курсор на

семь строк вперед («+» и «-» – команды перемещения на начало следующей или предыдущей строки). Команда **G** (**G**o) без множителя – переход на **последнюю** строку файла (незаменима при дописывании текста), а для перехода на строку с **номером 33** используется команда **33G**.

В любом случае можно экспериментировать с **одной** командой, как угодно: даже в самой ранней версии **vi** есть команда **u** – **u**ndo (отмена). Отменить можно только одно – последнее действие с текстом. Если соблюдать некоторую внутреннюю дисциплину и не совершать **необдуманных** действий, одного уровня должно быть достаточно. В **vim**, в отличие от **vi**, присутствует бесконечный **undo**.

8.2.2. Редактирование текста

Основные команды редактирования **vi**, которыми часто придется пользоваться, – **комбинированные**. Комбинированная команда в самом простом виде состоит из ***действия*** и команды перемещения. В этом случае команда перемещения не изменяет положения курсора, а определяет фрагмент текста, к которому **vi** применит действие. Например, операция **dw** (**d**elete **w**ord) состоит из действия **d** (**d**elete) и команды **w**, в результате удалится одно слово. Другая комбинированная команда – **c** (**c**hange) – аналогична **d**; единственное отличие – после выполнения **c** редактор переходит в режим вставки. Это очень важная команда **vi**, потому что с ее помощью можно исключить одно неудобство, свойственное практически всем текстовым редакторам с упрощенной концепцией управления.

В типичном текстовом редакторе есть два режима ввода текста – **вставка** и **замена**. И в ситуации, когда одно слово надо заменить другим, пользователь должен или сначала удалить это слово (нажать на клавишу **<Del>**), а затем в режиме вставки вписать новое слово, или начать вписывать слово в режиме замены, а когда новое станет длиннее старого, переключиться в режим вставки. В редакторе **vi** для замены слова используется команда **cw** (**c**hange **w**ord), после чего необходимо просто вписать требуемый текст. Классический **vi** поставит (временно) в конце заменяемой области символ **$**, и при наборе текст до **$** будет заменяться, а после – вставляться. В **vim** такое свойство можно **включить** одним из значений настройки **cpoptions**, но по умолчанию он сразу удаляет заменяемый участок текста.

Перемещением в комбинированной команде может быть не только **w**, но любая команда перемещения. Например, довольно удобны команды вида **c/шаблон** (изменить текст от курсора до найденного шаблона) или **d10G** (удалить строки от текущей до десятой). Между прочим, смысловые названия команд **vi** позволяет **читать** их полные имена: например, предыдущая команда читается **delete 10 Go**, а команда **c2w** – **change 2 words**.

Для вызова внешних утилит используется команда «**!**». После команды перемещения она спросит в командной строке имя утилиты. Строки, определяемые командой перемещения, **vi** передаст этой утилите, а результат подставит вместо них. Например, для того чтобы отсортировать все строки до конца файла, достаточно ввести команду **!Gsort**.

8.2.3. Операции с файлами

Редактор **vi** позволяет открывать файлы в командном режиме, а также сразу указывать открываемый файл при запуске из командной строки. Вот пример открытия файла **/etc/grub.conf**:

```
$ vi /etc/grub.conf
```

Если необходимо отредактировать другой файл во время текущего сеанса работы в **vi**, можно использовать команду **:e**, чтобы открыть другой файл для редактирования:

```
~
~
:e new_file.txt
```

Если редактор **vi** запускался с параметром **<имя файла>**, то, выполнив команду **:w** и нажав клавишу **<Enter>**, можно сохранить результат изменений. При вводе символа «**:**» в командном режиме курсор переместится в нижнюю часть экрана. Здесь можно будет вводить команды редактора **ex**.

Если редактор **vi** запускался без параметров, то необходимо определить имя файла:

```
~
~
:w new_file.txt
```

Можно использовать этот метод, чтобы сохранять файл под другим именем. Если имя файла, которое было указано, существует, то **vi** не перезапишет существующий файл. В таком случае необходимо использовать команду **:w!**, чтобы перезаписать существующий файл.

Со временем, возможно, вам потребуется вставить содержимое одного файла в текущий файл. В этом случае можно использовать **:r**, например:

```
~
~
:r anotherfile.txt
```

Для этого необходимо установить курсор на строке, где необходимо вставить файл, а затем нажать **:r**, ввести имя файла и нажать на клавишу **<Enter>**. Для того чтобы выйти из редактора **vi**, используется команда **:q**. Если файл изменен и изменения не были сохранены, **vi** не позволит выполнить выход. Для того чтобы выйти, не сохраняя результатов, необходимо воспользоваться командой **:q!**.

Существуют дополнительные возможности, позволяющие существенно облегчить редактирование файлов. Для этого необходимо выполнить соответствующую команду редактора **ex** из следующего списка:

- **:<number>** – перейти на строку с номером **<number>**;
- **:set number** – отобразить слева нумерацию строк (**:set nonumber** – спрятать нумерацию);
- **:set wrap** – переносить длинные строки (**:set nowrap** – не переносить);
- **:sy[ntax] on/off** – включить/выключить подсветку синтаксиса;

- **:colorscheme <name>** – задать цветовую тему (где **<name>** – имя темы, **<TAB>** работает как автодополнение);
- **:h** или **:help** – список возможной помощи (**:viusage**, :**exusage**);
- **:set fileformat=dos** – привести концы строк в файле к виду **dos**;
- **:set fileformat=unix** – привести концы строк в файле к виду **unix**;
- **:set ts=4** – задать размер табуляции в 4 пробела;
- **:$** – переместиться в конец файла.

Одним из основных отличий редактора **vim** от **vi** является наличие дополнительного режима работы – **визуального** режима. Данный режим предназначен для выделения блока текста и дальнейшей работы с ним и состоит из трех последовательных шагов:

- пометка начала блока с помощью команд **v**, **V** или **<Ctrl+V>**. Блок помечается с того символа, на котором находится курсор (по умолчанию);
- перемещение курсора в конец необходимого блока. Последний символ также будет включен в выделенный блок;
- вызов необходимой команды.

Команда «**v**» предназначена для выделения текста посимвольно. Символы до и после курсора в строке не будут входить в выделение. Данная команда необходима в первую очередь для копирования участков текста в строке.

Команда **V** необходима для выделения текста построчно. Команда **<Ctrl+V>** – для выделения прямоугольного участка текста, она удобна в случае редактирования структурированного текста.

С выделенным участком текста можно производить стандартные действия по редактированию (копирование – «**y**», удаление – «**d**»). В редакторе **vim** при удалении участка текста он автоматически помещается в специальный буфер, что равноценно «вырезанию» участка текста, который потом можно будет вставить с помощью команды «**p**» (вставка текста после курсора) или «**P**» (вставка текста перед курсором).

Еще одной замечательной особенностью редактора **vim** является возможность работы с окнами.

Эта возможность особенно полезна, если необходимо выполнить:

- редактирование одного файла по образу другого;
- копирование и вставку текста между разными файлами;
- сравнение двух файлов.

Редактор **vim** позволяет выполнять следующие операции с окнами:

- разбивать окна по горизонтали или вертикали;
- быстро перемещаться между окнами;
- копировать и перемещать текст между окнами;
- использовать внешние утилиты для обработки текста в окнах.

Для начала редактирования одновременно двух файлов можно использовать следующую команду:

```
$ vim -o2 file1 file2
search lpr.jet.msk.su service.jet.msk.su
```

```
#nameserver  192.168.11.2
nameserver 10.31.5.4
nameserver 10.30.1.144
#nameserver 192.168.11.136
~
~
/etc/resolv.conf                                                    1,1
All
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1       localhost.localdomain   localhost       oit11-4
192.168.11.135  debian
192.168.50.254  exchange
192.168.50.147  hel
~
~
/etc/hosts                                                          1,1       All
```

В данном случае были открыты два файла **/etc/resolv.conf** и **/etc/hosts**.

Ключом **-o**<N> можно задавать количество открываемых горизонтальных окон. Если сеанс редактора уже запущен и редактируется всего лишь один файл, то для редактирования другого файла в новом окне используется команда **:split <файл>**, в которой необходимо указать путь до другого файла. Если необходимо выполнить вертикальное разбиение окна, то вместо команды split используется команда **:vsplit** с теми же параметрами.

Переключение между окнами выполняется при помощи последовательных нажатий на клавиши **<Ctrl+W>**. Для того чтобы отобразить названия всех открытых файлов, а также строки, на которых выполняется редактирование, используется команда **:ls**.

Для закрытия окон используются команды **:quit[!]** (Выйти из текущего окна), **:close[!]** (Закрыть текущее окно) и **:only[!]** (Сделать текущее окно единственным открытым).

8.3. Настройка редакторов vi и vim

Настройку редакторов vi и vim можно выполнять непосредственно в режиме редактирования файлов при помощи команды **:set** или же заранее в описании соответствующих конфигурационных файлов **~/.exrc** и **~/.vimrc** для редакторов **vi** и **vim** соответственно. В конфигурационных файлах содержатся те же команды **:set**, но без двоеточия. Основные способы вызова команды **:set** в режиме редактирования следующие:

- **:set <X>** – включить опцию X;
- **:set no<X>** – включить опцию X;
- **:set X=значение** – установить значение для опции X;
- **:set** – вывести измененные опции;
- **:set all** – вывести все опции;
- **:set <X>?** – вывести значение опции X.

В следующем листинге приведен пример конфигурационного файла **exrc**:

```
set scroll=15          # количество строк прокрутки (команды ^D и ^U)
set shiftwidth=2       # количество пробелов при уменьшении или увеличении
                       # отступа
set showmatch          # перемещаться к завершающим символам ) и } при нажатии
                       # на эти клавиши
set showmode           # отображать текущий режим выполнения вставки, например
                       # OPEN MODE или APPEND MODE
set terse              # выводить короткие сообщения об ошибках
set directory=/tmp     # место размещения временных файлов
```

Файл **vimrc** имеет идентичный формат записей. В следующем листинге представлен пример данного файла:

```
set keymap=russian-jcukenwin          # установить keymap, чтобы по Ctrl+^
                                      # переключался на русский и обратно
set ic                                # игнорировать регистр при поиске
set hls                               # подсвечивать поиск
set guifont=courier_new:h10:cRUSSIAN  # установить шрифт Courier New Cyr
set ts=4                              # задать размер табуляции в четыре пробела
```

Для получения более подробной информации по файлу **vimrc** можно воспользоваться встроенной командой **:help vimrc**. В обычном режиме **vim** по умолчанию ожидает, что консоль находится в режиме ввода латинских символов. Если, например, выполняется редактирование с помощью **vim** текста на русском языке или в смешанных кодировках (например, HTML-страницы на русском), то постоянная необходимость переключать системную клавиатурную раскладку очень быстро надоедает. Для облегчения этого момента в файл **.vimrc** нужно добавить следующие строки:

```
set keymap=russian-jcukenwin
set iminsert=0
```

После этого системную раскладку клавиатуры можно будет оставить в режиме ввода латинских символов, а переключение между языками осуществлять уже внутри самого редактора с помощью команды **<Ctrl+^>**.

Редактор **Vim** автоматически конвертирует открываемый файл в текущую кодировку, а при сохранении конвертирует обратно в кодировку файла. Текущая кодировка задается переменной **encoding**(enc). Кодировка файла задается переменной **fileencoding**(fenc). Кодировка файла автоматически определяется при его открытии, но ее можно и задать вручную. Для того чтобы сохранить файл в нужной кодировке, надо перед сохранением задать соответствующее значение переменной **fileencoding**.

Чтобы узнать список всех доступных значений переменных **encoding** и **fileencoding**, используется следующая команда:

```
:help encoding-values
```

Модуль 9. Работа с учетными записями

Изучив данный модуль, вы научитесь:

- различать пользователей системы и определять их предназначение;
- запускать команды от имени других пользователей;
- выполнять административные задачи по управлению учетными записями.

9.1. Пользователи и группы в ОС Linux

В ОС Linux существуют три типа учетных записей пользователей: **административные** (root), **регулярные** (обычные пользователи) и **сервисные** (служебные).

Учетная запись, **account** – объект системы, при помощи которого ОС Linux ведет учет работы пользователя в системе. Учетная запись содержит данные о пользователе, необходимые для регистрации в системе и дальнейшей работы с ней.

Учетные записи могут быть созданы во время установки системы или после установки. Главное для человека в учетной записи – ее название, входное имя пользователя. Именно о нем спрашивает система, когда выводит приглашение «**login:**». Помимо входного имени, в учетной записи содержатся некоторые сведения о пользователе, необходимые системе для работы с ним. Основной **административной** учетной записью в ОС Linux является запись пользователя **root**, которая имеет неограниченные административные права. **Регулярные** учетные записи имеют ограничения на запуск программ – они позволяют запускать стандартные программы (редакторы текста, просмотрщики графических файлов и документов и другие программы неадминистративного назначения). Регулярные пользователи могут создавать файлы только в своих домашних каталогах и не имеют доступа к системным конфигурационным файлам, если на то не были даны им соответствующие права. Помимо административных и регулярных учетных записей, существуют **сервисные** учетные записи. Почтовые сервисы, веб-сервисы, сервисы баз данных выполняются из-под специальных учетных записей с минимально необходимыми привилегиями. Такие учетные записи, как правило, предполагают доступ только к тем файлам и запуск только тех программ, которые необходимы для корректной работы сервиса. Если имеется возможность настроить работу сервиса не из-под административной учетной записи, то лучше это сделать для повышения безопасности системы в целом.

Каждая учетная запись в ОС Linux имеет набор данных, отличающих ее от других учетных записей. Среди этих данных можно выделить:

- **Регистрационное имя пользователя.** Каждый пользователь в системе имеет свое собственное регистрационное имя, предназначенное для простого определения владельца ресурса файловой системы, прав на этот ресурс, а также для регистрации пользователя в системе (аутентификации).
- **Идентификатор пользователя.** ОС Linux связывает **входное имя** с **идентификатором пользователя** в системе – **UID** (User ID). **UID** – это положительное целое число, по которому система отслеживает пользователей. Обычно это число выбирается автоматически при регистрации учетной записи, однако оно не может быть произвольным. В ОС Linux есть некоторые соглашения относительно того, каким типам пользователей могут быть выданы идентификаторы из того или иного диапазона. В частности, **UID** от 0 до 500 зарезервированы для **системных пользователей**. Максимальное значение идентификатора – 65535.
  Идентификатор пользователя, UID – **уникальное число, однозначно идентифицирующее** учетную запись **пользователя в системе. Таким числом снабжены все процессы ОС Linux и все объекты файловой системы. Идентификатор используется для персонального учета действий пользователя и определения прав доступа к другим объектам системы.**
- **Идентификатор группы.** Кроме идентификационного номера пользователя, с учетной записью связан ***идентификатор группы***. Группы пользователей применяются для организации доступа нескольких пользователей к некоторым ресурсам. У группы, так же как и у пользователя, есть имя и идентификационный номер – **GID** (Group ID). В ОС Linux пользователь должен принадлежать как минимум к одной группе – группе по умолчанию. При создании учетной записи пользователя обычно создается и группа по умолчанию, имя которой совпадает с входным именем пользователя. Пользователь может входить более чем в одну группу, но в учетной записи указывается только номер группы по умолчанию.
- **Домашний каталог.** Файлы всех пользователей в ОС Linux хранятся раздельно, у каждого пользователя есть собственный домашний каталог, в котором он может хранить свои данные. Доступ других пользователей к домашнему каталогу пользователя может быть ограничен. Информация о домашнем каталоге обязательно должна присутствовать в учетной записи, потому что именно с него начинает работу пользователь, зарегистрировавшийся в системе.
- **Командный интерпретатор.** Каждому пользователю нужно предоставить способ взаимодействия с системой: передача ей команд и получение от нее ответов. Для этой цели служит специальная программа – командный интерпретатор. Она должна быть запущена для каждого пользователя, который зарегистрировался в системе. Поскольку в ОС Linux доступны несколько разных интерпретаторов, в учетной записи указано, какой из них нужно запустить для данного пользователя. Если специально не указывать командную оболочку при создании учетной записи, она будет назначена по умолчанию, вероятнее всего, это будет **bash**.

9.2. Регистрация и смена пароля

Для регистрации в системе пользователь должен ввести в приглашении **login** – свое регистрационное имя и пароль. В случае успешной аутентификации пользователь попадает в графическое окружение или же в режим командной строки. Конечно, процедура аутентификации имеет очевидное значение для систем, к которым имеют непосредственный или сетевой доступ многие не связанные друг с другом пользователи. Для тех пользователей, которым процедура идентификации необязательна (например, единственным пользователям персональных компьютеров), существует возможность получить доступ к системе, минуя процедуру идентификации. Для этого применяется программа **autologin**. Она предоставляет доступ к работе с графическим интерфейсом сразу после загрузки системы, не запрашивая имени пользователя и пароля. В действительности **autologin** запускает все программы от имени одного пользователя, зарегистрированного в системе.

После ввода регистрационного имени пользователю необходимо ввести свой **пароль**. Поскольку пароль – это единственная гарантия, что вашей учетной записью не воспользуется никто, кроме вас, есть смысл выбирать в качестве пароля неочевидные последовательности символов. В ОС Linux нет существенных ограничений на длину пароля или входящие в него символы (в частности, использовать пробел можно), но нет смысла делать пароль слишком длинным – велика вероятность его забыть. Надежность паролю придает его непредсказуемость, а не длина. Пользователь может в любой момент изменить свой пароль. Единственное, что требуется для смены пароля, – знать текущий пароль. Для смены пароля используется команда **passwd**. Пароль пользователя root задается при установке системы, однако пользователь **root** может изменить его впоследствии самостоятельно.

9.3. Запуск программ от имени других пользователей

Для выполнения большого количества административных задач, таких как настройка сетевых интерфейсов, изменение параметров ядра, настройка оборудования, в ОС Linux требуются административные права. Регулярно работать с ОС Linux из-под учетной записи пользователя root не рекомендуется, так как это может скомпрометировать безопасность ОС Linux или стать критической ошибкой в процессе неправильного администрирования. Для этого почти во всех дистрибутивах ОС Linux существуют команды, позволяющие овладеть правами суперпользователя root на некоторое время. Для выполнения административных задач от имени суперпользователя или любого другого пользователя в ОС Linux существуют три способа:

- выполнить вход в систему под учетной записью root и продолжать работать;
- использовать команду **su**, которая позволит стать на время пользователем root и выполнить необходимые административные действия;

- использовать команду **sudo** и просто выполнить необходимую команду, не становясь суперпользователем.

Команда **su** (от англ. ***S**ubstitute **U**ser*) – команда, позволяющая пользователю войти в систему под другим пользователем, не завершая текущего сеанса. Команда имеет следующий синтаксис:

```
su [-] [имя_пользователя [аргументы ... ]]
```

Типичным примером запуска сеанса нового пользователя является следующая команда:

```
$ su -
```

При этом заново устанавливаются переменные окружения **$HOME** и **$PATH**, то есть домашним каталогом становится **/root**, а в **$PATH** добавляются **/sbin** и **/usr/sbin**. При запуске **su** без дефиса переменная **$HOME** остается равной домашнему каталогу пользователя, который запускал эту команду, и в результате работы программ, запущенных из-под **su**, туда могут быть записаны файлы (например, конфигурации этих программ) с правами **root**, что может привести к ошибкам и странностям в дальнейшей работе.

Для использования команды **su** необходимо ввести соответствующий пароль (если только команду не выполняет пользователь **root**). Если введен правильный пароль, команда **su** создает новый процесс командного интерпретатора, с такими же реальными и эффективными идентификаторами пользователя и группы, а также списком дополнительных групп, как у указанного пользователя. В качестве нового командного интерпретатора используется указанный в файле **/etc/passwd** для соответствующего пользователя. Если командный интерпретатор не указан, используется **/usr/bin/sh**. Чтобы вернуться в исходный сеанс, необходимо нажать на клавиши **<Ctrl+D>**.

Любые дополнительные `аргументы`, заданные в командной строке, передаются новому командному интерпретатору. При использовании программ типа sh, если аргумент имеет вид **-c <строка>**, строка выполняется интерпретатором как команда.

Если первый аргумент команды **su** – дефис (**-**), будет установлена среда переменного окружения, такая же, как у указанного пользователя. Иначе передается текущая среда, за исключением значения **$PATH**, которое задается переменными **PATH** и **SUPATH** в файле **/etc/default/su**.

Команда **sudo** позволяет запустить одиночную команду с правами другого пользователя (по умолчанию – root), при этом вводится пароль (если не включен беспарольный запуск) пользователя, который **запускает sudo** (а не того, от чьего имени запускается команда, в отличие от **su**). По умолчанию никто не имеет права запускать что-либо. Поэтому для начала использования **sudo** необходимо ее настроить. Настройки хранятся в файле **/etc/sudoers**. О настройках **sudo** можно почитать в руководстве **man(5) sudoers**. Кроме того, пример правил, предоставляющих пользователям, являющимся членами группы **rpm**, возможность устанавливать, обновлять и удалять пакеты в системе, приведен в файле **/usr/share/doc/sudo-<версия>/rpm.sudoers**. В большинстве случаев грамотная настройка **sudo** делает работу от имени суперпользователя ненужной.

Настройка работы утилиты **sudo** выполняется посредством правки файла **/etc/sudoers**. Редактирование данного файла выполняется командой **visudo**, которая проверяет синтаксис файла перед его сохранением. Данная команда позволяет осуществить безопасное редактирование **/etc/sudoers** и имеет следующие ключи:

- **c** [-q] – проверить существующий файл без вызова редактора;
- **f** <имя_файла> – указать другой файл **sudoers**;
- **s** – выполнять более строгую проверку файла **sudoers**.

В файле **/etc/sudoers** описываются права пользователей на выполнение команд с помощью **sudo**. Права состоят из операторов трех типов: определение синонимов (**Alias**), переопределение конфигурационных параметров и описание прав пользователей. Если для одного пользователя подходят несколько описаний, то действует последнее. Комментарии начинаются с символа «**#**», если это не **UID**. Продолжение команды на следующей строке обозначается символом «\» в конце строки.

Синонимы используются для определения пользователей и групп, а также всех остальных субъектов и объектов ОС. Существуют следующие типы синонимов:

- **User_Alias** <имя> = <список-пользователей-через-запятую>;
- **Runas_Alias** <имя> = <список-пользователей-через-запятую>;
- **Host_Alias** <имя> = <список-хостов-через-запятую>;
- **Cmd_Alias** <имя> = <список-команд-через-запятую>.

Имя синонима может состоять из прописных латинских букв, цифр и подчеркивания. Зарезервирован синоним «**ALL**» для каждого типа. На одной строке может размещаться несколько описаний синонимов, разделенных символом «**:**».

Пользователь может определяться (восклицательный знак перед именем означает отрицание) следующими способами:

- **имя**;
- **%имя-группы**;
- **User_Alias**.

Эффективный пользователь может определяться (восклицательный знак перед именем означает отрицание) следующими способами:

- **имя**;
- **#uid**;
- **%имя-группы**;
- **Runas_Alias**.

Хост может определяться (восклицательный знак перед именем означает отрицание; сетевая маска записывается в точечной записи или **CIDR**) следующими способами:

- **имя-хоста**;
- **IP-адрес**;
- **network/netmask**;
- **Host_Alias**.

Команда может определяться (восклицательный знак перед именем означает отрицание; разрешает выполнить любую команду из данного каталога, но не из подкаталога):

- **полное-имя-файла**;
- **полное-имя-файла аргументы**;
- **каталог**;
- **Cmnd_Alias**.

Конфигурационные параметры можно переопределить для всех пользователей, для определенных пользователей, для определенных хостов, для команд, выполняемых от имени указанного пользователя:

- **Defaults <список-параметров-через-запятую>**;
- **Defaults:<имя-пользователя> <список-параметров-через-запятую>**;
- **Defaults@<имя-хоста> <список-параметров-через-запятую>**;
- **Defaults><имя-пользователя> <список-параметров-через-запятую>**;

Параметр задается в виде **«имя = значение»**, **«имя += значение»**, **«имя -= значение»**, **«имя»** или **«!имя»**.

Описание прав пользователей (списки через запятую, теги и имена целевых пользователей наследуются) задается следующим образом:

- **<список-пользователей> <список-хостов> = <список-описаний-команд> [: <список-хостов> = <список-описаний-команд>]**;
- **<описание-команды> ::= [(<список-целевых-пользователей>)] {тег:} команда**.

Параметр «**тег**» может иметь следующие значения: **NOPASSWD** (не запрашивать пароль текущего пользователя), **PASSWD** (запрашивать пароль текущего пользователя), **NOEXEC** (запретить выполнение), **EXEC** (разрешить выполнение).

Рассмотрим несколько наиболее простых и интересных примеров. В следующем листинге приведен пример разрешения всем пользователям со всех хостов выполнять все, что только можно выполнить.

```
ALL ALL = (ALL) ALL
```

Рассмотрим, каким образом можно ограничить эту свободу, наложив на пользователя определенные ограничения.

1-е поле в строке – это список пользователей и (или) групп, к которым применяется данная строка, например можно указать следующее:

```
%wheel ALL = (ALL) ALL
```

Здесь разрешается всем членам группы **wheel** запуск любой команды от любого имени. Так более правильно, при этом в группу **%wheel** включаются только разрешенные пользователи.

2-е поле – имя хоста, с которого можно выполнять команды. Имеет значение только при использовании сетевого входа в систему.

3-е поле (в скобках) – список пользователей, **от имени которых** можно выполнять команду. Так как обычно команду **sudo** используют для запуска программ от имени **root**, это поле используется не очень часто.

4-е поле – команда, которую разрешено выполнять. Здесь, как и в остальных полях, можно указывать список команд через запятую.

Любой пользователь может командой sudo -l узнать, какие команды ему разрешено выполнять. При запуске команды **sudo** запрашивается пароль пользователя, запустившего команду. В течение 5 минут после последнего запуска **sudo** его можно запускать еще раз уже без пароля. Можно в файле **/etc/sudoers** в нужных строках указать тег **NOPASSWD**:, тогда команды, указанные в этой строке, могут выполняться без пароля, например:

```
ALL ALL = NOPASSWD: (ALL) /sbin/shutdown, /sbin/reboot
```

Данная запись позволяет выключать и перезагружать компьютер любому пользователю без указания пароля, командами **sudo /sbin/shutdown** и **sudo /sbin/reboot** соответственно.

По умолчанию все успешные и неуспешные попытки выполнения команды sudo (включая имя пользователя и полную команду) регистрируются в системном журнале **/var/adm/sulog** и могут быть отправлены по почте системному администратору.

При настройке утилит **su** и **sudo** в плане ограничения доступа пользователям надо быть осторожным, так как неправильно настроенная утилита **sudo** может позволить пользователю запустить с правами суперпользователя любую команду.

9.4. Управление учетными записями пользователей: файлы /etc/passwd, /etc/shadow и /etc/groups

ОС Linux предоставляет администратору два метода управления учетными записями пользователей: непосредственное редактирование конфигурационных файлов (**/etc/passwd** и **/etc/group**) или использование консольных программ (**useradd**, **groupadd**). Кроме того, в ОС Linux существует графическая программа управления пользователями, для ее использования необходимо установить пакет **system-config-users** и запустить программу из меню **«Administration» ⇒«Users and Groups»**.

Файл **/etc/passwd** содержит список пользователей, которые известны системе. В процессе регистрации пользователя система обращается к этому файлу в поисках идентификатора пользователя и его домашнего каталога. Каждая строка файла описывает одного пользователя и содержит семь полей, разделенных двоеточиями.

1. **Регистрационное имя** должно быть уникальным и состоять не более чем из 32 символов. Оно может содержать любые символы, кроме двоеточия и символа новой строки. Регистрационное имя не должно начинаться с цифры.
2. **Зашифрованный пароль, или «заполнитель» пароля**. В данном поле возможно присутствие символа «x», символа «*» или набора случайных символов и чисел. В первом случае указывается, что пароль хранится в файле **/etc/shadow**. Во втором случае говорится, что данная учетная запись вре-

менно отключена. В последнем случае указывается истинный пароль пользователя в зашифрованном виде.

3. **Идентификатор пользователя** – 32-битное целое число, которое уникально идентифицирует пользователя в системе. По умолчанию идентификаторы меньше 500 зарезервированы для служебных учетных записей. Регулярные пользователи имеют идентификаторы, начиная с 500.
4. **Идентификатор группы по умолчанию (*GID*)** – 32-битное целое число. Идентификатор 0 зарезервирован для группы с именем **root**, идентификатор 1 – для группы **bin**, а 2 – для группы **daemon**. Группы определяются в файле **/etc/group**, а поле идентификатора группы в файле **/etc/passwd** задает стандартный («эффективный») идентификатор на момент регистрации пользователя в системе. Этот идентификатор не играет особой роли при определении прав доступа; он используется лишь при создании новых файлов и каталогов. Новые файлы обычно включаются в эффективную группу своего владельца, но если у каталога установлен специальный бит **setgid** (02000) или файловая система смонтирована с опцией **grpid**, новые файлы принадлежат группе владельца каталога. По умолчанию в ОС Linux для каждого нового пользователя создается новая группа с таким же регистрационным именем и идентификатором.
5. **Поле персональных данных.** Данное поле используется главным образом для хранения персональной информации о каждом пользователе. Оно не имеет четко определенного синтаксиса, его структура может быть произвольной, но команда **finger** интерпретирует разделенные запятыми элементы поля в следующем порядке: полное имя; номер офиса и здания; рабочий телефон; домашний телефон.
6. **Домашний каталог.** Войдя в систему, пользователь попадает в свой домашний каталог. Если на момент регистрации этот каталог отсутствует, выводится соответствующее сообщение. Если в качестве значения поля домашнего каталога по умолчанию **DEFAULT_HOME** в файле **/etc/login.defs** установлено **«no»**, продолжение регистрации пользователя будет невозможно; в противном случае пользователь попадает в корневой каталог. По умолчанию домашние каталоги пользователей создаются в каталоге **/home**. При добавлении в систему нового пользователя в его домашний каталог копируются все файлы из каталога **/etc/skel**, в котором содержатся персональные конфигурационные файлы командного интерпретатора и различных программ.
7. **Командный интерпретатор**. В качестве регистрационной оболочки, как правило, задается интерпретатор команд, например **/bin/sh** или **/bin/csh**, но, в принципе, это может быть любая программа. По умолчанию используется интерпретатор **bash**. Пользователи могут менять интерпретатор с помощью команды **chsh**. Файл **/etc/shells** содержит список тех интерпретаторов, которые доступны для выбора.

Типовые записи файла **/etc/passwd** имеют следующий вид:

```
root:x:0:0:test,w,1,w:/root:/bin/bash
sabayon:x:86:86:Sabayon user:/home/sabayon:/sbin/nologin
```

В первой записи указана учетная запись пользователя **root**. Как видно, идентификатор пользователя и идентификатор группы пользователя для данной учетной записи совпадают и равны нулю. Затем следует поле персональных данных, домашний каталог, которым является каталог **/root,** и командный интерпретатор пользователя – **/bin/bash**.

Файл **/etc/shadow** доступен для чтения только пользователю **root** и предназначен для хранения зашифрованных паролей. В нем также содержится учетная информация, которая отсутствует в файле **/etc/passwd**. При использовании скрытых паролей соответствующие поля в файле **/etc/passwd** всегда содержат символ **«x»**. Оба файла необходимо сопровождать независимо друг от друга (или использовать команду **useradd** для автоматического управления файлами). Как и **/etc/passwd**, файл **/etc/shadow** содержит одну строку для каждого пользователя. Каждая строка состоит из девяти полей, разделенных двоеточиями:

1. **Регистрационное имя**, которое берется из файла **/etc/passwd**. Оно связывает записи файлов **/etc/passwd** и **/etc/shadow**.
2. **Зашифрованный пароль**.
3. **Дата последнего изменения пароля**. Данное поле обычно заполняется командой **passwd**.
4. **Минимальное число дней между изменениями пароля**. В данном поле задается количество дней, спустя которые пользователь сможет изменить свой пароль.
5. **Максимальное число дней между изменениями пароля**. Максимальное время жизни пароля определяется суммой значений данного и седьмого полей.
6. **Количество дней до истечения срока действия пароля, когда выдается предупреждение.** В данном поле задано количество дней, оставшихся до момента, когда программа **login** должна предупредить пользователя о необходимости изменить пароль.
7. **Количество дней по истечении срока действия пароля, когда учетная запись отключается.**
8. **Срок действия учетной записи.** По окончании этого срока пользователь не сможет зарегистрироваться в системе, пока администратор не сбросит значение поля. Если поле содержит пустое значение, учетная запись всегда будет активной.
9. **Зарезервированное поле**, которое в настоящее время всегда пусто.

Типовой вид записи пользователя в файле **/etc/shadow** имеет следующий вид:

```
aivanov:!!$1$bNurEkCt$rDah2BOTfXZndRWSXKWFC1:14260:0:14:7::14303:
```

Поля дат в файле **/etc/shadow** задаются в формате числа дней с 1-го января 1970 года, поэтому данные поля лучше задавать с помощью команд **usermod** или **chage**. Содержимое файлов **/etc/shadow** и **/etc/passwd** можно согласовать с помощью программы **pwconv**. Большинство параметров файла **/etc/shadow** берется из стандартных установок файла **/etc/login.defs**, поэтому для упрощения задания данных параметров следует определить их по умолчанию в данном файле. Следующий список содержит описание наиболее значимых параметров файла **/etc/login.defs**:

- PASS_MAX_DAYS – определяет максимальное количество дней действия пароля, после чего будет выполнен запрос на смену пароля;
- PASS_MIN_DAYS – определяет минимальное количество дней, по прошествии которых разрешено изменять пароль;
- PASS_WARN_AGE – определяет, через сколько дней следует выдать предупреждение для смены пароля;
- UMASK – задает маску разрешений (если не указано, используется 022);
- USERDEL_CMD – выполнять указанную команду при удалении пользователя;
- UID_MAX (number), UID_MIN (number) – определяет диапазон допустимых значений идентификатора пользователя;
- GID_MAX (number), GID_MIN (number) – определяет диапазон допустимых значений идентификатора группы.

Файл **/etc/group** содержит имена групп, присутствующих в ОС Linux, и списки членов каждой группы, например:

```
daemon:x:2:root,bin,daemon
```

В данном случае в системе имеется группа **daemon** с идентификатором, равным 2. В данную группу входят пользователи **root**, **bin** и **daemon**.

Каждая запись файла **/etc/group** представляет одну группу и содержит четыре поля:

1. **Имя группы**. По умолчанию при создании нового пользователя создается также его группа с тем же именем, что и регистрационное имя пользователя.
2. **Зашифрованный пароль**, или символ **x**, указывающий на использование файла **/etc/gshadow**.
3. **Идентификатор группы**.
4. **Список членов**, разделенный запятыми без пробелов.

Как и в файле **/etc/passwd**, поля разделяются двоеточиями. Имя группы не должно содержать больше восьми символов (из соображений совместимости, в ОС Linux такого ограничения нет). Несмотря на наличие поля пароля (с помощью которого пользователи могут присоединиться к группе, выполнив команду **newgrp**), пароль задается редко. В большинстве случаев в это поле вводятся звездочки, но можно оставить его пустым. Команда **newgrp** не сменит группу без пароля, если только пользователь не указан как член этой группы. Если в файле **/etc/passwd** пользователь объявлен членом определенной группы, а в файле **/etc/group** – нет, пользователь все равно включается в группу. На этапе регистрации членство в группе проверяется по обоим файлам, но лучше согласовывать их содержимое.

9.4.1. Управление учетными записями при помощи консольных программ

В ОС Linux существует набор программ, предназначенных для управления пользователями. Данные программы входят в состав пакета **shadow-utils**.

Использование консольных программ позволяет создавать собственные скрипты, облегчающие процесс создания учетных записей пользователей и вы-

полняющих вспомогательные проверки. Кроме того, данные программы обладают большей гибкостью, чем их графические аналоги.

Для добавления нового пользователя в систему используется команда **useradd**. Например, команда **useradd testuser** добавляет нового пользователя с именем **testuser** в систему. По умолчанию создается домашний каталог пользователя, в данном случае **/home/testuser**, в который затем копируются все файлы из каталога **/etc/skel** и добавляется командный интерпретатор **bash**. В табл. 9.1 указаны основные опции команды **useradd**.

Таблица 9.1. Основные опции команды useradd

Опция	Описание
-u <UID>	Использовать значение <UID> в качестве идентификатора пользователя
-g <GID>	Использовать значение <GID> в качестве идентификатора группы пользователя
-G <группы>	Использовать значение < группы > в качестве дополнительных групп пользователя, разделенных запятыми
-c <данные>	Использовать значение <данные> в качестве персональных данных пользователя
-d <каталог>	Использовать значение <каталог> в качестве домашнего каталога пользователя
-s <интерпретатор>	Использовать значение < интерпретатор > в качестве интерпретатора пользователя
-M	Не создавать домашний каталог пользователя
-m	Создавать домашний каталог пользователя, если таковой отсутствует
-n	Не создавать частную группу пользователя по умолчанию

При вызове команды **useradd** без параметров вызов осуществляется с параметрами, указанными в файле **/etc/defaults/useradd**. Содержание данного файла имеет следующий вид:

```
# cat /etc/default/useradd
# useradd defaults file
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
```

В данном файле параметры имеют следующее значение:

- GROUP=100 – определяет идентификатор основной группы для новых пользователей;
- HOME=/home – определяет, где создавать домашние каталоги пользователей;
- INACTIVE=-1 – деактивирует вновь созданные аккаунты до тех пор, пока не будет задан пароль;

- EXPIRE= – определяет время жизни пароля учетной записи;
- SHELL=/bin/bash – определяет командный интерпретатор по умолчанию;
- SKEL=/etc/skel – в случае запуска команды useradd с ключом –m содержимое каталога /etc/skel копируется в домашний каталог пользователя.

Для изменения параметров пользователя после его создания используется команда **usermod**, большинство основных опций которой аналогичны опциям команды **useradd**. Данная команда позволяет блокировать учетную запись при запуске с ключом **-L** и производить обратное действие при запуске с ключом **-U**. Кроме того, данная команда позволяет изменить регистрационное имя пользователя при указании опции **-l <регистрационное_имя>**.

После того как учетная запись пользователя будет создана, необходимо задать ее пароль при помощи команды **passwd <регистрационное_имя_пользователя>**. Команда **passwd** имеет следующие опции указания временных характеристик пароля:

- **n <*min*>** – аналогично параметру PASS_MIN_DAYS файла **login.defs**;
- **x <*max*>** – аналогично параметру PASS_MAX_DAYS файла **login.defs**;
- **c <*warn*>** – аналогично параметру PASS_WARN_AGES файла **login.defs**;
- **i <*days*>** – задает количество дней, по истечении которых учетная запись деактивируется в случае отсутствия всякой активности;
- **n** – задает нумерацию выводимых строк;
- **s** – подавляет вывод сообщений о несуществующих или нетекстовых файлах;
- **v** – задает отображение строк, не соответствующих шаблону.

Аналогичным вариантом изменения временных параметров учетной записи, связанных с паролем, является команда **chage**.

Для удаления учетной записи пользователя используется команда **userdel**, имеющая следующий синтаксис запуска:

```
userdell [-r] <регистрационное_имя_пользователя>
```

Частная группа пользователя также удаляется, пользователь автоматически исключается из всех групп, членом которых он являлся. Однако домашний каталог пользователя и созданные им файлы не удаляются. При указании ключа **-r** вместе с удалением учетной записи пользователя будет выполнено удаление его домашнего и почтового каталогов. После удаления учетной записи пользователя убедитесь, что в системе не осталось файлов с его идентификатором. Чтобы узнать точный путь к этим файлам, можно воспользоваться командой **find** с аргументом **-nouser**.

Для управления группами в ОС Linux используется команда **groupadd**, имеющая следующий синтаксис:

```
groupadd [-r] [-g <GID>] <имя_группы>
```

Если в аргументах данной команды указать только имя группы, то по умолчанию идентификатор группы будет исчисляться от 500 и увеличиваться на единицу при добавлении каждой новой группы. Чтобы задать желаемый идентифика-

тор группы, нужно использовать ключ **-g**. Для создания служебной группы используется ключ **-r**. При указании ключа **-r** вместе с ключом **-g** параметр **<GID>** должен быть меньше 500.

Для изменения параметров группы используется команда **groupmod**. В ОС Linux также включена команда **gpasswd**, позволяющая настраивать административные учетные записи пользователей групп, членов групп и пароля групп. Члены группы могут изменить название своей группы в текущем сеансе пользователя при помощи команды **newgrp**. Если указан пароль группы, то для смены группы система попросит ввести пароль группы. Для удаления групп используется команда **groupdel**, которая удаляет группу и, если в данной группе содержались какие-то пользователи, исключает их из данной группы.

Модуль 10. Разграничение прав доступа к данным

Изучив данный модуль, вы научитесь:

- описывать модель доступа к данным в ОС Linux;
- выполнять операции по изменению прав доступа на объекты файловой системы.

10.1. Модель доступа к данным ОС Linux

Модель доступа к данным в ОС Linux достаточно проста[1]. Для разграничения доступа используются три субъекта доступа: владелец (тот, кто создал объект на файловой системе), группа и все остальные пользователи.

В ОС Linux все объекты файловой системы (файлы и каталоги) имеют разрешения. Самый простой способ узнать текущие разрешения на файл – это воспользоваться командой вывода содержимого каталога – **ls**. В следующем листинге приведен полноформатный вывод команды **ls**. Свойства файлов, которые отображает команда **ls**, приведены на рис. 10.1.

```
# ls -l /var/
total[2] 156
drwxr-xr-x  2 root root 4096 Nov 18 19:35 account
drwxr-xr-x 11 root root 4096 Nov 18 19:44 cache
```

Файл на рис. 10.1 имеет двух владельцев: пользователя (root) и группу пользователей (root). Для этого файла также заданы индивидуальные права доступа к нему, которые разбиты на три группы:

- доступ для владельца файла (**owner**);
- доступ для группы – владельца файла (**group**);
- доступ для остальных пользователей (**others**).

Для каждой категории устанавливаются три вида доступа: (**x**) – право на запуск файла, (**r**) – право на чтение файла, (**w**) – право на изменение (редактирование) файла.

---

[1] На самом деле существует расширенная модель доступа на основе расширенных списков доступа (**ACL**s), которая позволяет разделять права доступа к объекту между конкретными пользователями.

[2] Значение **total** обозначает общий размер всех файлов и каталогов в данном каталоге в блоках файловой системы.

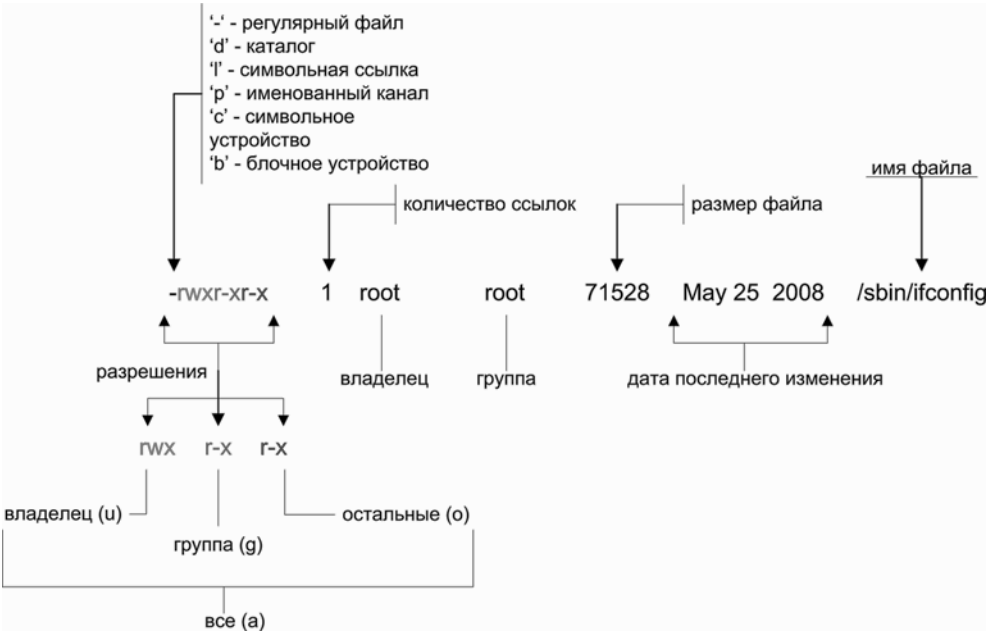


Рис. 10.1. Общие свойства файла

10.2. Изменение прав доступа к данным

В ОС Linux каждому файлу соответствуют девять битов разрешений. Данные биты формируют режим доступа к файлу и определяют, какие пользователи и группы имеют права на данный файл. Первые три бита определяют права для владельца файла; вторые три – определяют права для группы, которой принадлежит данный файл; последние три бита определяют права для всех остальных пользователей. Однако существуют еще четыре дополнительных независимых бита, которые автоматически устанавливаются при создании файла и не подлежат изменению.

Изменить режим доступа может только владелец файла или суперпользователь. Код доступа к файлу задается одним из двух режимов:

- восьмеричный режим (последовательность из трех цифр, представленная во второй колонке табл. 10.1);
- символьный режим (комбинация символов, представленных в первой колонке табл. 10.1).

В восьмеричном режиме используется последовательность из трех цифр. Итоговое разрешение на файл вычисляется путем сложения следующих восьмеричных значений:

- 4 (чтение);
- 2 (запись);
- 1 (выполнение).

Примечание

Этой последовательности может предшествовать четвертая цифра, задающая следующие режимы:

- 4 (установить **SETUID**[1] бит);
- 2 (установить **SETGID**[2] бит);
- 1 (установить **sticky**[3] бит).

Таблица 10.1. Типовые коды доступа к файлам ОС Linux

Код доступа (символьный режим)	Код доступа (восьмеричный режим)	Описание
rwx rwx rwx	777	Операции чтения, выполнения, записи доступны всем пользователям
rwx r-x r-x	755	Операции чтения и выполнения доступны всем пользователям; владельцу файла доступна операция записи
rwx r-x ---	750	Полные права доступны только владельцу, права на чтение и выполнение доступны группе; все остальные пользователи не имеют доступа к файлу
rwx --- ---	700	Доступ к файлу имеет только владелец
rw- r-- r--	644	Операции чтения и записи доступны владельцу; группе и остальным пользователям доступна только операция чтения

В символьном режиме права задаются по средствам комбинации символов ***who*** (кто), ***opcode*** (код операции) и ***permission*** (разрешение). Символ **who** является необязательным – если он не задан, то используется символ **a** (все). Символ **opcode** задается только один. В качестве файла можно указать цепочку файлов или каталогов. Доступные комбинации символов представлены в табл. 10.2.

Таблица 10.2. Коды доступа, используемые в символьном режиме

Символ who (кто)	Описание	Символ opcode	Описание	Символы разрешений	Описание
u	Владелец	+	Добавить разрешение	r	Чтение
g	Группа	–	Удалить разрешение	w	Запись

[1] Используется для запуска файлов с привилегиями пользователя **root**.

[2] Используется для временной смены группы пользователя на группу владельца файла (если установлен для исполняемых файлов) или для наследования атрибута группы всеми файлами и каталогами, созданными в каталоге с установленным битом SGID.

[3] Запрещает удалять файлы или каталоги, владельцем которых не является данный пользователь. В основном используется в общих каталогах для предотвращения удаления файлов другими пользователями.

Таблица 10.2. Коды доступа, используемые в символьном режиме (окончание)

Символ who (кто)	Описание	Символ opcode	Описание	Символы разрешений	Описание
o	Прочие	=	Назначить разрешение	x	Выполнение
a	Все			s	Назначение кода доступа пользователя или группы
				t	Установка бита с двойным значением (sticky бит)
				u	Текущее разрешение пользователя
				g	Текущее разрешение группы
				o	Текущее разрешение для всех

Для изменения прав доступа к файлу или каталогу в ОС Linux используется команда **chmod**. Ее синтаксис имеет следующий вид:

```
chmod [опции] [режим] файлы
```

Для изменения владельца и группы, которым принадлежит файл, в ОС Linux используются команды **chown** и **chgrp**.

Команда **chown** изменяет владельца файла и/или группу владельцев файла и имеет следующий синтаксис:

```
chown [опции] новый_владелец [:новая_группа] файлы
```

В качестве параметра **новый_владелец** допускается указывать либо код пользователя (user ID, UID), либо имя пользователя, заданное в файле **/etc/passwd**. Необязательный параметр **новая_группа** задается либо кодом группы (group ID, GID), либо именем группы, указанной в файле **/etc/group**. Если задан параметр **новая_группа**, действие данной команды заключается в смене владельца одного или нескольких файлов на **нового_владельца** и в обеспечении принадлежности данного файла к **новой_группе**. Выполнение команды **chmod** доступно только суперпользователю. В следующем листинге представлен пример использования команды **chown**.

```
# mkdir /var/www/html/site
# ls -ld /var/www/html/site/
drwxr-xr-x  2 root root 4096 Dec  7 20:55 /var/www/html/site/
# chown root:apache /var/www/html/site/
# ls -ld /var/www/html/site/
drwxr-xr-x  2 root apache 4096 Dec  7 20:55 /var/www/html/site/
```

После создания каталога видно, что его владельцем является суперпользователь **root**, а группой является группа **root**. В четвертой строке листинга осуществляется смена группы при помощи команды **chown**. Из последней строки кода видно, что каталог **/var/www/html/site** присвоен группе **apache**, то есть пользователи,

входящие в группу **apache**, теперь имеют права на чтение и поиск файлов в данном каталоге.

Команда **chgrp** используется для изменения группы одного или нескольких файлов и имеет следующий синтаксис:

```
chgrp [опции] новая_группа файлы
```

В качестве аргумента новая_группа может быть задан либо код группы, либо ее имя, присутствующее в каталоге **/etc/group**. Выполнение данной команды возможно только в том случае, если пользователь является владельцем файла или суперпользователем. В следующем листинге представлен пример использования команды **chgrp**.

```
# chgrp root /var/www/html/site/
# ls -ld /var/www/html/site/
drwxr-xr-x  2 root root 4096 Dec  7 20:55 /var/www/html/site/
```

10.3. Расширенные списки доступа к данным

В самом простом случае в ОС Linux права распределяются на уровне владельца, группы и всех остальных пользователей. Часто при создании гибкой системы доступа эти ограничения становятся не приемлемыми ввиду необходимости организации наследования разрешений, а также установки прав доступа на один объект для нескольких разных пользователей. Для того чтобы это стало возможным, используются расширенные списки доступа ***Extended ACLs (access control lists)***. Для использования данных списков необходимо смонтировать файловую систему с параметром **acl**:

```
# mount -o remount,acl /
```

а затем в файле **/etc/fstab** добавить опцию **acl** в опции монтирования соответствующей файловой системы.

Просмотр и изменение расширенных списков доступа выполняется командами **getfacl** и **setfacl** соответственно. Команда **setfacl** имеет следующий синтаксис:

```
setfacl [опции] операция запись:имя_записи:разрешения файл
```

Наиболее часто используемыми при работе с командой **setfacl** являются следующие опции:

- **d** – определяет разрешения по умолчанию для подкаталогов и файлов каталога, для которого установлен;
- **k** – отменяет разрешения по умолчанию;
- **R** – применяет параметры разрешений рекурсивно.

Операция определяет действие над списком и имеет следующие значения:

- **--set** – установить новые разрешения;
- **-m** – изменить текущие разрешения;
- **-x** – удалить текущие разрешения.

Запись и *имя_записи* определяют субъект, для которого устанавливаются разрешения. Записи бывают двух типов – пользователи (**u**) и группы (**g**). Имя_записи соответственно определяет имя пользователя или группы в системе.

В качестве разрешений используются стандартные формы разрешений в символьной нотации, описанные в предыдущем разделе.

В следующем листинге устанавливаются права для группы **account** на каталог **/home/accounts**:

```
# setfacl -m g:account:rwx /home/accounts/
# ls -ld /home/accounts/
drwxrwxr-x+ 2 root root 4096 Nov 18 01:36 /home/accounts/
```

Из листинга видно, что в выводе команды **ls** присутсвуте символ «+» после кода доступа, что свидетельствет об использовании расширенных списков доступа. Для просмотра установленных разрешений используется команда **getfacl**:

```
# getfacl /home/accounts/
getfacl: Removing leading '/' from absolute path names
# file: home/accounts
# owner: root
# group: root
user::rwx
group::r-x
group:account:rwx
mask::rwx
other::r-x
```

Как видно, помимо стандартынх разрешений, присутствует запись о группе **account**, устанавливающая права **rwx** на данный каталог. В следующем листинге устанавливаются разрешения для групп **account** и **sales** по умолчанию и выполняется просмотр сделанных настроек:

```
# setfacl -d -m g:account:rwx,g:sales:rx /home/accounts
# getfacl /home/accounts/
getfacl: Removing leading '/' from absolute path names
# file: home/accounts
# owner: root
# group: root
user::rwx
group::r-x
group:account:rwx
mask::rwx
other::r-x
default:user::rwx
default:group::r-x
default:group:account:rwx
default:group:sales:r-x
default:mask::rwx
default:other::r-x
```

Из листинга видно, что разрешения по умолчанию отмечены префиксом **default**.

Если использование расширенных списков доступа не планируется или не настроено, то права на файлы и каталоги по умолчанию определяются настройка-

ми командного интерпретатора (команда **umask**) в процессе входа пользователя в систему. Само по себе значение **umask** представляет собой число бит, которое надо вычесть из максимально доступных разрешений на файл (666) или каталог (777). Последовательность бит в параметре **umask** определяет параметры для пользователя, группы и всех остальных пользователей. Например, значение **umask**, равное 022, по умолчанию устанавливает код доступа 644 для всех вновь созданных файлов и 755 для всех вновь созданных каталогов.

Модуль 11. Знакомство с процессами

Изучив данный модуль, вы научитесь:

- описывать типы процессов;
- описывать принципы взаимодействия процессов в ОС Linux;
- просматривать работающие процессы и их характеристики;
- управлять процессами.

Чтобы эффективно управлять процессами и сервисами ОС Linux, необходимо хорошо понимать, что представляет собой процесс с точки зрения ОС и каким образом в ОС происходит взаимодействие между различными типами процессов. В данном разделе будут даны основные характеристики процессов ОС Linux и описаны принципы их взаимодействия.

11.1. Понятие процесса

Процесс – это объект ОС Linux, который состоит из ***адресного пространства памяти*** и набора структур данных. По сути, процесс – это запущенная программа.

Адресное пространство процесса представляет собой совокупность страниц памяти, которые были выделены ядром для выполнения данного процесса. При запуске процесса его код и необходимые библиотеки сначала загружаются в память сервера. Страницы адресного пространства процесса в конкретный момент могут находиться как в физической оперативной памяти, так и на жестком диске.

К основным параметрам процесса относятся:

- таблица распределения памяти;
- текущий статус процесса (активен, приостановлен, выполняется);
- приоритет процесса;
- информация об используемых ресурсах;
- информация о файлах и сетевых портах, открытых процессом;
- список блокируемых сигналов;
- имя владельца процесса.

Количество процессов может варьироваться от нескольких единиц до нескольких тысяч, запущенных одновременно. Центральный процессор в каждый момент выполняет только один процесс. Все остальные запущенные процессы ожидают своего выполнения в фоновом режиме.

ОС Linux использует для переключения процессов между центральным процессором специальный ***планировщик задач*** (scheduler). Данный механизм переключения процессов получил название ***многозадачности***, поскольку из-за большой скорости переключения процессов у пользователя создается ощущение, будто

процессор выполняет несколько процессов одновременно, хотя на самом деле это не так. Реальную многозадачность можно получить только при использовании нескольких процессоров или ядер, так чтобы каждый процессор или ядро выполнял по одному процессу в единичный момент.

11.2. Типы процессов

Не все процессы в ОС Linux одинаковы. Одни из них используются при выполнении пользователем какой-то команды, например команды в интерпретаторе bash или графической оболочке. Данные процессы получили название ***пользовательские процессы***. Часть процессов являются ***системными***, или ***процессами-демонами*** (daemons). Они предоставляют пользователю всевозможные сервисы, такие как веб-сервис, FTP-сервис, файловый сервис Samba. В ОС Linux тип процесса можно определить, используя команду **ps**. В следующем листинге приведена часть вывода команды **ps**, запущенной с опциями отображения всех процессов в полном формате.

```
# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 22:44 ?        00:00:00 init [5]
root         2     1  0 22:44 ?        00:00:00 [migration/0]
root         3     1  0 22:44 ?        00:00:00 [ksoftirqd/0]
root         4     1  0 22:44 ?        00:00:00 [watchdog/0]
root      2170     1  0 22:45 ?        00:00:00 auditd
root      3069     1  0 22:51 ?        00:00:02 gnome-terminal
root      3071  3069  0 22:51 ?        00:00:00 gnome-pty-helper
root      3072  3069  0 22:51 pts/1    00:00:00 bash
root      3097  2476  0 22:55 ?        00:00:00 sshd: root@pts/2
root      3099  3097  0 22:55 pts/2    00:00:00 -bash
root      3130  3099  0 22:57 pts/2    00:00:00 ps -ef
```

Можно заметить, что системные процессы в основном имеют на конце символ «**d**», например **sshd**, **klogd**. Имена пользовательских процессов, как правило, совпадают с названиями команд, запускающих данный процесс, например **ps**, **bash**.

11.3. Взаимодействие процессов

Каждый запущенный процесс в ОС Linux может породить дополнительные процессы. Процесс, запустивший новый процесс, называется ***родительским*** процессом. Новый процесс по отношению к создавшему его процессу называется ***дочерним***. Каждый процесс в ОС Linux характеризуется набором атрибутов, отличающим данный процесс от всех остальных процессов. Рассмотрим важнейшие атрибуты.

1. ***Идентификатор процесса*** (PID). Каждый процесс в системе имеет уникальный идентификатор. Каждый новый запущенный процесс получает номер на единицу больше предыдущего.
2. ***Идентификатор родительского процесса*** (PPID). Данный атрибут процесс получает во время своего запуска и используется для получения статуса родительского процесса. В листинге 2.10 видно, что процесс **portmap**

имеет идентификатор родительского процесса, равный **1**, то есть данный процесс был запущен процессом **init** – главным процессом ОС Linux, который запускает все основные системные процессы.

3. ***Реальный*** и ***эффективный идентификаторы пользователя*** (UID, EUID) и ***группы*** (GID, EGID), от которого был запущен процесс. Данные атрибуты процесса говорят о его принадлежности к конкретному пользователю и группе. Реальные идентификаторы совпадают с идентификаторами пользователя, запустившего процесс, и группы, к которой он принадлежит. Права доступа процесса к ресурсам ОС Linux определяются не реальными, а эффективными идентификаторами. Если у исполняемого файла программы имеется специальный бит **SGID** или **SUID**, то процесс данной программы будет обладать правами доступа владельца исполняемого файла. Для просмотра данных атрибутов можно воспользоваться командой **ps**, задав желаемый формат отображения колонок так, как это сделано в следующем листинге.

```
# ps -eo pid,ppid,uid,euid,gid,egid,comm
PID  PPID   UID  EUID   GID  EGID COMMAND
    1     0     0     0     0     0 init
    2     1     0     0     0     0 migration/0
    3     1     0     0     0     0 ksoftirqd/0
 2508     1     0     0    51    51 sendmail
 2516     1    51    51    51    51 sendmail
 2528     1     0     0     0     0 gpm
 2539     1     0     0     0     0 crond
 2576     1    43    43    43    43 xfs
 2587     1     0     0     0     0 anacron
 3071  3069     0     0    22    22 gnome-pty-helpe
 3072  3069     0     0     0     0 bash
 3180  2476     0     0     0     0 sshd
 3192  3180     0     0     0     0 bash
 3255  3192     0     0     0     0 ps
```

4. ***Приоритет*** (priority) и ***относительный*** (nice) ***приоритет процесса***. Данный атрибут определяет количество процессорного времени, которое может использовать процесс в работе, и зависит от таких параметров, как относительный приоритет процесса, текущее состояние, время ожидания запуска. Пользователь имеет возможность изменять только относительный приоритет процесса. В зависимости от поставленной задачи можно повышать или понижать приоритет одного процесса по отношению к другим. В ОС Linux существуют две команды управления приоритетом процессов: **nice** и **renice**.

   Относительный приоритет процесса – это переменный параметр, значение которого изменяется в диапазоне от –20 до +19. Чем больше значение данного параметра, тем меньше относительный приоритет процесса. С наименьшими относительными приоритетами работают системные процессы. Относительный приоритет дочернего процесса наследуется от родительского процесса, поэтому все процессы, запущенные командой **init**, работают с тем же приоритетом.

5. ***Состояние процесса***. В ОС Linux каждый процесс обязательно находится в одном из указанных ниже состояний и может быть переведен из одного состояния в другое системой или командами пользователя. Различают следующие состояния процессов.
   - ***Работоспособный процесс*** (runnable). Если процесс в текущий момент выполняет какие-либо действия или находится в очереди у центрального процессора на исполнение, он называется работоспособным и обозначается символом **R**.
   - ***Ожидающий процесс*** (sleeping). Данное состояние означает, что процесс инициализировал выполнение какой-либо системной операции и ожидает ее завершения. К таким операциям относятся ввод/вывод, завершение дочернего процесса и т. д. Процессы с таким состоянием обозначаются символом **S**.
   - ***Остановленный процесс*** (stopping). Любой процесс можно остановить. Это может делать как система, так и пользователь. Состояние такого процесса обозначается символом **T**.
   - ***Завершившийся процесс*** (zombie). Процессы данного состояния возникают в случае, когда родительский процесс, не ожидая завершения дочернего процесса, продолжает параллельно работать. Процессы с таким состоянием обозначаются символом **Z**. Завершившиеся процессы больше не выполняются системой, но по-прежнему продолжают потреблять ее вычислительные ресурсы.
   - ***Непрерываемый процесс*** (uninterruptible). Процессы в данном состоянии ожидают завершения операции ввода/вывода с прямым доступом в память. Такой процесс нельзя завершить, пока не завершится операция ввода/вывода. Процессы с таким состоянием обозначаются символом **D**.

Команда **nice** используется для запуска процессов с заданным приоритетом и имеет следующий синтаксис:

```
nice [-n] [команда [аргументы]...]
```

Здесь после необязательной опции **-n** необходимо указать значение относительного приоритета, а в качестве параметров «**команда**» и «**аргументы**» указать соответствующую команду и ее аргументы.

Например, для запуска нового процесса редактора **vi** со значением **-8** необходимо выполнить команду **nice -n -8 vi** . После ввода данной команды откроется окно редактора **vi**. Убедиться в изменении относительного приоритета процесса можно, воспользовавшись командой **ps**.

Команда **renice** используется для изменения приоритета уже запущенных процессов и имеет следующий синтаксис:

```
renice приоритет [ [ -p ] идентификаторы процессов ] [ [ -g ] группы ] [ [ -u ]
пользователи ]
```

Команде **renice** в качестве обязательных необходимо указать два аргумента: относительный приоритет процесса и один из трех дополнительных аргументов:

идентификатор процессов, имя группы, которой принадлежат процессы, имя пользователя, запустившего данный процесс.

Например, для изменения текущего относительного приоритета процесса **Xorg** необходимо выполнить команду **renice** следующим образом:

```
# renice -5 -p $(pgrep Xorg)
2852: old priority 0, new priority -5
# ps -eo nice,comm|grep Xorg
 -5 Xorg
```

Здесь в качестве параметра для опции **-p** используется подстановка команды **pgrep**, применяемой для определения идентификатора процесса по его имени. В последней строке кода данного листинга происходит проверка того, изменился ли относительный приоритет процесса **Xorg**.

11.4. Управление процессами

Под управлением процессами в ОС Linux подразумеваются следующие операции:

- запуск процессов;
- просмотр запущенных процессов;
- управление режимом работы процесса;
- завершение работы процесса подачей определенного сигнала.

11.4.1. Запуск процессов

Запуск процессов осуществляется довольно просто путем ввода произвольной команды или запуска произвольного скрипта. В следующем примере выполняется запуск процесса **xclock** в фоновом режиме:

```
# xclock &
[1]12234
```

В следующем примере выполняется запуск скрипта **backup.sh**, находящегося в рабочем каталоге:

```
# ./backup.sh
```

11.4.2. Просмотр запущенных процессов

Для просмотра запущенных процессов в ОС Linux используются утилиты **top** и **ps**, причем команда **top** отображает статус процессов в реальном времени.

Стандартный вывод команды top представлен в следующем листинге:

```
# top
top - 00:22:15 up  1:38,  2 users,  load average: 0.00, 0.01, 0.10
Tasks: 106 total,   1 running, 105 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3%us,  2.0%sy,  0.0%ni, 97.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    385484k total,   380060k used,     5424k free,    15484k buffers
Swap:  1052248k total,        4k used,  1052244k free,   252724k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2852 root      15   0  146m 9308 5140 S  0.7  2.4   0:33.02 Xorg
```

```
 2528 root       15   0  1872  472  392 S  0.3  0.1   0:00.83 gpm
 2616 root       15   0 24848  11m 3260 S  0.3  3.1   0:04.80 yum-updatesd
  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
  170 root       17  -5     0    0    0 S  0.0  0.0   0:00.00 cqueue/0
```

Как видно из листинга, стандартный вывод команды **top** имеет несколько полей, каждое из которых содержит соответствующий атрибут процесса. Описание полей вывода команды **top** представлено в табл. 11.1.

Таблица 11.1. Основные поля вывода команды top

Поле	Описание
PID	Идентификатор процесса
PPID	Идентификатор родительского процесса
USER	Идентификатор пользователя, запустившего процесс
GROUP	Идентификатор группы, которой принадлежит процесс
S	Состояние процесса
PR	Приоритет процесса
NI	Относительный приоритет процесса
TIME+, TIME	Количество процессорного времени, которое использует процесс с момента своего запуска
VIRT	Полный объем виртуальной памяти, которую занимает процесс
RES	Объем резидентной виртуальной памяти, которую занимает процесс
SHR	Объем общей виртуальной памяти, которую использует процесс
SWAP	Объем виртуальной памяти процесса, выгруженной на диск
%CPU	Процент использования общего процессорного времени
%MEM	Процент использования доступной физической памяти
%MEM	Процент использования доступной физической памяти
CMD	Команда, использованная для создания процесса

Команда **top** имеет встроенную систему комбинаций клавиш, которая позволяет сортировать вывод по конкретным полям и управлять процессами. Для вывода справочной информации по клавишам необходимо нажать клавишу **<h>**.

Для добавления определенного поля в общий вывод команды **top** необходимо нажать клавишу **<f>** и отметить символом * поля, которые необходимо отображать.

Вывод команды **ps** схож с выводом команды **top**, однако он отображает статический снимок процессов. По умолчанию команда **ps** выводит только информацию о процессах, запущенных в текущей сессии командного интерпретатора bash. Для вывода информации по всем процессам необходимо ввести команду **ps** с ключом **-e**. Для отображения желаемых полей необходимо ввести команду **ps** с ключом **-o поле1, поле2, …** , (через запятую перечисляются поля, которые необходимо отобразить).

11.4.3. Управление режимом работы процесса

Все процессы в ОС Linux могут функционировать в одном из двух режимов: ***фоновом*** и ***активном***. В фоновом режиме оболочка, запустившая процесс, не ожидает его завершения, а продолжает выполнение других процессов или задач. Все системные процессы-демоны работают, как правило, в фоновом режиме.

В активном режиме работают процессы, запущенные непосредственно пользователем. Для запуска процесса в фоновом режиме в конце команды указывают знак **&**. Для перевода активного процесса в фоновый режим его сначала необходимо остановить (используется специальное сочетание клавиш **<Ctrl+Z>**), а потом при помощи команды **bg** перевести в фоновый режим работы. Для перевода процесса из фонового режима в активный необходимо воспользоваться командой **fg**. Количество процессов, находящихся в фоновом режиме, можно просмотреть, используя встроенную команду **jobs**. Пример перевода процесса из одного режима работы в другой представлен в следующем листинге.

```
# find / -name *.txt$ -exec rm {} \;
[1]+  Stopped                 find / -name *.txt$ -exec rm {} \;
# jobs
[1]+  Stopped                 find / -name *.txt$ -exec rm {} \;
# bg 1
[1]+ find / -name *.txt$ -exec rm {} \; &
# fg 1
find / -name *.txt$ -exec rm {} \;
```

В первой строке листинга осуществляется запуск команды **find**, которая ищет все файлы, оканчивающиеся на строку **tmp**, и удаляет их. Процесс поиска останавливается с помощью комбинации клавиш **<Ctrl+Z>**. В последних строках листинга осуществляется перевод процесса в фоновый режим, а затем в активный при помощи команд **bg** и **fg** соответственно.

11.4.4. Завершение работы процесса

В ОС Linux для завершения работы процесса или выполнения других операций над процессом используется подача специального сигнала процессу. ***Сигнал*** – это системный запрос на прерывание. Сигналы могут посылаться разными способами: ядром – в случае нарушения принятых стандартов взаимодействия, терминалом – в случае ввода специальных комбинаций клавиш и пользователем – при помощи команд **kill** или **killall** и др.

Команда **kill** используется для подачи сигнала процессу в соответствии с его идентификатором и имеет следующий синтаксис:

```
kill [ -s сигнал | -p ] [ -a ] [ – ] идентификатор процесса ...
```

`сигнал` – символическое имя или номер посылаемого сигнала;
`идентификатор процесса` – номер процесса **(PID)**, которому посылается сигнал.

В случае запуска команды **kill** без указания сигнала в качестве посылаемого будет использован сигнал TERM, который предназначен для завершения работы процесса. Следует отметить, что команда **kill** может завершать работу лишь определенных процессов, владельцем которых является пользователь, запустивший команду **kill**.

В следующем листинге приведен пример использования команды **kill** для завершения процесса демона **cupsd**. Для определения идентификатора процесса де-

мона **cupsd** можно использовать команду **pgrep**, которая возвращает идентификатор найденного процесса, или команду **ps**.

```
# kill $(pgrep cupsd)
# pgrep cupsd
```

Команда **killall** используется для подачи сигнала процессу в соответствии с его именем и имеет следующий синтаксис:

```
killall [-u пользователь] [ -сигнал ] имя процесса ...
```

`сигнал` – символическое имя или номер посылаемого сигнала;
`пользователь` – имя пользователя, который является владельцем процесса;
`имя процесса` – имя процесса, которому посылается сигнал.

При использовании команд **kill** или **killall** можно подавать сигналы процессам, обладающим определенными параметрами, например владелец.

Модуль 12. Программирование в оболочке bash

Изучив данный модуль, вы научитесь:

- понимать принцип составления сценариев;
- применять основные конструкции языка bash;
- отлаживать созданные сценарии.

Знание языка командной оболочки является залогом успешного решения задач администрирования системы. Во время загрузки ОС Linux выполняется целый ряд сценариев из каталога **/etc/rc.d**, которые настраивают конфигурацию системы и запускают различные сервисы, поэтому очень важно четко понимать эти скрипты и иметь достаточно знаний, чтобы вносить в них какие-либо изменения. Почти все системные утилиты, входящие в состав ОС Linux, имеют файлы-сценарии, предназначенные для выполнения конкретных задач. Понимание принципов программирования в оболочке позволит эффективно управлять системой в целом, а также интегрировать различные приложения для решения поставленных задач.

Язык сценариев легок в изучении, в нем не так много специфических операторов и конструкций. Синтаксис языка достаточно прост и прямолинеен, он очень напоминает команды, которые приходится вводить в командной строке. Короткие скрипты практически не нуждаются в отладке, и даже отладка комплексных скриптов отнимает весьма незначительное время. Сценарии командной оболочки очень хорошо подходят для быстрого создания прототипов сложных приложений, даже несмотря на ограниченный набор языковых конструкций и определенную «медлительность». Такая методика позволяет детально проработать структуру будущего приложения, обнаружить возможные проблемы и лишь затем приступить к разработке кода на **C**, **C++**, **Java** или **Perl**.

Неприменимость сценариев командной оболочки может быть оправдана в следующих случаях:

- для ресурсоемких задач, особенно когда важна скорость исполнения (поиск, сортировка и т. п.);
- для задач, связанных с выполнением математических вычислений, особенно это касается вычислений с плавающей запятой, вычислений с повышенной точностью, комплексных чисел;
- для кросс-платформенного программирования;
- когда необходим прямой доступ к аппаратуре компьютера.

Во всех остальных случаях применение сценариев командной оболочки может быть вполне оправдано.

12.1. Структура и выполнение сценариев

Сценарий не относится к компилируемым программам, поскольку он интерпретируется построчно. Первой строкой сценария всегда должна быть строка вида:

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

В данной строке система получает указание, где следует искать интерпретатор **BASH**. Каждый сценарий может содержать комментарии; если комментарии размещаются в сценарии, первым символом в строке комментария будет символ **#**. Встретив подобный символ, интерпретатор команд игнорирует строку комментария.

Если файл сценария начинается с последовательности **#!**, которая в мире Unix называется ***sha-bang***, то это указывает системе, какой интерпретатор следует использовать для исполнения сценария. Это двухбайтовая последовательность, определяющая тип сценария, в данном случае это сценарий командной оболочки **bash**. Более точно: **sha-bang** определяет интерпретатор, который вызывается для исполнения сценария, это может быть командная оболочка, иной интерпретатор или утилита, например **awk** или **sed**.

Сценарий просматривается интерпретатором команд в направлении сверху вниз. Перед выполнением сценария требуется воспользоваться командой **chmod**, устанавливающей право доступа на выполнение, и убедиться в том, что правильно указан путь к сценариям, тогда для его выполнения достаточно будет указать имя файла сценария. Запустить сценарий также можно, непосредственно обратившись к команде **bash**.

После того как файл сценария стал исполняемым, можно запустить его следующей командой:

```
# ./<имя_файла_сценария>
```

Если при этом текст сценария начинается с корректной сигнатуры (**sha-bang**), то для его исполнения будет вызван соответствующий интерпретатор. И наконец, завершив отладку сценария, можно поместить его в каталог **/usr/local/bin**, чтобы сделать его доступным для других пользователей системы (если это необходимо). После этого сценарий можно вызвать, просто напечатав название файла сценария в командной строке и нажав на клавишу **<Enter>**.

Ниже рассматривается пример простого сценария, который отменяет отображение сообщений **/var/adm** путем усечения файла сообщений. В задачи этого сценария также входит удаление всех журнальных файлов в каталоге **/usr/local/log**.

```
# cat cleanup
#!/bin/sh
# имя_сценария: cleanup
# этот сценарий выполняет очистку журнальных файлов
echo "starting cleanup...wait"
rm /usr/local/log/*.log
tail -10 /var/adm/messages /tmp/messages
rm /var/adm/messages
mv /tmp/messages /var/adm/messages
echo "finished cleanup"
exit 0
```

Для выполнения сценария необходимо применить команду **chmod**:

```
# chmod u+x cleanup
```

Чтобы запустить сценарий на выполнение, необходимо ввести его название:

```
# cleanup
```

При отображении сообщения об ошибке, например

```
# cleanup
sh: cleanup: command not found,
```

необходимо выполнить сценарий следующим образом:

```
# ./cleanup
```

12.2. Переменные сценария. Позиционные параметры

Переменные – это одна из основ любого языка программирования. Они участвуют в арифметических операциях, в синтаксическом анализе строк и совершенно необходимы для абстрагирования каких-либо величин с помощью символических имен. Физические переменные представляют собой не что иное, как участки памяти, в которые записана некоторая информация.

Если командный интерпретатор **bash** встречает в тексте сценария ***имя*** переменной, то он вместо него подставляет ***значение*** этой переменной. Поэтому ссылки на переменные называются ***подстановкой переменных***.

В отличие от большинства других языков программирования, **bash** не производит разделения переменных по «типам». По сути, переменные **bash** являются строковыми переменными, но в зависимости от контекста **bash** допускает целочисленную арифметику с переменными. Определяющим фактором здесь служит содержимое переменных.

В интерпретаторе **bash** существуют два типа переменных:

- локальные переменные;
- переменные окружения.

Локальные переменные – это переменные, область видимости которых ограничена блоком кода или телом функции (локальные переменные в функциях). ***Блок кода*** (фигурные скобки), известный также как «**вложенный блок**», – это конструкция, фактически создающая анонимную функцию. Однако, в отличие от обычных функций, переменные, создаваемые во вложенных блоках кода, доступны всему сценарию:

```
# { local a; a=123; }
bash: local: can only be used in a function
# a=123
# { a=321; }
echo "a = $a"   # a = 321   (значение, присвоенное во вложенном блоке кода)
a = 321
```

Код, заключенный в фигурные скобки, может выполнять перенаправление ввода-вывода, например:

```
# head -2 /etc/fstab
/dev/VolGroup00/LogVol00 /                       ext3    defaults        1 1
LABEL=/boot              /boot                   ext3    defaults        1 2
# cat <<EOF >readline.sh
>#!/bin/bash
># Чтение строк из файла /etc/fstab.
>File=/etc/fstab
>{
>read line1
>read line2
>} < $File
>echo "Первая строка в $File:"
>echo "$line1"
>echo "Вторая строка в $File:"
>echo "$line2"
>exit 0
>EOF
# chmod u+x readline.sh
# ./readline.sh
Первая строка в /etc/fstab:
/dev/VolGroup00/LogVol00 /                       ext3    defaults        1 1
Вторая строка в /etc/fstab:
LABEL=/boot              /boot                   ext3    defaults        1 2
```

В данном листинге из файла **/etc/fstab** выполняется перенаправление первых двух строк в переменные **line1** и **line2**.

Переменные, которые затрагивают командную оболочку и порядок взаимодействия с пользователем, называются ***переменными окружения***, они доступны всем процессам, в том числе и процессу оболочки. Каждый раз, когда запускается командный интерпретатор, для него создаются переменные, соответствующие переменным окружения. Изменение переменных или добавление новых переменных окружения заставляет оболочку обновить свои переменные, и все дочерние процессы (и команды, исполняемые ею) наследуют это окружение.

Если сценарий изменяет переменные окружения, то они должны «**экспортироваться**», то есть передаваться окружению, локальному по отношению к сценарию. Эта функция возложена на команду **export**.

Сценарий может **экспортировать** переменные только дочернему процессу, то есть командам и процессам, запускаемым из данного сценария. Сценарий, запускаемый из командной строки, **не может** экспортировать переменные «наверх» командной оболочке. Дочерний процесс **не может** экспортировать переменные родительскому процессу.

От того, как будет определена переменная в сценарии, зависит все. Переменная может быть просто определена с использованием знака присвоения «**=**». Однако часто значение переменной невозможно предугадать в силу различных условий, которым должна удовлетворять переменная. Поэтому в командной оболочке существует много способов подстановки значения переменной. Вот некоторые из них:

- **${переменная}** – используется, если сразу же за переменной следует произвольный текст;
- **${переменная:-значение}** – использовать переменную, если она установлена; в противном случае использовать указанное значение;

- **${переменная:=значение}** – использовать переменную, если она установлена; в противном случае использовать указанное значение и присвоить переменной заданное значение;
- **${переменная:?значение}** – использовать переменную, если она установлена; в противном случае вывести значение и выполнить выход. Если значение не задано, вывести сообщение об ошибке;
- **${переменная:+значение}** – использовать заданное значение, если переменная установлена; в противном случае ничего не использовать;
- **${#@}** – использовать количество ***позиционных параметров***.

Позиционные параметры – это аргументы, передаваемые сценарию из командной строки, – $0, $1, $2, $3..., где $0 – это **название файла сценария**, $1 – первый аргумент, $2 – второй, $3 –третий и т. д. Аргументы, следующие за $9, должны заключаться в фигурные скобки, например ${10}, ${11}, ${23}. Специальные переменные **$*** и **$@** содержат **все** позиционные параметры (аргументы командной строки).

Скобочная нотация позиционных параметров дает довольно простой способ обращения к **последнему** аргументу, переданному в сценарий из командной строки. Такой способ подразумевает использование косвенной адресации:

```
args=$#           # Количество передаваемых аргументов.
lastarg=${!args}  # Обратите внимание: lastarg=${!$#} неприменимо.
```

В сценарии можно предусмотреть различные варианты развития событий в зависимости от имени сценария. Для этого сценарий должен проанализировать аргумент **$0** – имя файла сценария. Это могут быть и имена символьных ссылок на файл сценария.

Если сценарий ожидает передачи аргументов в командной строке, то при их отсутствии он получит «пустые» переменные, что может вызвать нежелательный побочный эффект. Один из способов обхода подобных ошибок – добавить дополнительный символ в обеих частях операции присваивания, где используются аргументы командной строки:

```
variable1_=$1_
```

12.3. Коды завершения сценария

Для того чтобы отследить поведение сценария, отладить его алгоритм работы, полезно использовать ***коды завершения***. Каждая команда возвращает ***код завершения*** (иногда код завершения называют ***возвращаемым значением***). В случае успеха команда должна возвращать 0, а в случае ошибки – ненулевое значение, которое, как правило, интерпретируется как код ошибки. Практически все команды и утилиты ОС Linux возвращают 0 в случае успешного завершения, но имеются и исключения из правил.

Код завершения команды является целым числом в диапазоне 0–255, значение которого содержится в переменной **$?** после выполнения **одной** произвольной команды. Аналогичным образом ведут себя функции, расположенные внутри сценария, и сам сценарий, возвращая код завершения. Код, возвращаемый функцией или сценарием, определяется кодом возврата **последней команды**.

Для определения кода завершения целого сценария (не одной команды) используется команда **exit**. Кроме того, данная команда может возвращать некоторое значение, которое может быть проанализировано вызывающим процессом.

Команде **exit** можно явно указать код возврата в виде: **exit nnn**, где **nnn** – это код возврата. Когда работа сценария завершается командой **exit** без параметров, то код возврата сценария определяется кодом завершения **последней** исполненной команды (не считая саму команду **exit**).

Код возврата последней команды хранится в специальной переменной **$?**. После исполнения кода функции переменная **$?** хранит код завершения последней команды, исполненной в функции. Таким способом в **bash** передается «значение, возвращаемое» функцией. После завершения работы сценария код возврата можно получить, обратившись из командной строки к переменной **$?**, то есть это будет код возврата последней команды, исполненной в сценарии.

В следующем сценарии приведены примеры определения кодов возврата команд и самого сценария:

```
# cat exit.sh
#!/bin/bash
echo hello
echo $?    # код возврата = 0, поскольку команда выполнилась успешно.
lskdf      # Несуществующая команда.
echo $?    # Ненулевой код возврата, поскольку команду выполнить не удалось.
exit 113   # Явное указание кода возврата 113.
# ./exit.sh
hello
0
./exit.sh: line 4: lskdf: command not found
127
# echo $?
113
```

12.4. Проверка условий. Логические и условные операторы

При создании сценария уточняется идентичность строк, права доступа к файлу или же выполняется проверка численных значений. На основе результатов проверки предпринимаются дальнейшие действия. Проверка обычно осуществляется с помощью команды **test**. Команда **test** используется для тестирования строк, проверки прав доступа к файлу и численных данных. В командной оболочке **bash**, помимо команды **test**, существуют несколько дополнительных конструкций проверки условий:

- Оператор **if/then/else** проверяет, является ли код завершения списка команд 0 (поскольку 0 означает «успех»), и если это так, то выполняет одну или более команд, следующих за словом **then**:

```
# a=2
# if [ $a -eq 2 ]; then echo yes; fi
yes
```

- Команда – **[** (левая квадратная скобка). Она является синонимом команды **test** и встроенной командой (то есть более эффективной в смысле производительности). Эта команда воспринимает свои аргументы как выражение сравнения или как файловую проверку и возвращает код завершения в соответствии с результатами проверки (0 – «истина», 1 – «ложь»):

```
# [ -f /var/log/messagess ]
# echo $?
1
# [ -f /var/log/messages ]
# echo $?
0
```

  Здесь ключ **–f** выполняет проверку существования файла. Существуют следующие основные проверки файлов:

  -d Каталог
  -f Обычный файл
  -L Символьная связь
  -r Файл для чтения
  -s Файл имеет ненулевой размер, он не пуст
  -w Файл для записей
  -u Файл имеет установленный бит suid
  -x Исполняемый файл

- Конструкция [[...]] – расширенный вариант команды **test**, которая выполняет сравнение с использованием логических операторов. **Bash** исполняет команду **[[$a -lt $b]]** как один элемент, который имеет свой код возврата.
- Конструкция **((...))** и предложение **let ...** также возвращают код 0, если результатом арифметического выражения является ненулевое значение. В данной конструкции выполняется арифметическое вычисление. Таким образом, арифметические выражения могут участвовать в операциях сравнения.
  Предложение **let «1<2»** возвращает 0 (так как результат сравнения «1<2» равен «1», или «истина»);
  Предложение **((0 && 1))** возвращает 1 (так как результат операции «0 && 1» равен «0», или «ложь»).
- Условный оператор **if** проверяет код завершения любой команды, а не только результат выражения, заключенного в квадратные скобки:

```
if grep -q Bash file
then echo "Файл содержит, как минимум, одно слово Bash."
fi
```

  Когда **if** и **then** располагаются в одной строке, то конструкция **if** должна завершаться точкой с запятой.

Во всех конструкциях сравнения используются операции сравнения чисел или строк. Для сравнения чисел используются следующие конструкции:

- **-eq** (равно);
- **-ne** (не равно);
- **-gt** (больше);

- **-ge** (больше или равно);
- **-lt** (меньше);
- **-le** (меньше или равно).

Для сравнения строк используются следующие конструкции:

- `строка` – строка содержит текст;
- **-n** `строка1` – строка имеет ненулевую длину;
- **-z** `строка1` – строка имеет нулевую длину;
- `строка1` **==** `строка2` – строка1 и строка2 равны. Строка2 может быть подстановкой;
- `строка1` **!=** `строка2` – строка1 и строка2 не равны;
- `строка1` **=~** `строка2` – строка1 соответствует расширенному регулярному выражению строка2.

В следующем листинге приведены примеры сравнения строк и чисел в сценарии:

```
#!/bin/bash
a=4
b=5
if [ "$a" -ne "$b" ]
then
  echo "$a не равно $b"
  echo "(целочисленное сравнение)"
fi
if [ "$a" != "$b" ]
then
  echo "$a не равно $b."
  echo "(сравнение строк)"
fi
if [ -n $a ]    # $a не была объявлена или инициализирована.
then
  echo "Строка \"a\" не пустая."
else
  echo "Строка \"a\" пустая."
fi
```

Иногда для выполнения сравнений приходится использовать комбинированные условия, которые позволяют объединять единичные проверки в единую проверку. Чтобы реализовать подобную проверку, интерпретатор **bash** предлагает следующие конструкции логических операторов:

- (`условие`) – является истинным, если значение условия истинно;
- **!** `условие` – является истинным, если условие ложно;
- `условие1` **–a** `условие2` – является истинным, если оба условия истинны;
- `условие1` **–o** `условие2` – является истинным, если любое из условий истинно;
- `условие1` **&&** `условие2` – является истинным, если оба условия истинны (допускается использовать только в конструкции [[...]]).
- `условие1` **||** `условие2` – является истинным, если любое из условий истинно (допускается использовать только в конструкции [[...]]).

Bash исполняет конструкцию **[[$a -lt $b && $b –eq $c]]** как один элемент, который имеет единый код возврата.

12.5. Управляющие конструкции FOR, WHILE, UNTIL, CASE

Почта все сценарии, за редким исключением, обладают свойством самоуправляемости. В чем состоит управление ходом выполнения сценария? Предположим, что в состав сценария включены несколько команд:

```
#!/bin/bash
# создание каталога
mkdir /home/dave/mydocs
# копирование всех файлов с расширением doc
cp *.doc /home/dave/docs
# удаление всех файлов с расширением doc
rm *.doc
```

Рассматриваемый сценарий выполняет определенные задачи. Каковы же могут быть причины возможных неприятностей? Проблема возникнет, например, в том случае, если нельзя будет создать данный каталог. Как поступить в иной ситуации, если каталог может быть создан, но при копировании файлов появляется сообщение об ошибке? Что произойдет, если применить команду **cp** к разным файлам из различных каталогов? Продуманные решения нужно принимать до применения команды, а еще лучше, если они реализуются при получении результатов выполнения последней команды. Интерпретатор **Bash** поддерживает наборы командных операторов, которые помогают принять верное решение в зависимости от успеха или неудачи при выполнении команды либо при обработке списка.

Существуют два вида таких командных операторов:

- операторы цикла (FOR, WHILE, UNTIL);
- операторы, изменяющие ход выполнения сценария (IF,THEN,ELSE, CASE, SELECT).

Операторы **IF, THEN, ELSE** позволяют реализовать условное тестирование. Проверить условия можно самыми различными способами. Например, могут производиться оценка размера файла, проверка установленных прав доступа к файлу, сравнение каких-либо числовых значений или строк. В результате выполнения сравнений возвращается значение «истина» (0) либо «ложь» (1), и затем предпринимаются действия на основании полученного результата. Перед тем как мы приступим к обсуждению условного тестирования, стоит отметить, что некоторые понятия из этой области были рассмотрены ранее.

Операторы **CASE и SELECT** дают возможность выполнять поиск по шаблонам, словам или числовым значениям. После нахождения нужного шаблона или слова можно переходить к выполнению других операторов, которые основываются исключительно на условии, по которому устанавливалось соответствие.

Цикл, или итерация, – это процесс повторного выполнения наборов команд. В распоряжении пользователя имеются три вида операторов цикла:

- **FOR** – последовательная обработка значений до тех пор, пока не встретится окончание списка;

- **UNTIL** – используется реже всего. Оператор определяет непрерывное выполнение цикла, пока условие не станет истинным. Проверка условия выполняется в конце цикла;
- **WHILE** – задает выполнение цикла до тех пор, пока не будет встречено заданное условие. Проверка условия выполняется в начале цикла.

Каждый цикл, содержащий операторы управления ходом выполнения сценария, может быть вложен. Например, внутри цикла **FOR** может размещаться другой цикл **FOR**.

Цикл **FOR** имеет следующую конструкцию:

```
for ((начальное_значение; условие; приращение))
do
    команды
done
```

В следующем листинге приведен пример цикла **FOR**:

```
for ((x=1; x <=20; x +=2))
do
    grep $1 chap$x
done
```

В данном листинге выполняется поиск фразы в каждой нечетной главе. Главы представляют собой файлы, в имени которых последние два символа обозначают номер главы. В качестве фразы используется первый аргумент командной строки (**$1**). Поиск выполняется утилитой **grep** до тех пор, пока последние два символа файла не станут равными **20**, то есть пока не будет обработан файл **chap20**.

Цикл **WHILE** имеет следующую конструкцию:

```
while условие
do
    команды
done
```

Пока удовлетворяется условие, выполнять команды. В следующем сценарии приведен пример цикла **WHILE**:

```
#!/bin/bash
LIMIT=10
a=1
while [ "$a" -le $LIMIT ]
do
  echo -n "$a "
  let "a+=1"
done
exit 0
```

В данном примере выполняется построчный ввод чисел от 1 до 10. В квадратных скобках записано условие проверки обрабатываемого числа. В команде **let** выполняется прибавление 1 к текущему обрабатываемому числу.

Цикл **UNTIL** имеет следующую конструкцию:

```
until условие
do
    команды
done
```

В следующем сценарии приведен пример использования цикла **UNTIL**:

```
#!/bin/bash
until [ "$var1" = end ]
do
  echo "Введите значение переменной #1 "
  echo "(end - выход)"
  read var1
  echo "значение переменной #1 = $var1"
done
exit 0
```

Здесь в начале каждой итерации выполняется проверка переменной, и в случае если значение переменной совпадает со строкой **end**, выполняется немедленный выход из цикла.

Оператор **CASE** имеет следующую структуру:

```
case значение in
образец1) команды1;;
образец2) команды2;;
...
образецN) командыN;;
esac
```

В операторе **CASE** выполняется определенный набор команд, в зависимости от того, чему равно значение в операторе. Если значение соответствует 1-му образцу, то выполняется первый набор команд и т. д. В следующем листинге приведен пример использования оператора **CASE** для проверки введенной клавиши:

```
#!/bin/bash
echo; echo "Нажмите любую клавишу и затем клавишу Return."
read Keypress
case "$Keypress" in
  [a-z]   ) echo "буква в нижнем регистре";;
  [A-Z]   ) echo "Буква в верхнем регистре";;
  [0-9]   ) echo "Цифра";;
  *       ) echo "Знак пунктуации, пробел или что-то другое";;
esac
exit 0
```

Оператор **SELECT** имеет следующую структуру:

```
case значение [in список]
do
команды
done
```

Этот оператор предлагает пользователю выбрать один из представленных вариантов. Примечательно, что **select** по умолчанию использует в качестве приглашения к вводу – **PS3 (#?)**, который легко изменить, задав соответствующее значение в приглашении **PS3**. Обычно оператор select используется для создания текстовых меню. В следующем листинге приведен пример сценария с данным оператором:

```
#!/bin/bash
PS3='Выберите номер элемента: '
```

```
select event in Format Page View Exit
do
            case "$event" in
            Format  ) nroff $file | lp;;
            Page            ) pr $file | lp;;
            View            ) less $file;;
            *               ) echo «Недопустимый элемент меню»;;
            esac
done
exit 0
```

12.6. Использование позиционных параметров. Команды SHIFT и GETOPTS

Ранее рассматривались способы передачи параметров сценариям с помощью специальных переменных **$1…$9**. Специальная переменная **$#** указывает количество передаваемых сценарию параметров. В данном разделе будут рассмотрены следующие темы:

- применение команды **shift**;
- работа с командой **getopts**.

Если использовать прямой перебор всех входных параметров, то может получиться достаточно громоздкий код, большая часть которого будет выполнять проверку аргументов командной строки (например, в случае использования оператора **CASE** для этих целей).

К счастью, интерпретатор команд **Bash** поддерживает команду **shift**, с помощью которой можно выбирать различные опции. Команда **shift** позволяет устранить ограничение, состоящее в том, что при передаче параметров применяются только специальные переменные **$1…$9**.

Команда имеет следующий синтаксис:

```
shift [N]
```

Команда **shift** выполняет смещение позиционных параметров (например, **$2** становится **$1**). Если задано значение **N**, выполняется смещение на **N** позиций влево. Целесообразно использовать команду **shift** совместно с циклом **WHILE** для выполнения итераций над аргументами командной строки.

Чтобы уточнить принцип действия команды **shift**, рассмотрим простой сценарий. Здесь применяется цикл **while**, обеспечивающий отображение на экране всех аргументов, переданных сценарию.

```
#!/bin/bash
loop=0
while [ $# -ne 0 ]
do
            echo $1
done
```

Создается впечатление, что указанный сценарий будет выполняться до тех пор, пока в командной строке не останется аргументов. К сожалению, это не так.

При запуске сценария на экран выводится только первый аргумент, поскольку в сценарии не предусмотрен переход к следующему параметру. Вот результат выполнения вышеприведенного сценария:

```
$ ./scr1 file1 file2 file3
file1
file1
file1
```

Для обработки каждого передаваемого аргумента достаточно воспользоваться командой **shift**. Ниже приведен соответствующий сценарий:

```
#!/bin/bash
loop=0
while [ $# -ne 0 ]
do
            echo $1
            shift
done
```

Результат выполнения данного сценария будет следующий:

```
$ ./scr1 file1 file2 file3
file1
file2
file3
```

Применение команды **getopts** обеспечивает создание программного кода, который без труда справляется с несколькими аргументами командной строки. Благодаря использованию этой команды процесс обработки командной строки приводится в соответствие с некоторым стандартом. Ведь сценарии должны соответствовать стандартному формату файлов командных опций.

В общем случае команда **getopts** выполняет обработку аргументов командной строки и проверку на допустимые опции. Стандартный формат файловых опций указывает, что опции командной строки должны начинаться с символа «-». Может выполняться комбинирование опций, то есть можно вводить несколько букв после символа «-». Обработка опций завершается указанием символов «--» в командной строке. Команда имеет следующий синтаксис:

```
getopts OPTSTRING var
```

Параметр OPTSTRING содержит буквы опций, подлежащие распознаванию при выполнении сценария. Опции, которые требуют указания аргумента, пишутся с указанием символа «:» за ними. Допустимые опции обрабатываются по очереди и сохраняются в переменной командного интерпретатора **$var**. Аргументы опций сохраняются в переменной **$OPTARG**. Несколько аргументов передаются в команду одной строкой командного интерпретатора. Это выполняется путем заключения их в кавычки или указанием через запятую.

С командой **getopts** тесно взаимосвязаны переменные **$OPTIND** – указатель на аргумент и **$OPTARG** – аргумент опции.

Обычно **getopts** заключается в цикл **WHILE**, в каждом проходе цикла извлекаются очередная опция и ее аргумент (если он имеется), обрабатывается, затем

уменьшается на 1 переменная **$OPTIND** и выполняется переход к началу новой итерации. Есть несколько замечаний по использованию команды **getopts**:

1. Опциям (ключам), передаваемым в сценарий из командной строки, должен предшествовать символ «минус» (–) или «плюс» (+). Этот префикс (– или +) позволяет **getopts** отличать опции (ключи) от прочих аргументов. Фактически **getopts** не будет обрабатывать аргументы, если им не предшествует символ «–» или «+», выделение опций будет прекращено, как только встретится первый аргумент.
2. Типичная конструкция цикла **WHILE** с **getopts** несколько отличается от стандартной из-за отсутствия квадратных скобок, проверяющих условие продолжения цикла.

В следующем сценарии приведен пример использования команды **getopts**:

```
#!/bin/bash
ALL=false
HELP=false
FILE=false
VERBOSE=false
COPIES=0 # значение опции -c равно нулю
while getopts   :ahfgvc: OPTION
do
            case $OPTION in
            a)ALL=true
              echo "ALL is $ALL"
              ;;
            h)HELP=true
              echo "HELP is $HELP"
              ;;
            f)FILE=true
              echo "FILE is $FILE"
              ;;
            v)VERBOSE=true
              echo "VERBOSE'is $VERBOSE"
              ;;
            c)COPIES=$OPTARG
              echo "COPIES is $COPIES"
            \?)
              echo "`basename $0`  -[a h f v]   -[c value]   file"» >&2
              ;;
            esac
done
```

При выполнении указанного выше сценария с опцией **-c**, не содержащей значения, возникает ошибка. В этом случае отображается сообщение **usage**:

```
$ ./getopt -ah -c
ALL is true
HELP is true
getopt -[a h f v] -[c value] file
```

Если выполнить сценарий следующим образом, то будут учитываться все опции:

```
$ ./getopt -ah -c 3
ALL is true
HELP is true
COPIES is 3
```

12.7. Использование функций. Отладка сценариев

До сих пор весь программный код сценариев выполнялся последовательно от начала до конца программы. Подобный подход неплох, но при этом некоторые фрагменты кода дублируются в пределах одного сценария.

Интерпретатор команд **Bash** позволяет группировать наборы команд или конструкций, создавая повторно используемые блоки. Подобные блоки называются ***функциями***.

Функция состоит из двух частей:

- метка функции;
- тело функции.

В качестве ***метки*** выступает имя функции; ***тело*** функции образует набор команд, составляющих саму функцию. Имя функции должно быть уникальным: если это не так, результаты будут плачевны. Это связано с тем, что в случае наличия двух различных функций с одинаковыми именами сценарий просто не сможет вызвать нужную функцию.

Формат, применяемый для определения функций, следующий:

```
имя_функции ()
{
команды...
}
```

Вызов функции осуществляется простым указанием ее имени в тексте сценария.

Функцию можно представлять себе как некоторый вид сценария, находящегося внутри другого сценария, но в этом случае следует учитывать одну особенность. При вызове функции она остается в текущем интерпретаторе, а ее копия хранится в памяти. С другой стороны, если вызывается или выполняется сценарий из другого сценария, создается отдельный интерпретатор. В таком случае становятся недействительными все существующие переменные, определенные в предыдущем сценарии.

Функции могут быть размещены в том же самом файле, что и сценарии, либо в отдельном файле, содержащем функции. При этом функции не всегда включают множество конструкций или команд; может просто содержаться единственная конструкция **echo**, которая выполняется при вызове функции.

Рассмотрим следующий сценарий:

```
#!/bin/bash

hello ()
{
echo "Hello world! Today is $(date)"
}
```

```
echo "Now we going to the function hello"
hello
echo "Back from the function"
exit 0
```

При выполнении данного сценария получатся следующие результаты:

```
$ ./func
Now we going to the function hello
Hello world! Today is Sun Sep 11 19:37:40 MSD 2009
Back from the function
```

Порядок передачи параметров функции аналогичен передаче параметров обычному сценарию. При этом используются специальные переменные $1, $2, ... $9. При получении функцией переданных ей аргументов происходит замена аргументов, изначально переданных сценарию интерпретатора **Bash**. В связи с этим неплохо было бы повторно присвоить значения переменным, получаемым функцией. В любом случае это стоит сделать, поскольку при наличии ошибок в функциях их можно будет легко обнаружить, воспользовавшись именами локальных переменных. Для вызывающих аргументов (переменных), находящихся внутри функции, имя каждой переменной начинается с символа подчеркивания, например:

```
_FILENAME или _filename
```

После естественного завершения выполнения функции либо в том случае, когда она завершается в результате выполнения какого-либо условия, можно выбрать один из двух возможных вариантов:

1) дождаться, пока функция естественным образом не завершится сама с последующей передачей управления той части сценария, которая вызвала данную функцию;
2) воспользоваться ключевым словом **return**, в результате чего будет осуществлена передача управления конструкции, которая расположена за оператором вызова функции. При этом может также указываться и необязательный числовой параметр. Этот параметр принимает значение 0 в случае отсутствия ошибок и значение 1 – при наличии ошибок. Действие данного параметра аналогично действию кода завершения последней команды. При использовании ключевого слова **return** применяется следующий формат:
   - **return** – возвращает результат из функции, использует код завершения последней команды для проверки состояния;
   - **return 0** – применяется при отсутствии ошибок;
   - **return 1** – применяется при наличии ошибок.

В следующем листинге приведен пример использования функций и ключевого слова **return**:

```
char_name ()
{
            # char_name
            # вызов:  char_name string
            # назначение аргумента новой переменной
            _LETTERS_ONLY=$1
```

```
        # использование awk для проверки на наличие символов !
        _LETTERS_ONLY=$(echo  $1|awk  '{if($0~/[^a-z  A-Z]/}print  "1"}')
        if   [ "$_LETTERS_ONLY" != "" ]; then
                # ошибки
                return 1
        else
                # содержит только символы
                return 0
        fi
}
```

Сначала переменной **$1** будет присвоено более осмысленное имя. Затем применяется утилита **awk**, осуществляющая проверку, состоит ли переданная строка из одних литер. В результате выполнения возвращается код, включающий 1 (для символов, не являющихся литерами) и 0 – для символов-литер. Этот код присваивается переменной **_LETTERS_ONLY**. Затем выполняется проверка значения переменной. Если переменной присвоено какое-либо значение, то это свидетельствует о наличии ошибки; в случае отсутствия присвоенного значения ошибки нет. На основании результатов этой проверки формируется код возврата. Использование кода возврата позволяет сценарию фиксировать момент завершения проверки, выполняемой функцией в вызывающей части сценария.

Одной из самых сложных задач при создании сценариев является их отладка. Желательно, чтобы пользователь, выполняющий эту задачу, получил консультации на данном этапе. Чтобы избежать распространенных ошибок, достаточно следовать указанному ниже правилу: разбейте предлагаемый сценарий на задачи или процедуры, затем запрограммируйте и проверьте каждую процедуру и лишь потом переходите к следующему этапу.

Чаще всего при написании сценариев пропускаются кавычки либо ключевое слово **fi** в конце конструкции **if**.

Следует учитывать, что если интерпретатор команд выдает сообщение о наличии ошибки в сценарии, нужно проанализировать не только строку, где может находиться ошибка, но также и блок кода, включающий эту строку. Интерпретатор **Bash** не всегда точно указывает на местонахождение ошибки: сообщение об ошибке обычно появляется после выполнения строки с ошибочным оператором.

Далее приведен перечень инструментов, которые могут помочь при отладке сценариев:

1) команда **echo** в критических точках сценария поможет отследить состояние переменных и отобразить ход исполнения;
2) команда-фильтр **tee**, которая поможет проверить процессы и потоки данных в критических местах;
3) ключи **-n -v –x**:
   – **sh -n <сценарий>** – проверит наличие синтаксических ошибок, не запуская самого сценария. Того же эффекта можно добиться, вставив в сценарий команду **set -n** или **set -o noexec**. Некоторые из синтаксических ошибок не могут быть выявлены таким способом;

– **sh -v <сценарий>** – выводит каждую команду, прежде чем она будет выполнена. Того же эффекта можно добиться, вставив в сценарий команду **set -v** или **set -o verbose**.
  Ключи -n и -v могут употребляться совместно: **sh -nv scriptname**;
– **sh -x <сценарий>** – выводит, в краткой форме, результат исполнения каждой команды с раскрытием аргументов командной строки. Того же эффекта можно добиться, вставив в сценарий команду **set -x** или **set -o xtrace**. Полезно изменить значение переменной PS4 для отображения номера исполняемой строки в процессе отладки следующим образом:
  `export PS4='+ $LINENO: '`
  Вставив в сценарий **set -u** или **set -o nounset**, вы будете получать сообщение об ошибке **unbound variable** всякий раз, когда будет производиться попытка обращения к необъявленной переменной.

Рассмотрим заранее неработоспособный сценарий и определим, почему он не работает:

```
#!/bin/bash
#  Ожидается, что этот сценарий будет удалять в текущем каталоге
#    все файлы, имена которых содержат пробелы.
#  Но он не работает. Почему?
badname=`ls | grep ' '`
# echo "$badname"
rm "$badname"
exit 0
```

Для начала попробуем проверить сценарий на наличие синтаксических ошибок:

```
$ sh -n ./error
$ echo $?
0
```

Если код выполнения равен 0, значит, синтаксические ошибки отсутствуют. Затем проверим сценарий, выполнив его с ключом **–x**:

```
# sh -x error
++ ls
++ grep ' '
+ badname=
+ rm ''
rm: cannot remove `': No such file or directory
+ exit 0
dsds
```

Как видно из вывода, переменная **badname** содержит пустое значение, следовательно, команда rm не сможет удалить переданный ей в переменной **badname** файл. Эту же ошибку можно было бы определить, раскомментировав 6-ю строку в сценарии, которая выводит значение переменной **badname** перед дальнейшей подстановкой.

Модуль 13. Работа с дисковым пространством

Изучив данный модуль, вы научитесь:

- описывать процесс организации хранения данных;
- описывать структуру файловой системы;
- управлять дисковым пространством и определять его характеристики;
- подключать внешние устройства.

13.1. Организация хранения данных

Поскольку все в ОС Linux представлено в виде файлов, от того, каким образом организовано их хранение, будет зависеть многое, в том числе производительность системы ввода-вывода, надежность хранения и масштабируемость. Процесс организации хранения данных условно можно разделить на следующие этапы:

- выбор необходимого устройства для хранения данных;
- создание разделов на данном устройстве и его форматирование (разметка);
- создание и настройка файловой системы на созданных разделах;
- пометка разделов и их монтирование в систему.

13.1.1. Управление разделами

Для создания разделов в ОС Linux существуют две утилиты: **fdisk** и **parted**. Возможности утилиты **parted** шире, с ее помощью, в частности, можно расширять разделы.

Для разметки установленного в системе диска необходимо запустить утилиту **fdisk** следующим образом:

```
fdisk <название_блочного_устройства>
```

Для отображения текущей таблицы разделов используется команда **p**, для создания нового раздела используется команда **n**, для удаления раздела – команда **d**.

При отображении текущей таблицы разделов ОС Linux указываются «сырой» объем доступного дискового пространства (например, 8589 MB, 8589934592 bytes) и геометрия устройства (например, 255 heads, 63 sectors/track, 1044 cylinders). Затем следует непосредственно таблица разделов, состоящая из семи следующих полей:

- устройство;
- является ли данное устройство загрузочным;

- начальный цилиндр;
- конечный цилиндр;
- количество занимаемых блоков;
- идентификатор таблицы разделов;
- тип таблицы разделов (Linux, Linux swap и прочее)[1].

В процессе создания разделов утилита **fdisk** требует ввести тип и порядковый номер раздела. В ОС Linux разделы бывают двух типов: основной **(primary)** и расширенный **(extended)**. Основной раздел может являться загрузочным (содержит символ * в поле Boot), то есть содержать файлы загрузчика GRUB и ядро ОС Linux. Размер раздела можно указывать в единицах Мбайт или Кбайт. После выбора типа и порядкового номера раздела необходимо указать тип таблицы раздела, исходя из того, для чего будет использован данный раздел. Например, если раздел будет использоваться под подкачку, необходимо указать идентификатор 82, который соответствует таблице разделов типа Linux swap.

Утилита **fdisk** не создает раздел мгновенно – для того чтобы изменения вступили в силу, необходимо выполнить команду **w**. В ОС Linux можно создавать на жестком диске до 4 основных раздела и 16 расширенных.

В следующем листинге приведен пример создания раздела размером 1 Гбайт, который будет использоваться как раздел подкачки:

```
Command (m for help): n
Command action
     e  extended
     p  primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-10402, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-10402, default 10402): +1000M

Command (m for help): p

Disk /dev/hdb: 5368 MB, 5368709120 bytes
16 heads, 63 sectors/track, 10402 cylinders
Units = cylinders of 1008 * 512 = 516096 bytes

   Device Boot    Start       End    Blocks    Id  System
/dev/hdb1             1      1939    977224+   83  Linux

Command (m for help): t
Selected partition 1
Hex code (type L to list codes): 82
Changed system type of partition 1 to 82 (Linux swap / Solaris)

Command (m for help): w
```

[1] В данном поле содержится тип таблицы раздела, соответствующий указанному идентификатору раздела. Для просмотра поддерживаемых типов используется команда l в интерактивном режиме работы утилиты fdisk.

Работа утилиты **parted** схожа с работой утилиты **fdisk**. Справочную информацию по командам утилиты **parted** можно просмотреть, введя команду **help** в приглашении **(parted)**, как это показано в следующем листинге:

```
(parted) mkpart
Partition type? primary/extended? primary
File system type? [ext2]? linux-swap
Start? 101MB
End? 1100MB

(parted) print

Disk /dev/sdb: 10.7GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type     File system  Flags
 1      0.51kB  100MB   100MB   primary  ext2
 2      101MB   1100MB  1000MB  primary
```

После того как были выполнены соответствующие операции над разделами (не важно, какой утилитой), необходимо выполнить перезагрузку системы или ввести команду **partprobe**, чтобы перечитать таблицу разделов без перезагрузки системы.

Следующим этапом в организации хранения данных является создание файловой системы. Для создания файловых систем **ext2** или **ext3** предназначена команда **mke2fs -j <имя_раздела>** (опция **-j** указывает на необходимость создания журнала файловой системы). В большинстве случаев достаточно лишь ввести команду **mke2fs – j** и указать имя раздела.

Если необходимо создать раздел подкачки, используется команда **mkswap <имя_устройства>**.

Для определения типа файловой системы произвольного устройства полезно использовать утилиту **blkid**, которая отобразит информацию о файловой системе в следующем виде:

```
# blkid /dev/sda1
/dev/sda1: LABEL="/boot" UUID="058a9e9d-e7c4-41d0-8847-257091ef3d8b"
SEC_TYPE="ext2" TYPE="ext3"
```

Файловая система типа **ext3** состоит из пяти структурных компонентов:

- набора блоков для индексных дескрипторов;
- набора распределенных «суперблоков»;
- таблицы распределения дисковых блоков;
- сводки использования блоков;
- совокупности блоков данных.

Дисковое пространство выделяется для журнала так, как если бы он был обычным файлом в корневом каталоге новой файловой системы, поэтому в действительности он не является отдельным структурным компонентом.

Каждый раздел файловой системы разбивается на группы блоков. Дисковое пространство для групп блоков, занятых таблицами индексных дескрипторов,

выделяется таким образом, чтобы те блоки, обращения к которым взаимосвязаны, хранились на диске подряд. Это группирование уменьшает необходимость поиска по всему диску при обращении к блокам одного файла.

Индексные дескрипторы – это табличные записи фиксированной длины, которые содержат информацию об одном файле. Поскольку пространство для индексных дескрипторов выделяется при начальном структурировании файловой системы, необходимо заранее решить, сколько индексных дескрипторов следует создавать. Точно предсказать, сколько файлов (индексных дескрипторов) понадобится в будущем, невозможно. Команды построения файловой системы используют эмпирическую формулу для оценки этого числа на основании величины раздела и среднего размера файла. При создании файловой системы можно увеличить или уменьшить количество индексных дескрипторов. В файловых системах с множеством мелких файлов (например, в разделах, где хранятся исходные коды программ) индексных дескрипторов должно быть больше, а в файловых системах с несколькими большими файлами (например, в разделах с файлами баз данных) – меньше.

Суперблок – это запись, содержащая характеристики файловой системы. В суперблоке хранятся информация о величине дискового блока, размере и расположении таблиц индексных дескрипторов, таблица распределения дисковых блоков и информация об их использовании, данные о размерах групп блоков и другие важные параметры файловой системы. Поскольку повреждение суперблока приводит к удалению исключительно важной информации, создается несколько копий суперблока, которые размещаются в разных местах (в начале каждой группы блоков). Для каждой монтируемой файловой системы ядро хранит одну копию суперблока в памяти и несколько копий на диске. Системный вызов **sync** периодически записывает кешированные суперблоки на диск, восстанавливая целостность файловой системы. Это позволяет предотвратить ущерб, который может быть нанесен системе при аварийной остановке. Вызов **sync**, кроме того, записывает на диск модифицированные индексные дескрипторы и кешированные блоки данных. В файловой системе **ext3** синхронизация выполняется каждые 5 секунд.

Таблица распределения дисковых блоков содержит сведения о местонахождении свободных блоков в файловой системе. При создании новых файлов происходит анализ этой таблицы на предмет выбора наиболее эффективной схемы размещения данных. Сводка использования блоков отражает основную информацию о блоках, которые уже используются.

После того как на разделе была создана файловая система, его желательно пометить, используя команду **e2label <имя_устройства> <название_метки>**. Данная процедура не является обязательной, однако при добавлении информации о разделе в файл **/etc/fstab** метка[1] раздела может быть использована вместо на-

[1] Особенно полезно в случае использования дисков SCSI. В случае изменения SCSI ID (номера устройства) система будет в состоянии отыскать нужный диск и выполнить монтирование файловой системы.

звания устройства. Если команда **e2label** запускается без указания названия метки, то будет отображено текущее название метки раздела, если таковое имеется.

Все файлы в ОС Linux иерархически организованы в виде единого дерева, с корнем в /, при этом файлы могут находиться на разных физических устройствах. Команда **mount** позволяет присоединить файловую систему, находящуюся на каком-либо устройстве, к этому единому дереву; команда **unmount** позволяет выполнить обратную операцию.

Заключительным этапом организации хранения данных являются создание точки монтирования, выполнение процедуры монтирования и запись сделанных изменений в файл **/etc/fstab**. Типовой пример файла **/etc/fstab** имеет следующий вид:

```
LABEL=/     /        ext3    defaults1       1
/dev/VolGroup00/LogVol00    /home   ext3    defaults,usrquota       1       2

LABEL=/boot /boot    ext3    defaults1       2
tmpfs       /dev/shmtmpfs    defaults0       0
```

Файл **/etc/fstab** имеет шесть полей, разделенных пробелами, которые описаны в табл. 13.1. Каждая строка файла **/etc/fstab** содержит описание одной файловой системы.

Таблица 13.1. Описание полей файла /etc/fstab

Опция	Описание
Метка	Название устройства или его метка
Точка монтирования	Точка монтирования устройства
Тип файловой системы	Тип файловой системы. В ОС Linux по умолчанию поддерживаются следующие типы файловых систем: **ext**, **ext2**, **ext3**, **msdos**, **vfat**, **devpts**, **proc**, **tmpfs**, **udf**, **iso9660**, **nfs**, **smb** и **swap**
Опции монтирования	Опции монтирования. Ключевое слово **defaults** указывает комбинацию опций **rw**, **suid**, **dev**, **exec**, **auto**, **nouser** и **asyn** (эти опции подробно описаны на страницах руководства man команды **mount**). Можно также добавить опцию **acl**, чтобы активизировать расширенные списки контроля доступа
Использование дампинга	Данное поле может содержать либо 0, либо 1. Если указана 1, то данные автоматически сохраняются на диск при помощи команды **dump** при выходе из системы
Порядок проверки файловой системы	В данном поле определен порядок проверки файловых систем утилитой **fschk** в процессе загрузки ОС Linux. Файловая система, содержащая корневой раздел, должна иметь значение 1 в данном поле. Все остальные локальные файловые системы должны иметь значение 2. Удаленные файловые системы должны не проверяться и иметь значение 0 в данном поле

Файл **/etc/fstab** считывают команды **mount**, **umount**, **swapon** и **fsck**, поэтому важно, чтобы представленные в нем данные были полными и достоверными. Команды **mount** и **umount** берут из этого файла недостающую информацию, если в командной строке указано только имя раздела или точка монтирования.

Для того чтобы выполнить монтирование всех локальных файловых систем определенного типа, например **ext3**, необходимо запустить команду **mount** с ключом **-at ext3**.

Команда **mount** считывает файл **/etc/fstab** последовательно, поэтому записи файловых систем, монтируемых к другим файловым системам, должны следовать в файле за описаниями родительских разделов. Например, строка для точки монтирования **/var/log** должна находиться после строки для точки монтирования **/var**, если раздел **/var** представлен как отдельная файловая система.

Для монтирования внешних устройств используется та же команда mount, единственное, что надо помнить, – так это тип монтируемого устройства и тип файловой системы на нем. Например, для монтирования компакт-диска можно использовать следующую команду:

```
# mount -t iso9660 /dev/hdc /mnt
mount: block device /dev/hdc is write-protected, mounting read-only
```

Здесь необязательным ключом **–t** указывается тип файловой системы. Указывать тип файловой системы рекомендуется в случае, если команда **mount** не определяет тип файловой системы самостоятельно.

Для монтирования файлов образов (архив, содержащий файлы и каталоги) стандарта ISO используется следующая команда:

```
# mount -o loop /tmp/debian_lenny.iso /mnt
```

В данном случае выполняется монтирование файла **/tmp/debian_lenny.iso** в точку монтирования **/mnt**. После выполнения монтирования можно приступать к работе в каталоге **/mnt**.

Команда **umount**, предназначенная для размонтирования файловых систем, имеет схожий с командой **mount** синтаксис. Размонтирование файловой системы, в которой какой-либо процесс хранит свой текущий каталог или в которой есть открытые файлы, невозможен. Поэтому, прежде чем выполнять операцию размонтирования, необходимо убедиться, что файлы на смонтированном устройстве никто не использует. Для этого можно воспользоваться следующими рекомендациями.

- Определить, используя команду **pwd**, не находитесь ли вы на файловой системе, которую пытаетесь размонтировать.
- Использовать команды **fuser –m <устройство>** или **lsof <устройство>** (затем **lsof <файл>**), которые отобразят список процессов, использующих данное устройство. После определения процесса, использующего устройство, послать ему завершающий сигнал командой **kill**.
- Использовать команду **umount –f** для принудительного размонтирования недоступных сетевых файловых систем NFS.

- Использовать команду **umount –l** для принудительного размонтирования выбранной файловой системы и удаления всех ссылок на нее (работает в случае, если не срабатывают другие опции, например **–f**).
- Перейти в режим «одного пользователя» (single user mode), используя команду **telinit s**, и размонтировать файловую систему (рекомендуется использовать, если предыдущие варианты не решили проблему).

В случае программного или аппаратного повреждения некоторых областей диска может понадобиться проверка целостности всей файловой системы. Проверка целостности и исправление ошибок на файловой системе выполняются при помощи утилиты **fsck**. Прежде чем проверять файловую систему данной утилитой, ее необходимо размонтировать. Если утилита **fsck** по каким-то причинам не смогла восстановить часть файловой системы, то есть смысл проверить устройство на наличие поврежденных блоков данных, используя утилиту **badblocks**.

В процессе проверки файловой системы утилита **fsck** иногда не в состоянии объединить части одного поврежденного файла воедино, в этом случае фрагменты файла помещаются в специальный каталог – **lost+found**, который располагается на самом верхнем уровне соответствующего каталога файловой системы (например, **/home/lost+found**). Помимо регулярных файлов, в каталог **lost+found** могут попадать другие каталоги и даже специальные файлы. По возможности файлы, попавшие в этот каталог, рекомендуется восстанавливать из резервных копий.

13.2. Определение характеристик дискового пространства

В ОС Linux основной характеристикой дискового пространства является его объем, который динамически изменяется в процессе работы с системой.

Для получения информации по объему дискового пространства используются две команды – **df** и **du**.

Команда **df** используется для определения количества свободных блоков на диске во всех смонтированных файловых системах или указанных объектах. Команда имеет следующий синтаксис:

```
df [опции] [имя]
```

В качестве параметра имя может быть задано имя устройства, имя точки монтирования, имя каталога или удаленной файловой системы.

Наиболее часто используемыми при работе с командой **df** являются следующие опции:

- **a** – выводить сообщения обо всех файловых системах;
- **h** – выводить информацию в удобочитаемой форме;
- **i** – выводить информацию по количеству используемых файловых дескрипторов;
- **l** – вывести информацию только по локальным файловым системам;
- **x <тип>** – вывести информацию только по файловым системам, не относящимся к заданному типу;

- **B <n>** – вывести информацию о дисковом пространстве в блоках размером n байт.

В следующем листинге приведен пример вывода команды df:

```
# df -hT
Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/hda1     ext3    7.3G  6.7G  191M  98% /
none         tmpfs    982M     0  982M   0% /dev/shm
```

В данном примере в выводе команды **df** присутствуют несколько колонок:

- Файловая система (**Filesystem**). В данном поле отображается устройство, которое в настоящий момент смонтировано к системе.
- Тип файловой системы (**Type**). В данном необязательном поле указывается тип файловой системы.
- Общий размер (**Size**). В данном поле указывается общий объем доступного дискового пространства.
- Занятый объем (**Used**). В данном поле указывается объем занятого дискового пространства.
- Доступный объем (**Avail**). В данном поле указывается общий объем доступного дискового пространства.
- Процент занятости файловой системы (**Used %**). В данном поле указывается процент занятого дискового пространства.
- Точка монтирования (**Mounted on**). В данном поле указывается точка монтирования файловой системы.

Команда **du** выводит информацию об использовании дискового пространства каталогами файловой системы, а также файлами, содержащимися в данных каталогов, а именно количество блоков, используемое каждым объектом. Команда имеет следующий синтаксис:

```
du [опции] [объект]
```

Размер[1] блока по умолчанию составляет 1024 байта. Наиболее часто используемыми при работе с командой **du** являются следующие опции:

- **B <размер>** – использовать заданный размер блока;
- **h** – выводить информацию в удобочитаемой форме;
- **i** – выводить информацию по количеству используемых файловых дескрипторов;
- **L** – при наличии ссылки обрабатывать сам файл, на который ссылается ссылка, а не саму ссылку;
- **X <файл>** – исключить файлы из обработки, указанные в заданном файле;
- **--exclude=<образец>** – исключить файлы из обработки, соответствующие образцу;
- **s** – выводить только общую сумму для каждого из перечисленных каталогов;
- **--max-depth <уровень>** – вычислять размер каталогов на заданное число уровней относительно указанного каталога.

[1] Размер блока файловой системы можно определить при помощи команды **tune2fs**.

В следующем листинге приведен пример вывода команды **du** для объектов, содержащихся в каталоге **/tmp**, включая сам каталог **/tmp**:

```
# du -h --max-depth 1 /tmp
568K    /tmp/.uli_mod
108K    /tmp/rrdbuild
524K    /tmp/Pod-Simple-3.07
4.0K    /tmp/ssh-ZmtvZC7346
4.0K    /tmp/.ICE-unix
8.0K    /tmp/boot
4.0K    /tmp/.font-unix
13M     /tmp/sysreport.RLa23257
15M     /tmp/
```

Следует отметить, что команда **du** сможет подсчитать информацию только по тем файлам, которые доступны для чтения пользователю, который запускает команду.

Модуль 14. Сетевые клиенты

Изучив данный модуль, вы научитесь:

- настраивать сетевые интерфейсы;
- использовать средства сетевой диагностики;
- выполнять удаленный доступ к ОС Linux;
- работать с веб-клиентами и клиентами электронной почты.

14.1. Настройка сетевых интерфейсов

Основной задачей при настройке сетевых интерфейсов является присвоение IP-адресов. В ОС Linux существуют два варианта их присвоения: статический и динамический. В первом случае администратор сам устанавливает IP-адрес и соответствующие параметры сетевого интерфейса, во втором случае IP-адрес и параметры сетевого интерфейса устанавливаются автоматически при помощи сервиса DHCP.

Команда **ifconfig** по умолчанию выводит информацию обо всех присутствующих в системе сетевых интерфейсах. Имена интерфейсов в ОС Linux обозначаются как **eth0**[1], **eth1** и т. д. Исключение из данного правила составляет интерфейс обратной связи, который обозначается как **lo**. Каждому из отображаемых сетевых интерфейсов соответствует группа параметров, представленных в табл. 14.1.

Таблица 14.1. Параметры вывода команды ifconfig

Параметр	Описание
HWaddr	MAC-адрес сетевого интерфейса
inet addr	IP-адрес сетевого интерфейса
Bcast	Широковещательный адрес подсети
Mask	Маска подсети
RX packets	Статистика по принятым пакетам
TX packets	Статистика по отправленным пакетам
collisions	Статистика по коллизиям Ethernet
RX bytes	Количество байт, принятых через интерфейс с момента его активации
TX bytes	Количество байт, отправленных через интерфейс с момента его активации

[1] Название сетевого интерфейса в ОС Linux на самом деле является псевдонимом, описанным в файле **/etc/modprobe.conf**, где указано и название самого драйвера сетевого интерфейса. Как правило, большинство драйверов в ОС Linux выполнены в виде динамически загружаемых модулей ядра.

В самом простом случае команда **ifconfig** имеет следующий синтаксис:

```
ifconfig  [интерфейс]  [параметры]
```

интерфейс – имя сетевого интерфейса, который необходимо настроить или отобразить информацию;
параметры – параметры сетевого интерфейса.

Команда **ifconfig** позволяет настраивать следующие основные параметры сетевого интерфейса:

- IP-адрес;
- маску подсети;
- широковещательный адрес.

```
# ifconfig eth0 192.168.137.10 netmask 255.255.255.0 broadcast 192.168.137.255
```

После перезагрузки системы настройки сетевого интерфейса, сделанные командой **ifconfig**, не сохраняются. В команде **ifconfig** не указываются такие важные параметры сетевого интерфейса, как маршрут по умолчанию и сервер разрешения имен DNS. Эти параметры хранятся в специальных конфигурационных файлах, приведенных в табл. 14.2.

Таблица 14.2. Основные конфигурационные файлы сетевого взаимодействия ОС Linux

Конфигурационный файл	Описание
/etc/sysconfig/network	Файл содержит имя хоста, маршрут по умолчанию и дополнительные параметры сети
/etc/sysconfig/network-scripts/ifcfg-интерфейс	Каталог содержит скрипты, которые включают и выключают сетевые устройства, а также конфигурационные файлы сетевых интерфейсов
/etc/modprobe.conf	Файл содержит соответствие модулей ядра сетевым устройствам
/etc/hosts	Информация об IP-адресах и соответствующих им именах хостов
/etc/resolv.conf	Указываются DNS-серверы, а также дополнительные параметры, связанные с настройкой DNS клиента
/etc/rc.d/init.d/network	Инициализационный скрипт, использующийся для активации и деактивации сетевого стека ОС

Для настройки сетевого интерфейса **eth0** необходимо отредактировать файл **/etc/sysconfig/network-scripts/ifcfg-eth0**. Для настройки маршрута по умолчанию и имени хоста необходимо отредактировать файл **/etc/sysconfig/network**. Типовые примеры файлов **/etc/sysconfig/network и /etc/sysconfig/network-scripts/ifcfg-eth0** приведены в следующих двух листингах.

```
# cat /etc/sysconfig/network
NETWORKING=yes
NETWORKING_IPV6=no
```

```
HOSTNAME=rhel5.linux.lab
GATEWAY=192.168.137.2
# cat /etc/sysconfig/network-scripts/ifcfg-eth0
# Advanced Micro Devices [AMD] 79c970 [PCnet32 LANCE]
DEVICE=eth0
BOOTPROTO=static
BROADCAST=192.168.137.255
HWADDR=00:0C:29:07:FC:29
IPADDR=192.168.137.4
NETMASK=255.255.255.0
NETWORK=192.168.137.0
ONBOOT=yes
```

В табл. 14.3 и 14.4 приведены описания параметров, содержащихся в этих файлах.

Таблица 14.3. Описание параметров сети

Параметр	Значение
NETWORKING	Указывает, следует ли активизировать сетевой стек. Возможные значения: yes или no
HOSTNAME	Указывает имя хоста
GATEWAY	Указывает маршрут по умолчанию

Таблица 14.4. Описание параметров сетевого интерфейса

Параметр	Значение
BOOTPROTO	Определяет тип протокола автоматической настройки интерфейса. Возможные значения: **none**, **dhcp** или **bootp**
STARTMODE	Определяет, будет активирован данный интерфейс в процессе загрузки ОС или нет. Возможные значения: **onboot** или **manual**
IPADDR	Указывает IP-адрес сетевого интерфейса
NETMASK	Указывает маску подсети
NETWORK	Указывает адрес подсети
BROADCAST	Указывает широковещательный адрес подсети

14.2. Сетевая диагностика

Основными утилитами сетевой диагностики ОС Linux являются **ping, tarceroute, netstat** и **tcpdump**.

Команда **ping** посылает ICMP-пакет **ECHO_REQUEST** конкретному хосту и ждет от него ответа **ECHO_RESPONSE**. Несмотря на простоту, команда **ping** является одним из основных инструментов тестирования сети. В обработке ее запроса участвуют таблицы маршрутизации, физические компоненты сетей и сетевые шлюзы, поэтому для достижения успешного результата сеть должна быть в более или менее рабочем состоянии. Команда **ping** имеет следующий синтаксис:

```
ping [опции] место_назначения,
```

где опции – разнообразные параметры, позволяющие управлять посылкой ICMP-пакетов. Наиболее часто используются следующие опции:

- **c (--count)** – количество посылаемых пакетов. Если этот аргумент не задан, то команда **ping** работает в бесконечном цикле. Чтобы прервать работу команды, нужно нажать клавиши **<Ctrl+C>**;
- **i (--interval)** – интервал времени в секундах между посылкой двух пакетов;
- **s** – используется для указания размера посылаемого пакета в байтах.

место_назначения – целевой объект (например, хост), куда необходимо посылать ICMP-пакеты. В следующем листинге содержится пример использования команды **ping**.

```
# ping -c 3 192.168.137.3
PING 192.168.137.3 (192.168.137.3) 56(84) bytes of data.
64 bytes from 192.168.137.3: icmp_seq=1 ttl=64 time=2.71 ms
64 bytes from 192.168.137.3: icmp_seq=2 ttl=64 time=0.289 ms
64 bytes from 192.168.137.3: icmp_seq=3 ttl=64 time=0.862 ms
--- 192.168.137.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.289/1.288/2.715/1.036 ms
```

В выводе команды **ping** отображается статистика количества переданных и принятых пакетов (**packets transmitted/received**), времени сессии (**time**) и среднего времени отклика (**rtt**).

Команда **traceroute** отображает последовательность хостов, через которые проходит IP-пакет по пути к месту назначения. Данная команда используется в основном для проверки корректности сетевых маршрутов. Команда **traceroute** имеет следующий синтаксис:

```
traceroute [опции] пункт_назначения
```

опции – разнообразные параметры команды **traceroute**;
пункт_назначения – целевой объект (например, хост), куда необходимо посылать IP-пакеты.

Вывод команды **traceroute** представляет собой перечень хостов, начинающийся с первого хоста и заканчивающийся пунктом назначения. В выводе команды **traceroute** отображается статистика по количеству пройденных хостов и времени прохождения IP-пакетов.

Команда **netstat** собирает различную информацию о состоянии сетевого программного обеспечения, включая статистику сетевых интерфейсов, данные о маршрутизации и таблицы соединений. Данную команду используют в следующих ситуациях:

- при анализе конфигурационных данных интерфейсов;
- при мониторинге состояния сетевых соединений;
- для выявления прослушивания сетевых сервисов;
- при изучении таблицы маршрутизации;
- для просмотра статистических данных сетевых протоколов.

В следующем листинге приведен пример использования команды **netstat** для просмотра сервисов и соответствующих портов, которые находятся в режиме прослушивания TCP-трафика:

```
# netstat -tl
```

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 rhel5.linux.lab:smtp    *:*                     LISTEN
tcp        0      0 rhel5.linux.lab:2207    *:*                     LISTEN
tcp        0      0 *:ssh                   *:*                     LISTEN
```

В данном выводе команды **netstat** в поле **State** указан статус соединения, в полях **Recv-Q** и **Send-Q** указывается количество запросов, стоящих во входящих и исходящих очередях на данном порте. Локальные и удаленные адреса указаны в формате **имя_хоста:название_сервиса**, где вместо названия сервиса может указываться порт. Для сопоставления имен сервисов и портов команда **netstat** использует файл **/etc/services**.

Команда **tcpdump** относится к классу программ сетевых анализаторов[1] и используется для детального анализа сетевых пакетов, проходящих через сетевые интерфейсы, и является промышленным стандартом. Команда **tcpdump** имеет следующий синтаксис:

```
tcpdump [опции] [условие]
```

`опции` – разнообразные параметры команды **tcpdump**;
`условие` – специальное выражение, которому должны удовлетворять пакеты.

Пример использования команды **tcpdump** приведен в следующем листинге.

```
# tcpdump 'tcp port 22'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
13:38:36.535475 IP 192.168.137.4.ssh > 192.168.137.1.high-criteria: P
282252:282400(148) ack 1249 win 12236
13:38:36.535807 IP 192.168.137.4.ssh > 192.168.137.1.high-criteria: P
282796:282944(148) ack 1249 win 12236
```

В выводе команды указывается сетевой интерфейс (в примере – **eth0**), на котором осуществляется анализ пакетов, далее указываются тип соединения (в примере – Ethernet) и размер блока обрабатываемых данных (в примере – 96 байт). После этого следует непосредственное отображение сетевых данных. В данном случае команда **tcpdump** была запущена с целью отображения пакетов протокола TCP, а в качестве порта был указан 22-й порт. В результате команда **tcpdump** отобразила несколько пакетов, принадлежащих сеансу **ssh**, с указанием отправителя и получателя, размером окна и номера последовательности **ACK**.

14.3. Инструменты удаленного доступа и администрирования

В ОС Linux, как и в любой другой ОС, имеются средства удаленного доступа и администрирования. Эти средства условно можно разделить по задачам, для выполнения которых они предназначены:

[1] Наиболее универсальным анализатором сетевых пакетов является программа **wireshark** (http://www.wireshark.org/), позволяющая производить анализ пакетов в графическом режиме и имеющая хорошие возможности по фильтрации, формату входных данных и методам обработки.

- удаленное выполнение команд (**ssh**);
- прием и передача данных (**scp, lftp, rsync**);
- удаленное администрирование (**ssh, vnc, rdesktop**).

Основным универсальным средством, подходящим для выполнения всех задач, является защищенный интерпретатор команд **Secure Shell** (ssh).

Интерпретатор **ssh** является заменой небезопасному интерпретатору **telnet**. Он использует защищенную аутентификацию для подтверждения личности пользователя и шифрует все данные, передаваемые от одного хоста к другому.

В ОС Linux версия защищенного командного интерпретатора **ssh** называется **OpenSSH**. Принцип работы **OpenSSH** основывается на архитектуре «клиент-сервер». Система, к которой необходимо подключиться, выступает в качестве сервера. В данной системе функционирует демон **sshd**, который обслуживает удаленные подключения. Система, которая подключается к демону **sshd,** выступает в качестве клиента. Помимо базовых возможностей, таких как работа в командной строке, интерпретатор **ssh** позволяет организовывать безопасную трансляцию сеансов **X Window** с удаленного хоста на хост администратора, позволяя тем самым работать в графической среде удаленной системы при помощи локального терминала.

Еще одной замечательной возможностью, которой обладает данный интерпретатор, является перенаправление портов (**port forwarding**), при помощи которого возможно перенаправлять соединения к серверу, на котором настроено перенаправление, на удаленный сервер, который используется для конечной обработки запроса.

Настройки клиентской части **OpenSSH** содержатся в конфигурационном файле **/etc/ssh/ssh_config**, который содержит директиву **Host**, определяющую применимость всех директив данного файла к указанному хосту, а также некоторые директивы, присутствующие в файле **sshd_config**.

К основным операциям, выполняемым при помощи интерпретатора **ssh**, относятся следующие.

- **Подключение к удаленному хосту** с использованием следующей команды:
  ssh <имя_пользователя>@<имя_хоста>,
  где в качестве аргумента **<имя_пользователя>** необходимо ввести соответствующее регистрационное имя удаленного пользователя, а в аргументе <имя_хоста> указать DNS-имя или IP-адрес удаленного хоста.
- **Выполнение команды на удаленном хосте.** Для ее запуска используется следующая команда:
  ssh <имя_пользователя>@<имя_хоста> <команда>,
  где в качестве аргумента **<команда>** необходимо указать нужную команду. Следует отметить, что бинарный файл, указанный в данной команде, должен существовать на удаленном сервере.
- **Передача файлов на удаленный хост.** Данная команда имеет следующий синтаксис:
  scp <локальный_файл> <имя_пользователя>@<имя_хоста>:<удаленный_файл>

Вместо аргумента **<локальный_файл>** и **<удаленный_файл>** необходимо, соответственно, указать локальный и удаленный файлы.

- **Перенаправление портов** позволяет шифровать данные, передаваемые по незащищенным портам. Например, клиент, находящийся в сети Интернет, может забирать свои почтовые сообщения по протоколу POP3 с почтового сервера, находящегося в локальной сети организации, и при этом быть уверен, что передаваемые им данные защищены.
  Основной синтаксис команды, используемой для перенаправления локальных портов, следующий:
  ssh -L <локальный_порт>:<удаленный_хост>:<удаленный_порт> <имя_пользователя>@<удаленный_хост>
  Когда клиент выполняет подключение к серверу на локальный порт, соединение перенаправляется на удаленный порт удаленного сервера. Далее выполняется аутентификация клиента на основе имени пользователя и хоста, который содержит его учетную запись. При этом формируется шифрованный туннель между клиентом и удаленным хостом.

Для передачи файлов в ОС Linux есть множество команд, основными из которых являются команды **scp**, **lftp** и **rsync**. Каждая из этих команд использует свой протокол передачи данных и применяется, исходя из объемов и требований к надежности передачи.

Утилита **scp** используется для передачи данных по защищенному протоколу **SSH**. Она очень удобна для перемещения небольшого количества небольших файлов, и может использоваться в качестве хорошего средства при передаче файлов поверх публичных сетей. В следующем листинге приведен пример рекурсивного копирования файлов на удаленный хост:

```
# scp -r /var/log root@10.31.5.160:/opt
The authenticity of host '10.31.5.160 (10.31.5.160)' can't be established.
RSA key fingerprint is 5d:9f:06:97:07:4e:d9:fa:12:14:a1:e1:84:df:ad:15.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.31.5.160' (RSA) to the list of known hosts.
anaconda.xlog                                 100%   60KB  60.2KB/s
00:00
1.dbf                                         100%    0     0.0KB/s
00:00
mrtg.log                                        4%   10MB   2.7MB/s
01:19 ETA
```

Сразу после ввода команды предлагается убедиться в подлинности хоста, к которому выполняется подключение. Это сделано с целью защиты от подмены имени хоста, к которому выполняется подключение. Если согласиться с установкой соединения, то соответствующий публичный ключ удаленного хоста будет добавлен в список известных хостов (по умолчанию **~/.ssh/known_hosts**). После установки соединения выполняется процесс передачи файлов на удаленный хост. В процессе передачи файлов на экране отображаются имена передаваемых файлов, процент завершения передачи (в том числе и по объему), текущая скорость передачи данных (Mb/s) и время завершения передачи (ETA).

Помимо утилиты **scp**, в ОС Linux для копирования файлов на удаленные хосты, а также для синхронизации между локальными каталогами широко используется утилита **rsync**. Кроме того, большинство файлообменных сервисов в сети Интернет поддерживают протокол **rsync** для загрузки или скачивания файлов.

Основным преимуществом команды **rsync** при копировании файлов является то, что копируются **только изменения**, сделанные в файле, тем самым существенно уменьшается общее время копирования. Кроме того, утилита **rsync** позволяет сжимать данные в процессе передачи и осуществлять надежный контроль целостности данных.

Например, для копирования домашнего каталога пользователей **/home** в каталог **/backups**, находящийся на удаленном сервере **backup.linux.lab**, используется следующая команда:

```
# rsync -avz /home backup.linux.lab:backups/
```

Здесь аргумент **-a** используется для архивирования данных, то есть выполняется рекурсивное копирование всех подкаталогов вместе с их файлами с сохранением прав доступа, символьных ссылок, временных оттисков файлов, владельца и группы файлов. Аргумент **-z** используется для сжатия данных перед копированием.

При копировании файлов на удаленный хост важно наличие или отсутствие символа «/» в конце названия копируемого каталога. Если символ «/» в конце названия копируемого каталога отсутствует, на удаленный хост будет создан аналогичный каталог со всем его содержимым. Если данный символ указан, то на удаленном хосте будут скопированы только файлы, содержащиеся в копируемом каталоге.

Если необходимо выполнить загрузку или выгрузку файлов большого объема, то самым быстродействующим способом будет использование протокола **FTP** передачи данных. Для обмена данными по данному протоколу в ОС Linux имеется множество клиентов, однако наиболее простым и понятным является утилита **lftp**, которая также позволяет передавать файлы по протоколам SFTP, HTTP и HTTPS.

Для работы с программой **lftp** в первую очередь необходимо узнать IP-адрес хоста (или DNS-имя), к которому будет осуществляться подключение, учетное имя и пароль для доступа к файлам. Для запуска программы **lftp** используется следующая команда:

```
# lftp
```

Работа с командой **lftp** выполняется в интерактивном режиме, после появления соответствующего приглашения **lftp :~>**.

Далее необходимо подключиться к хосту, введя команду:

```
lftp :~> open <хост>
```

Затем необходимо аутентифицироваться, введя команду:

```
lftp :~> login <имя_пользователя>
```

Если все данные были указаны корректно, **lftp** подключится к хосту, и будет получен доступ к содержимому корневого каталога.

Для копирования файлов с удаленного хоста на локальную машину и наоборот используется следующая команда:

```
lftp :~> get <имя_файла>
```

В следующем листинге приведен пример копирования файлов:

```
lftp:~> get README -o debian.README README.mirrors -o debian.mirrors
```

Опция «-o» позволяет задать новое локальное имя для скачиваемого файла.

В случаях, когда нужно скопировать не отдельный файл или группу файлов в соответствии с шаблоном, а целый каталог со всем его содержимым, следует использовать команду **mirror**.

```
lftp:~> mirror remote_dir local_dir
```

Здесь выполняется зеркалирование удаленного каталога **remote_dir** в локальный каталог **local_dir**.

Во время запуска утилита **lftp** считывает файл **/etc/lftp.conf** и извлекает оттуда параметры настройки, потом проверяются пользовательские файлы (**~/.lftprc** и **~/.lftp/rc**).

Следующим классом программ являются программы, предназначенные для удаленного администрирования посредством подключения к удаленному рабочему столу. Среди этих программ можно выделить **vnc** и **rdesktop**.

Программа **vnc** использует свой собственный протокол передачи данных. Управление осуществляется путем передачи нажатий клавиш на клавиатуре и движений мыши с одного компьютера на другой и ретрансляции содержимого экрана через сеть. Программа **vnc** платформонезависима: VNC-клиент, называемый **vnc viewer**, запущенный на одной операционной системе, может подключаться к **vnc-серверу**, работающему на любой другой ОС. К одному vnc-серверу одновременно могут подключаться множественные клиенты. Наиболее штатные способы использования **vnc** – это удаленная техническая поддержка и доступ к рабочему столу из публичных сетей. В ОС Linux наибольшее распространение получила программа **TightVNC**, содержащая клиентскую и серверную части.

Для подключения к рабочему столу пользователя с удаленной машины необходимо выполнить следующую команду:

```
# vncviewer <удаленный_хост>:0
```

Например, если имя удаленного хоста – **linux.example.com**, то команда будет выглядеть следующим образом:

```
# vncviewer linux.example.com:0
```

Утилита **vncviewer** также имеет графический интерфейс пользователя, который можно запустить из меню **Приложения** (Applications) ⇒**Стандартные** (Accessories) ⇒**VNC Viewer**.

В дополнение к **vncviewer** можно также подключаться к рабочим столам пользователей Windows или Linux при помощи **Клиента терминального сервера** (Terminal Server Client), который можно запустить из меню **Приложения** (Applications) ⇒**Интернет** (Internet) ⇒**Клиент терминального сервера** (Terminal Server Client).

Клиент терминального сервера работает как графический интерфейс к утилитам командной строки **vncviewer** и **rdesktop** и может быть запущен из командной строки при помощи команды **tsclient**.

Клиент терминального сервера – это стандартное приложение среды **GNOME** для удаленного доступа к терминальным службам **Microsoft Windows XP/2000/2003/2008**. Он также поддерживает подключения, используя другие методы работы с удаленным рабочим столом, такие как **vnc**, **xest** и **citrix**.

В следующем листинге приведен пример запуска терминального клиента с командной строки:

```
# export DISPLAY=10.31.88.70:0.0
# rdesktop -u administrator -p q1 -g 800x600 10.31.5.56:3389
```

Как видно из приведенного листинга, в параметрах команды **rdesktop** можно указывать разрешение экрана, учетную запись пользователя и удаленный хост (обязательный параметр). Отметим, что если на локальном хосте отсутствует графическая система **X Window**, то перед запуском команды **rdesktop** (или любой другой команды, использующей в работе графику) необходимо экспортировать переменную **DISPLAY**, в которой должно содержаться место расположения графического дисплея.

14.4. Работа с почтовыми и веб-клиентами

14.4.1. Веб-обозреватель Mozilla Firefox

Веб-обозреватели ОС Linux обычно могут отображать информацию, предоставляемую различными типами серверов, а не только HTTP-серверами, посылающими клиентам страницы формата HTML. Например, при обращении к документу по протоколу FTP отобразится листинг каталогов FTP-сервера. Популярным веб-обозревателем является **Mozilla Firefox** – мощный веб-обозреватель, предназначенный для пользователей любого уровня знаний. Его принципы – согласие стандартов и простота. Интерфейс **Mozilla Firefox** обладает почти неограниченными функциями, что делает его приемлемым для работы в любых случаях, будь то стандартный просмотр веб-страниц или же отладка сценариев **JavaScript**.

После установки программы **Mozilla Firefox** в меню **Приложения** (Applications) ⇒**Интернет** (Internet) появится соответствующий ярлык **Firefox**, который служит для запуска программы. Обозреватель **Mozilla Firefox** относится к обозревателям вкладочного типа, то есть новые страницы можно открывать в новых вкладках, доступ к которым выполняется из единого окна обозревателя.

Вкладка представляет собой «выступ» с надписью, расположенный на границе выделенной под сменное содержимое области экрана. Клик мышью по вкладке делает ее активной, и на управляемой вкладками области экрана отображается соответствующее ей содержимое. Вкладки располагаются друг за другом горизонтально, реже вертикально.

Типовое окно веб-обозревателя **Mozilla Firefox** представлено на рис. 14.1.

При первом запуске **Mozilla Firefox** открывает начальную страницу – по умолчанию это будет домашняя страница обозревателя. Впоследствии можно сменить ее на любую другую, введя адрес нужной страницы в адресной строке и нажав на клавишу **<Enter>**.

Список посещенных ресурсов сохраняется, и с помощью адресной строки можно выбирать соответствующие ссылки из списка. Чтобы повторно зайти на

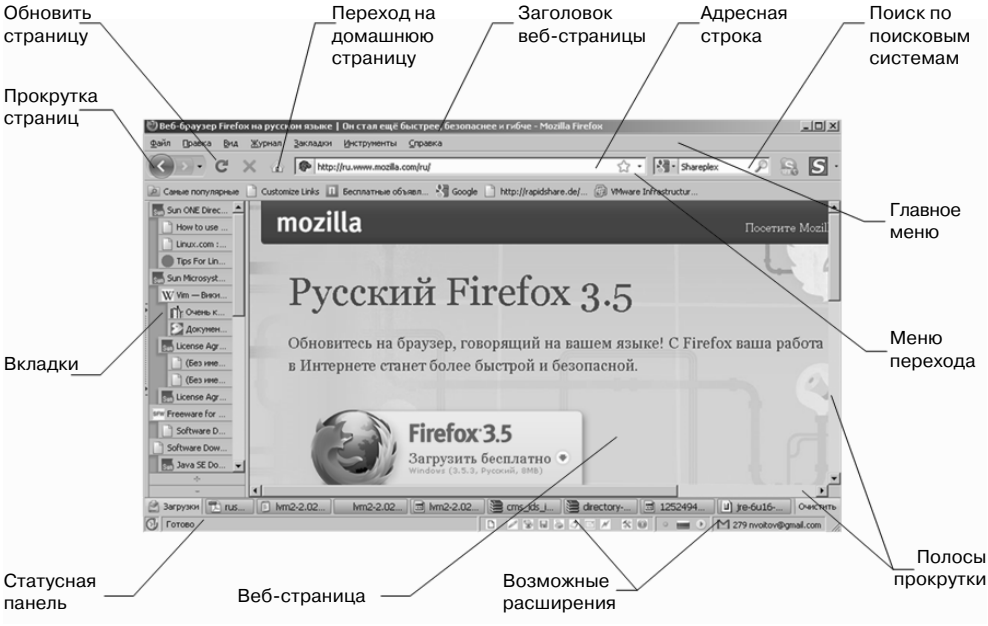


Рис. 14.1. Окно веб-обозревателя Mozilla Firefox

какую-либо страницу из тех, которые уже были просмотрены в течение данного сеанса работы, необходимо открыть меню **Переход** и выбрать из ниспадающего списка нужную страницу. **Панель быстрого поиска** (поиск по поисковым системам) предназначена для поиска информации по конкретным поисковым системам, таким как **google**, **yahoo**, **wikipedia**, **yandex** и прочее. Существует возможность добавлять большое количество поисковых систем и подключать собственные поисковые системы. Панель вкладок по умолчанию располагается над веб-страницей, однако при помощи специальных расширений, упрощающих навигацию при большом количестве открытых вкладок, панель вкладок может располагаться в любом месте экрана. Для открытия новых вкладок используется комбинация клавиш **<Ctrl+T>**. Для переключения между вкладками используются клавиши **<Ctrl+Shift+Tab>**. Клавиша **<Ctrl+W>** используется для закрытия вкладки. Полный список наиболее используемых клавиш представлен в приложении 14.1.

Статусная панель служит для отображения служебной информации, такой как загрузка страницы, а также местом для информативных расширений, например следящих за обновлением внешнего почтового ящика.

Навигация по веб-странице осуществляется при помощи полос прокруток, а также специальных комбинаций клавиш, приведенных приложении 14.1.

Для выполнения типовых задач над страницей, таких как печать страницы, сохранение страницы, просмотр журнала или добавление страницы в закладки, используется соответствующее меню из панели **Главное меню**.

Веб-обозреватель **Mozilla Firefox** обладает огромным количеством ***расширений*** – плагинов, которые позволяют сделать работу с обозревателем максимально

комфортной. Например, существуют дополнения по переводу страниц, загрузке файлов, редактированию страниц перед отправкой на печать. Расширения добавляют новые возможности в программы или разрешают модифицировать существующие настройки. Они могут добавить практически что угодно: от кнопки на панели инструментов до совершенно новых возможностей.

Механизм расширений превращает изначальную простоту обозревателя **Mozilla Firefox** в одно из основных его преимуществ: устанавливая расширения, пользователь может выбрать именно ту функциональность, которая необходима ему для комфортного просмотра веб-контента, при этом не занимая рабочее пространство и ресурсы ненужными функциями. На рис. 14.1 в окне обозревателя присутствуют несколько расширений, таких как статус почтового ящика, строка недавних загрузок, панель управления снимками экрана и настройка перевода текста.

14.4.2. Текстовые веб-клиенты: lynx и wget

В ОС Linux, помимо графических веб-обозревателей, существуют текстовые обозреватели, позволяющие просматривать страницы в режиме командной строки. Одним из первых текстовых веб-обозревателей является **lynx**[1]. Данный обозреватель поддерживает почти все возможности HTML, включая SSL, и в основном используется для просмотра справочных страниц в формате HTML, а также для проверки правильности отображения веб-страниц. Особенно полезно использовать **lynx** для быстрого просмотра веб-страницы, когда нет времени ждать полной загрузки графики или же надо выполнить некоторые действия над веб-страницей в программном сценарии. Запуск обозревателя lynx выполняется при помощи одноименной команды, имеющей следующий синтаксис:

```
lynx [опции] [путь | URL-адрес]
```

Команда **lynx** имеет очень большое количество опций, наиболее часто из которых используются следующие:

- **color** – использовать цветовую поддержку, если в настройках терминала нет информации об управлении цветами;
- **dump** – направить отформатированные данные в стандартный канал вывода;
- **editor=<редактор>** – использовать в качестве внешнего редактора указанный редактор;
- **justify** – выполнить выравнивание текста;
- **term=<тип_терминала>** – использовать заданный тип терминала;
- **traversal** – обойти все HTTP-ссылки, присутствующие в указанном пути или URL-адресе;
- **use_mouse** – использовать мышь в случае наличия библиотеки **ncurses**;
- **vikeys** – использовать клавиши перемещения по странице, аналогичные используемым в редакторе **vi**;
- **width=<N>** – использовать N колонок для форматирования вывода.

---

[1] Помимо веб-обозревателя lynx, существует более совершенный вариант – веб-обозреватель **ELinks** (http://www.elinks.cz/), поддерживающий **JavaScript**, таблицы и многое другое.

Полный список опций доступен в справочном руководстве **man**.

На рис. 14.2 приведено типовое окно веб-обозревателя **lynx**.

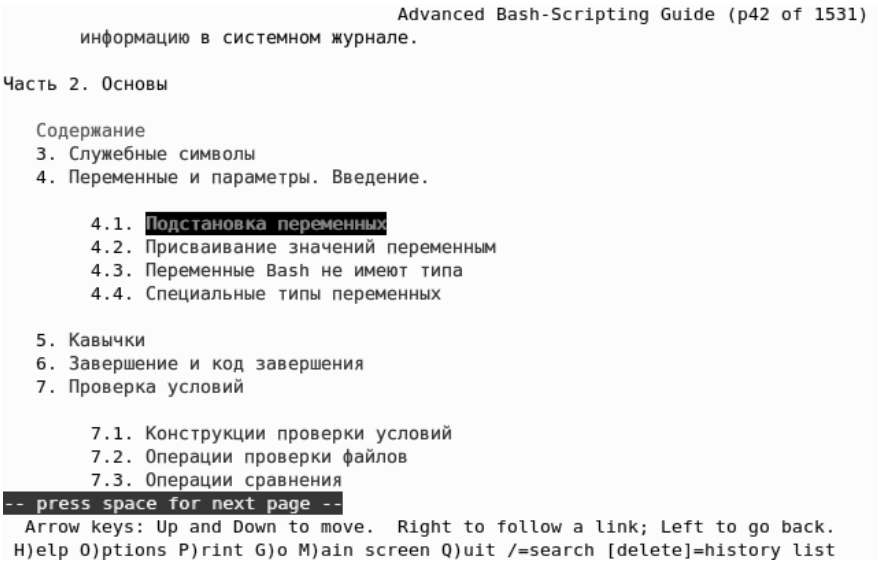


Рис. 14.2. Окно веб-обозревателя Lynx

Внизу основного окна присутствует краткий список основных команд, доступных в интерактивном режиме. Если нажать на клавишу <**H**>, то будет выведена справочная информация по использованию обозревателя.

Программа **Wget** – утилита для загрузки файлов по сети с использованием протоколов HTTP/HTTPS/FTP. Программа **Wget** является неинтерактивной программой. Так как большинство веб-обозревателей рассчитаны на интерактивное взаимодействие с пользователем, то скачивание большого количества файлов вручную будет в этом случае довольно долгим занятием. Программа **Wget** поддерживает загрузку URL-адресов, указанных в файле, что может существенно сэкономить время при скачивании большого количества файлов или даже целых сайтов. Файлы можно скачивать рекурсивно по ссылкам в HTML-страницах, как с одного сайта с определенной глубиной следования по ссылкам, так и с нескольких. Помимо этого, при загрузке по FTP файлы можно скачивать по маске имени (то есть можно использовать подстановки имен файлов, такие как «*»). Программа **Wget** поддерживает докачку файла в случае обрыва соединения, а также умеет использовать прокси-сервер для доступа к ресурсам.

Команда **wget** имеет следующий синтаксис:

```
wget [опции] [URL-адрес]
```

Wget имеет большое количество опций, наиболее часто из которых используются следующие:

- **v** – использовать подробный режим вывода;

- **i <файл>** – обработать список URL-адресов из файла;
- **B <URL-адрес>** – использовать в качестве базы указанный URL-адрес в случае указания в файле адресов **(-i)** относительных ссылок;
- **t <N>** – количество попыток скачивания;
- **O <файл>** – использовать для указания выходного файла, в который будет выполняться конкатенация всех скаченных файлов;
- **c** – продолжить докачку файлов;
- **S** – выводить ответы сервера;
- **--limit-rate=<N>k** – ограничить скорость закачки до **N** Кбайт/сек.;
- **--no-proxy** – не использовать прокси-сервер;
- **Q <N>k** – задать квоту (в килобайтах или мегабайтах) на закачку при скачивании файлов из списка;
- **--no-cache** – не использовать кеш сервера при закачке файлов.

В процессе загрузки файлов утилита **wget** отображает скорость передачи данных **(M/s)**, оставшийся процент докачки, тип загружаемых данных **([application/x-rpm])** и ответы сервера **(HTTP/1.1 200 OK)**, если был указан соответствующий ключ. В следующем примере приведена сессия загрузки файла с веб-сервера по протоколу **HTTP**:

```
--18:40:38--  http://172.16.0.5/yum/Server/Deployment_Guide-en-US-5.1.0-
11.noarch.rpm
Connecting to 172.16.0.5:80... connected.
HTTP request sent, awaiting response...
  HTTP/1.1 200 OK
  Date: Wed, 19 Aug 2009 06:19:47 GMT
  Server: Apache/2.2.3 (CentOS)
  Last-Modified: Fri, 23 Jan 2009 09:36:41 GMT
  ETag: "500c8-70e3a7-20547c40"
  Accept-Ranges: bytes
  Content-Length: 7398311
  Connection: close
  Content-Type: application/x-rpm
Length: 7398311 (7.1M) [application/x-rpm]
Saving to: `172.16.0.5/yum/Server/Deployment_Guide-en-US-5.1.0-11.noarch.rpm'
100%[======================================>] 7,398,311   2.84M/s   in 2.5s
18:40:40 (2.84 MB/s) - “172.16.0.5/yum/Server/Deployment_Guide-en-US-5.1.0-
11.noarch.rpm” saved [7398311/7398311]
```

В следующем листинге приведен пример загрузки файла с веб-сайта в указанный каталог:

```
# wget -P /opt ftp://ftp.debian.org/debian/README.mirrors.txt
```

Здесь ключом –P задается каталог для сохранения файла.

В следующем листинге приведен пример скачивания веб-сайта с глубиной следования по ссылкам 5 и журналированием всех действий в файл:

```
# wget -r -l5 http://mirrors.yandex.ru/ -o log
```

Ключ **–r** используется для рекурсивного скачивания, а ключ **–o** для указания журнального файла.

В следующем листинге приведен пример зеркалирования веб-сайта **www.example.com**:

```
# wget -m -np -R "*.tar" http://www.ccp14.ac.uk/mirror/
```

Здесь ключом **-m** задаются операции зеркалирования; ключи **-np** разрешают следовать только вниз по дереву каталогов при переходе по ссылкам внутри документа; ключ **–R "*.tar"** задает исключение всех файлов, попадающих под маску «***.tar**». Выполнив команду, начнется процесс выгрузки файлов сайта. По завершении процесса в каталоге, откуда была выполнена команда **wget**, будет создан новый подкаталог **www.example.com**, а под ним – каталог **mirror**, в котором будут находиться все выгруженные файлы.

14.4.3. Почтовый клиент Evolution

Современные программы чтения электронной почты обладают графическим интерфейсом и в своем развитии имеют тенденцию к унификации предлагаемых возможностей и интерфейса. Помимо доставки электронной почты, большинство из них позволяют сопровождать адресные книги и содержат в себе календари и другие средства коллективной работы.

Обычно почтовые клиенты предоставляют возможности чтения лент новостей, которые являются одним из самых старых способов распространения информации в сети Интернет и до сих пор могут служить одним из самых оперативных источников новостей.

Одним из самых популярных клиентов электронной почты в ОС Linux является **Evolution**.

Evolution – это приложение для рабочих групп (Groupware): в нем объединены электронная почта, календарь и адресная книга, благодаря чему связь и планирование задач оказываются в одном удобном пакете.

Почтовый клиент **Evolution** работает со стандартными протоколами почтовых серверов и может применяться практически в любом сетевом окружении. Пакет позволяет хранить почту на сервере (если взаимодействие с сервером осуществляется по протоколу IMAP), загружать в локальную систему сообщения (если взаимодействие с сервером осуществляется по протоколам IMAP(S) или POP(S)) или пользоваться спулингом почты на локальной системе, если на локальной системе работает собственный почтовый сервер. Кроме того, **Evolution** поддерживает возможность взаимодействия с серверами Microsoft **Exchange 2000/2003/2007** и **Novell GroupWise 6.5**, предоставляющими услуги электронной почты, календаря и адресной книги.

Запустить **Evolution** можно через соответствующий пункт в меню **Application (Приложения)** либо путем ввода команды **evolution** в командной строке. При этом должно появиться окно, подобное приведенному на рис. 14.3.

При первом запуске **Evolution** программа просит создать учетную запись электронной почты, для чего нужно ввести информацию о себе и доступе к электронной почте.

Обмен сообщениями при помощи Evolution

Чтобы начать работать с почтой **Evolution**, надо щелкнуть на кнопке **Inbox (Входящие)** или выбрать любую почтовую папку в панели папок. Окно представления почты делится на две части: в верхней располагается список сообщений,

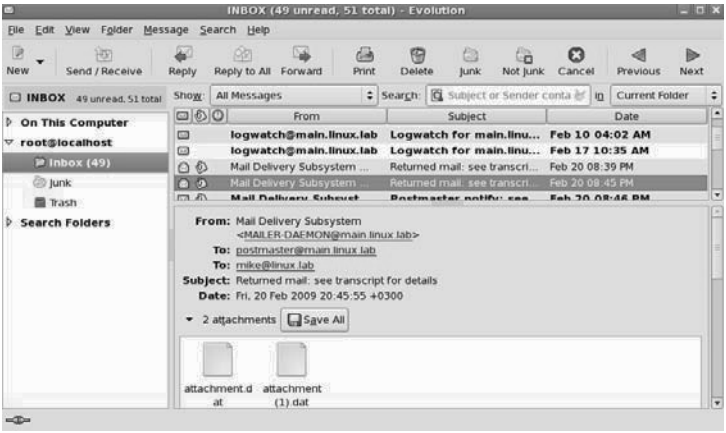


Рис. 14.3. Окно почтового клиента **Evolution**

а в нижней находится область чтения выбранного сообщения. Можно изменять соотношение частей, перетаскивая разделяющую их серую полосу, либо вообще скрыть область просмотра, выбрав пункт меню **View (Вид) ⇒Preview Pane (Панель предварительного просмотра)** или нажав комбинацию клавиш **<Ctrl+'>**.

В целом функции почты довольно просты: чтобы отправить почту, помещенную в очередь для отложенной отправки, и проверить поступление новой, нужно щелкнуть на кнопке **Send and Receive (Отправить/получить)**, а чтобы составить новое сообщение – щелкнуть на кнопке **New Message (Создать)** или одновременно нажать клавиши **<Shift+Ctrl+M>**.

После открытия письма необходимо заполнить поля **To (Кому)**, **Subject (Тема)** и в большом поле в нижней части окна ввести текст сообщения. Перед отправкой письма можно проверить орфографию, войдя в меню **Edit (Правка) ⇒ Spell Check Document (Проверить орфографию)** или нажав одновременно клавиши **<Shift+Ctrl+L>**. При необходимости можно присоединить к письму документ, нажав на кнопку **Attach (Вложить)**. Откроется окно, в котором нужно выбрать желаемый файл. Закончив составление письма, надо нажать на кнопку **Send (Отправить)** – письмо будет отправлено.

Что отличает **Evolution** от других почтовых программ, так это скорость поиска, мощь и простота фильтров, и уникальная характеристика **vFolders** – своего рода комбинация поиска с фильтрами.

Панель поиска располагается вверху списка сообщений. Для поиска в почте необходимо перейти в любую почтовую папку, выделить ту часть сообщения, в которой нужно выполнить поиск (тело сообщения, отправителя, все сообщение и т. д.), ввести в текстовое окно слово и нажать на клавишу **<Enter>**. Затем клиент **Evolution** выполнит предварительное индексирование почты и выдаст результат.

Фильтры добавляют в конце поиска операцию: при каждом получении почты **Evolution** осуществляет в новых сообщениях заданный поиск и выполняет действия, зависящие от его результатов. Чаще всего фильтры применяют для автома-

тического размещения сообщений в зависимости от отправителей и удаления сообщений, помеченных как спам. Чтобы создать фильтр, необходимо перейти в любое представление почты и открыть список фильтров, выбрав пункт меню **Tools (Сервис) ⇒Filters (Фильтры)**.

Календарь Evolution

Календарь **Evolution** предоставляет большую гибкость в создании и просмотре расписаний. Чтобы начать с ним работать, необходимо щелкнуть на кнопке **Calendar (Календарь)** в панели быстрого доступа. В результате будет отображено окно календаря (рис. 14.4).

Рис. 14.4. Окно календаря **Evolution**

В левой части окна выводится список имеющихся календарей, а в правой – содержимое выбранного календаря. Скрывать или показывать события, имеющиеся в отдельных календарях, можно, снимая или устанавливая флажок рядом с названием календаря в левой части окна.

Каждый набор событий выделяется своим цветом, чтобы избежать путаницы, что позволяет упростить восприятие списков событий в процессе их сравнения, когда необходимо скоординировать их или ликвидировать возникающие конфликты.

Календари могут относиться к нескольким категориям: **On this Computer (На этом компьютере)**, **On the Web (В сети)**, **Contacts (Контакты)** и, в зависимости от типа сервера для рабочих групп, категории **Exchange** или **GroupWise**. В самом начале у пользователя имеются всего два календаря: первый из них – это его личный календарь; второй – **Birthdays and Anniversaries (Дни рождения)** – содержит даты, которые были введены при добавлении контактов в адресную книгу.

Чтобы создать новый календарь, необходимо выбрать пункт меню **File (Файл) ⇒New (Создать) ⇒Calendar (Календарь)** и в открывшемся диалоге указать категорию создаваемого календаря. В случае выбора первой категории необходимо определить имя нового календаря и его цвет, после чего необходимо подтвердить

создание календаря, щелкнув на кнопке **ОК**. Для календаря в сети потребуется ввести те же данные, а также **URL-адрес** файла календаря и частоту обновления, с которой программа **Evolution** будет проверять наличие изменений.

Календари категорий **Contacts (Контакты)** создаются автоматически, нельзя иметь только по одному календарю из этих категорий. Чтобы создать новый календарь из категории **Exchange**, необходимо с помощью инструмента **Exchange** подписаться на доступ к папке календаря на сервере **Exchange**.

Адресная книга Evolution

Справочник контактов **Evolution** (рис. 14.5), или адресная книга, представляет собой, вероятно, наименее яркую часть пакета. Однако она тесно интегрирована со средствами электронной почты. Карточки контактов создаются щелчком на кнопке **New Contact (Новый контакт)** в представлении контактов, но можно создать карточку, щелкнув правой кнопкой на любом почтовом адресе в полученном сообщении электронной почты.

Если необходимо найти в адресной книге чей-то почтовый адрес, можно щелкнуть правой кнопкой на его карточке и выбрать отправку сообщения либо послать эту карточку кому-то другому, сделав два щелчка.

Чтобы посмотреть на справочник контактов, необходимо щелкнуть на кнопке **Contacts (Контакты)** в панели быстрого доступа или выбрать какую-нибудь папку с контактами в панели папок. Если необходимо упорядочить контакты, например список телефонов, то надо выбрать пункт меню **View (Вид) ⇒Current View (Текущий вид) ⇒Phone List (Список телефонов)**. Можно также вывести список по организациям, а не по именам.

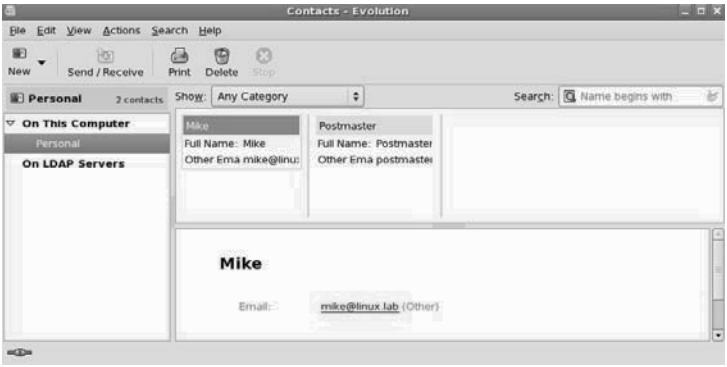


Рис. 14.5. Окно контактов **Evolution**

Приложения

Приложение 4.1. «Горячие» клавиши командного интерпретатора bash

Сочетание клавиш	Результат выполнения
Основные действия	
<TAB>	Автозавершение введенного имени файла или команд
<↑ >/<↓>	Просмотр предыдущей введенной команды
<↑ >/<↓>	Просмотр следующей введенной команды
<Ctrl+L>	Очистка экрана
Перемещение в пределах командной строки	
<Ctrl+A>	Перемещение курсора в начало строки
<Ctrl+E>	Перемещение курсора в конец строки
<Esc+B>	Перемещение курсора назад на одно слово
<Esc+F>	Перемещение курсора вперед на одно слово
Удаление и вставка текста в командной строке	
<Ctrl+K>	Удаление всего текста от текущей позиции курсора до конца строки
<Ctrl+U> (<Ctrl+X+ Backspace>)	Удаление всего текста от текущей позиции курсора до начала строки
<Ctrl+W>	Удаление слова, находящегося перед курсором
<Alt+D>	Удаление слова, находящегося после курсора
<Ctrl+Y>	Вставка последнего удаленного фрагмента текста
<Ctrl+C>	Удаление всей текущей строки
Модификация текста	
<Ctrl+D>	Удаление текущего символа в строке
<Ctrl+T>	Взаимная смена символа, находящегося до курсора, и символа, находящегося под курсором
<Esc+T> (<ALT+T>)	Взаимная смена двух слов, находящихся до или после текущего положения курсора
<Esc+U> (<ALT+U>)	Изменение всех символов, начиная с позиции курсора, до конца текущего слова на прописные
<Esc+L> (<ALT+L>)	Изменение всех символов, начиная с позиции курсора, до конца текущего слова на строчные
<Ctrl+X>, <Ctrl+E>	Запуск текстового редактора для редактирования командной строки, определенного в переменной $EDITOR. Если данная переменная не определена, то используется редактор Emacs
Поиск введенных команд	
<Ctrl+R>	Поиск ранее введенных команд
<Ctrl+G>	Выход из режима поиска

Приложение 4.2. Основные команды ОС Linux

Название команды	Описание
Команды управления файлами и каталогами	
cd	Изменение текущего рабочего каталога
chgrp	Изменение группы файла или каталога
chmod	Установка прав доступа к файлам или каталогам
chown	Установка владельца и группы на файл или каталог
cp	Копирование файлов или каталогов
file	Отображение типа файла
head	Вывод первых N строк файла или стандартного канала вывода
less	Постраничный просмотр текста или стандартного канала вывода
ln	Создание ярлыков (ссылок) на файл или каталог
ls	Просмотр содержимого целевого каталога и соответствующих атрибутов файлов и каталогов, содержащихся в целевом каталоге
mkdir	Создание каталога
mv	Перенос файла или каталога
pwd	Отображение текущего рабочего каталога
rm	Удаление файла или каталога
scp	Безопасное копирование файлов или каталогов
tail	Вывод последних N строк файла или стандартного канала вывода
Команды по работе с архивными файлами	
bunzip2	Распаковка архивных файлов формата bz2
bzip2	Создание архивных файлов формата bz2
cpio	Операции с архивными файлами формата cpio и tar
gunzip	Распаковка архивных файлов формата GNU Zip
gzip	Создание архивных файлов формата GNU Zip
tar	Операции с архивными файлами формата tar
Команды контроля состояния системы	
at	Запуск команды в заданное время
crontab	Редактирование файлов планировщика задач cron
date	Отображение или настройка текущей даты и времени
df	Отображение использования дискового пространства в разделах ОС
du	Отображение объема файлов или каталогов
env	Выполнение команды в измененном окружении командного интерпретатора
kill	Посылка заданного сигнала процессу (по умолчанию посылается сигнал SIGTERM, что в общем случае приводит к завершению процесса)
ps	Вывод информации по запущенным процессам
stty	Изменение и просмотр параметров терминала
who	Вывод информации по зарегистрированным пользователям
Команды обработки текста	
awk	Язык программирования обработки текста
cat	Вывод содержимого файлов в стандартный канал вывода
sed	Потоковый редактор текста

Название команды	Описание
sort	Сортировка строк в текстовом файле или стандартном канале вывода
tr	Преобразование текста
vi	Текстовый редактор
xargs	Составление команд из стандартного ввода и их последующее выполнение
Команды поиска файлов и каталогов	
locate	Быстрый поиск файлов
find	Поиск файлов в каталогах
grep/fgrep/ egrep	Поиск текста по заданному шаблону
Служебные информационные команды	
man	Работа со страницами справочного руководства ОС Linux
whatis	Быстрый поиск по страницам руководства
which	Вывод полного пути расположения исполнительного файла команды
info	Вывод справочной документации по командам ОС Linux в формате GNU texinfo
apropos	Поиск строковых совпадений в базе whatis

Приложение 4.3. Переменные окружения командного интерпретатора bash

Переменная	Значение
BASH	Путь к исполняемому файлу интерпретатора bash
BASH_ENV	Используется для запуска стартового файла конфигурации в случае неинтерактивного использования bash
EDITOR	Редактор по умолчанию
UID, EUID	Реальный и эффективный идентификаторы пользователя
HOME	Путь к домашнему каталогу пользователя
HOSTNAME	Имя хоста
OSTYPE	Тип ОС
HISTFILE	Путь к файлу истории команд
HISTSIZE	Количество команд в истории команд
LANG	Язык текущего сеанса
LINES	Число строк терминала
COLUMNS	Число столбцов терминала
PS1, PS2, PS3, PS4	Приглашения
SHELL	Используемый интерпретатор
TERM	Тип терминала
PWD	Текущий рабочий каталог
PATH	Список каталогов, разделенных двоеточиями, в которых производится «поиск» программ
USER	Имя текущего пользователя
LD_LIBRARY_PATH	Место «поиска» дополнительных библиотек
DISPLAY	Текущий дисплей графического сеанса X Window

Приложение 6.1. Перечень основных функций и команд gawk

Команда (функция)	Синтаксис и значение
#	Используется в сценариях awk в качестве символов комментария
and	**and(*выражение1*, *выражение2*)** Выполняет побитовую операцию AND (И) для значений *выражение1* и *выражение2*, которые должны соответствовать типу unsigned long языка C
asort	**asort(*исходный_массив* [,*целевой_массив*])** Сортирует *исходный_массив* по значениям элементов с выполнением необратимой замены индексов значениями от 1 до количества элементов массива. При указании *целевого_массива* выполняется копирование *исходного_массива* в *целевой_массив*, после чего *целевой_массив* сортируется, а *исходный_массив* отсается без изменений. Возвращает количество элементов в *исходном_массиве*
asorti	**asorti(*исходный_массив* [,*целевой_массив*])** Выполняется сортировка массива по индексам, а не по значениям
break	**break** Выполняет выход из цикла while, for или do
close	**close(*выражение*)** В большинстве реализаций AWK имеет не более 10 одновременно открытых файлов и одного канала. Данная функция позволяет закрыть файл или канал. В качестве *выражения* используется выражение, с помощью которого был открыт файл или канал. Это выражение должно быть посимвольно идентичным исходному
compl	**compl(*выражение*)** Возвращает побитовое дополнение *выражения*, которое должно соответствовать типу unsigned long языка C
continue	**continue** Начинает следующую итерацию цикла while, for или do
delete	**delete *массив*[*элемент*]** **delete *массив*** Удаляет *элемент* из *массива*. Второй вариант удаляет все элементы *массива*
do	**do** ***оператор*** **while (*выражение*)** Оператор цикла. Выполняет *оператор*, после чего определяет значение *выражения*, и если оно равно *true*, выполняет *оператор* еще раз
exit	**exit [*выражение*]** Выполняет выход из сценария, без чтения новых входных данных. Если задано действие END, то оно будет выполнено. В *выражении* задается код завершения сценария

Команда (функция)	Синтаксис и значение
for	**for(*начальное_выражение*; *тестовое_выражение*; *приращение*)** **for (*элемент* in *массив*)** ***оператор*** *Начальное_выражение* присваивает начальное значение переменной-счетчика. *Тестовое_выражение* представляет собой выражение, значение которого определяется каждый раз перед выполнением *оператора*. Если *тестовое выражение* имеет значение *false*, то происходит выход из цикла. *Приращение* используется для увеличения переменной-счетчика после каждой итерации. Второй вариант синтаксиса используется для чтения ассоциативных массивов. Для каждого *элемента массива* выполняется заданный *оператор*. Обращение к элементу массива выполняется в виде ***массив* [*элемент*]**
function	**function имя(список_параметров) {** **операторы** **}** Создает пользовательскую функцию с именем *имя*, состоящую из *операторов* awk, применяемых к заданному списку *параметров*
getline	**getline** **getline [переменная] [<файл]** **команда | getline [переменная]** Выполняет чтение следующей строки данных. Второй вариант синтаксиса выполняет чтение входных данных из *файла*. Третий вариант синтаксиса выполняет чтение входных данных *команды*. Все варианты синтаксиса считывают **по одной записи за раз**, и при каждом выполнении оператора выполняется чтение следующей записи входных данных. Запись помещается в переменную $0 и разделяется на поля; при этом устанавливаются значения NF, NR, FNR
gensub	**gsub(*регулярное_выражение, текст, n* [, *целевой_фрагмент*])**
gensub	Выполняет глобальную подстановку соответствий *регулярного_выражения* вместо *текста* в *целевом_фрагменте*. Если *целевой_фрагмент* не задан, используется $0. Если значение параметра n является числом, то выполняется замена каждого n-го соответствия. Если задано значение g, то выполняется глобальная подстановка. Возвращает новое текстовое значение. Оригинальное значение параметра целевой_фрагмент **не изменяется**
gsub	**gsub(*регулярное_выражение, текст,* [, *целевой_фрагмент*])** Глобальная подстановка
if	**if(условие)** **оператор 1** **[else** **оператор 2]** Стандартный условный оператор

Команда (функция)	Синтаксис и значение
printf	**print *выражение*** Выполняет форматированный вывод *выражения*
system	**system(*команда*)** Выполняет заданную системную *команду* и возвращает ее код завершения
while	**while (*условие*)** **оператор** Выполняет заданный *оператор*, пока *условие* имеет значение true. Группа из нескольких операторов заключается в фигурные скобки
if	**if (условие)** **оператор 1** **[else** **оператор 2]** Стандартный условный оператор. Условие может быть задано с использованием любых операторов сравнения <, <=,==,!=,>=,>, а также операторов вхождения в массив in и операторов сопоставления с образцом ~ и !~. Группа из нескольких операторов должна быть заключена в фигурные скобки. Сразу за else может следовать другой оператор if, создавая цепочку проверок
index	**index(*текст, фрагмент*)** Возвращает позицию *фрагмента* в заданном *тексте* или 0, если *фрагмент* отсутствует в *тексте*
int	**int(x)** Возвращает целое значение x путем удаления дробной части
length	**length([*аргумент*])** Возвращает длину аргумента или длину $0, если аргумент не задан
lshift	**lshift(*выражение*, n)** Возвращает результат сдвига выражения влево на n бит
match	**match(*текст*, *регулярное_выражение*)** **mathc(*текст*, *регулярное_выражение* [,массив])** Сопоставление с *обазцом*, заданным *регулярным_выражением*, в указанном *тексте* возвращает позицию в тексте, с которого начинается совпадение. Устанавливает в качестве значений RSTART и RLENGTH начало и длину совпадения соответственно и т. д. Если задан массив, то целиком совпавший текст помещается в элемент массива массив[0], текст, совпавший с первым выражением в скобках, – в массив[1], со вторым – в массив[2] и т. д.

Приложение 8.1. Основные команды редактора vim

Команда	Описание
Команды перемещения	
Перемещение по тексту	
w, b	Вперед, назад на «слово» (слова состоят из букв, цифр, символов подчеркивания)

Команда	Описание
W, B	Вперед, назад на «СЛОВО» (элементы слова разделяются только символами пробела, знаками табуляции и переходом на новую строку)
e	В конец слова
E	В конец СЛОВА
), (	В начало следующего, текущего предложения
}, {	В начало следующего, текущего абзаца
]], [[	В начало следующего, текущего раздела
Перемещение по строке	
0,$	В первую, последнюю позицию текущей строки
<Enter>	К первому непустому символу следующей строки
H	В верхнюю строку экрана
M	В строку посередине экрана
L	В последнюю строку экрана
nH	В n-ю строку от верхней строки экрана
nL	В n-ю строку от нижней строки экрана
:n	В n-ю строку файла
Перемещение по экрану	
<**Ctrl**+**F**>, <**Ctrl**+**B**>	Прокрутка вперед, назад на один экран
<**Ctrl**+**D**>, <**Ctrl**+**U**>	Прокрутка вниз, вверх на половину экрана
<**Ctrl**+**E**>, <**Ctrl**+**Y**>	Вывод еще одной строки внизу, вверху экрана
Поиск по тексту	
/образец	Прямой поиск образца. Завершается клавишей **Enter**
?образец	Обратный поиск образца. Завершается клавишей **Enter**
/	Повторение предыдущего поиска в конец файла
?	Повторение предыдущего поиска в начало файла
Команды вставки	
a	Ввод текста после курсора
A	Ввод текста в конец строки
C	Изменение текста всей строки
i	Ввод текста перед курсором
I	Ввод текста в начало строки
r	Изменение символа
R	Изменение текста
s	Подстановка символа
S	Подстановка всей строки
Esc	Выход из режима вставки
Команды редактирования	
cw	Изменение слова
cc	Изменение строки
dd	Удаление текущей строки
ndd	Удаление n строк
dw	Удаление слова
p	Вставка последнего удаленного или измененного текста после курсора
P	Вставка последнего удаленного или измененного текста перед курсором

Команда	Описание
yy	Копирование текущей строки
y$	Копирование от начала курсора до конца строки
ye	Копирование текста до конца слова
y$	Копирование от начала курсора до конца строки
J	Объединение текущей строки со следующей строкой
Команды сохранения и выхода	
:x	Выход из редактора vi (запись в файл выполняется только в случае его изменения)
:wq	Запись файла с последующим выходом
:w	Запись файла
:w файл	Сохранение копии текущего файла в новый файл
:n,mw файл	Запись с n по m строку в новый файл
:q!	Принудительный выход из редактора vi

Приложение 14.1. Основные клавиши в обозревателе Mozilla Firefox

Команда	Сочетание клавиш
Навигация	
Вперед	**Alt**+**стрелка вправо** **Shift**+**Backspace**
Домашняя страница	**Alt**+**Home**
Назад	**Alt**+**стрелка влево** **Backspace**
Обновить страницу	**F5**, **Ctrl**+**R**
Открыть файл	**Ctrl**+**O**
Принудительное обновление (не использовать кешированные данные)	**Ctrl**+**F5** **Ctrl**+**Shift**+**R**
Остановить	**Esc**
Текущая страница	
Информация о странице	**Ctrl**+**I**
Исходный код страницы	**Ctrl**+**U**
Перейти в конец страницы	**End**
Перейти в начало страницы	**Home**
Переместиться в предыдущий фрейм (на страницах с фреймами)	**Shift**+**F6**
Переместиться в следующий фрейм (на страницах с фреймами)	**F6**
Печать	**Ctrl**+**P**
Сохранить страницу как	**Ctrl**+**S**
Увеличить масштаб страницы	**Ctrl**++
Уменьшить масштаб страницы	**Ctrl**+–
Правка	
Вставить	**Ctrl**+**V**
Выделить все	**Ctrl**+**A**
Вырезать	**Ctrl**+**X**

Команда	Сочетание клавиш
Копировать	**Ctrl**+**C**
Отменить	**Ctrl**+**Z**
Повторить	**Ctrl**+**Y**
Удалить	**Delete**
Поиск	
Найти далее	**F3**, **Ctrl**+**G**
Найти на этой странице	**Ctrl**+**F**
Найти предыдущее совпадение	**Shift**+**F3**
Поиск в Интернете	**Ctrl**+**K**, **Ctrl**+**E**
Поиск ссылок по мере ввода	'
Поиск текста по мере ввода	/
Окна и вкладки	
Выбрать вкладку по ее порядковому номеру (от 1 до 8)	**Ctrl**+(**1**–**8**)
Выбрать последнюю вкладку	**Ctrl**+**9**
Закрыть вкладку	**Ctrl**+**W**, **Ctrl**+**F4**
Закрыть окно	**Ctrl**+**Shift**+**W**, **Alt**+**F4**
Новая вкладка	**Ctrl**+**T**
Новое окно	**Ctrl**+**N**
Открыть адрес в новой вкладке (из адресной строки или панели поиска)	**Alt**+**Enter**
Отменить закрытие вкладки	**Ctrl**+**Shift**+**T**
Перейти к предыдущей вкладке	**Ctrl**+**Shift**+**Tab** **Ctrl**+**Page Up**
Перейти к следующей вкладке	**Ctrl**+**Tab** **Ctrl**+**Page Down**
Переместить вкладку в начало (когда фокус на вкладке)	**Ctrl**+**Home**
Переместить вкладку в конец (когда фокус на вкладке)	**Ctrl**+**End**
Переместить вкладку влево (когда фокус на вкладке)	**Ctrl**+**стрелка влево** **Ctrl**+**стрелка вверх**
Переместить вкладку вправо (когда фокус на вкладке)	**Ctrl**+**стрелка вправо** **Ctrl**+**стрелка вниз**
Инструменты	
Включить/выключить режим активного курсора	**F7**
Добавить все вкладки в закладки	**Ctrl**+**Shift**+**D**
Добавить страницу в закладки	**Ctrl**+**D**
Журнал посещений	**Ctrl**+**H**
Загрузки	**Ctrl**+**J**
Закладки	**Ctrl**+**B**, **Ctrl**+**I**
Удалить личные данные	**Ctrl**+**Shift**+**Del**
Консоль ошибок	**Ctrl**+**Shift**+**Del**
Разное	
Выбор или управление поисковыми машинами	**Alt**+**стрелка вверх** **Alt**+**стрелка вниз**, **F4**
Выделить весь текст в панели адреса	**Alt**+**D**, **F6**, **Ctrl**+**L**
Дополнить адрес доменным суффиксом .com	**Ctrl**+**Enter**
Дополнить адрес доменным суффиксом .net	**Shift**+**Enter**
Дополнить адрес доменным суффиксом .org	**Ctrl**+**Shift**+**Enter**
Полноэкранный режим	**F11**
Удалить строку из списка автозаполнения адресов	**Del**

Практические работы

Описание виртуальных машин

DNS-имя	IP-адрес	Характеристики
rhel5.linux.lab	192.168.1.___	Виртуальная машина слушателя. ОС RHEL 5
main.linux.lab	192.168.1.210	Виртуальная машина преподавателя, являющаяся локальным репозиторием пакетов ОС Linux и FTP-сервером. Доступ к файлам осуществляется по протоколу FTP под учетной записью: Логин: **anonymous** Пароль: **user@linux.lab**

Практическая работа 2. Знакомство с пользовательским интерфейсом

Цель и результаты работы

Освоить навыки работы в графическом и текстовом интерфейсах пользователя. Понять идеологию работы с виртуальными консолями. Освоить базовые операции по работе с системой и графическим сервером **Xorg**.

Предварительные требования

Данная работа является вводной и не зависит от выполнения других практических работ.

Упражнение 2.1. Регистрация в системе. Работа с виртуальными консолями

Сценарий: вы впервые выполнили загрузку ОС Linux на своем персональном компьютере. Вам необходимо разобраться в меню загрузчика GRUB и освоить основные действия. Так как вы впервые выполняете загрузку ОС Linux, вам необходимо понять основные этапы загрузки системы. Поскольку ОС Linux является многопользовательской системой, вы решили убедиться в этом, используя виртуальные консоли и псевдотерминалы, а также определить зарегистрированных в системе пользователей.

Задачи	Описание действий
1. Настроить виртуальную машину на загрузку с локального жесткого диска	1. Запустить **VMware Player** и нажать клавишу <**F2**>. В результате должен отобразиться экран настройки **BIOS**. 2. В настройках **BIOS** необходимо перейти на вкладку **Boot** и первым пунктом в приведенном списке установить значение **Hard Drive**. 3. Сохранить сделанные изменения и выйти из настройки **BIOS**, нажав на клавишу <**F10**>
2. Выполнить загрузку ОС Linux на виртуальной машине и проанализировать этапы загрузки	1. До того, как исчезнет отсчет времени, необходимо нажать клавишу <**Space**> для входа в меню загрузчика **GRUB**. После нажатия клавиши <**Space**> будет отображено основное меню загрузчика **GRUB**, содержащее список установленных ядер ОС Linux. 2. Описать возможные действия пользователя в меню GRUB: Клавиша <**Enter**>:_____; Клавиша <**e**>:_____; Клавиша <**a**>:_____; Клавиша <**c**>:_____; Клавиша <**Esc**>:_____. 3. Выбрать единственную доступную версию ядра ОС Linux и нажать на клавишу <**e**>. 4. Проанализировать команду загрузчика GRUB **kernel** и зафиксировать ее аргументы: 1-й аргумент: _____; 2-й аргумент: _____; 3-й аргумент: _____. 5. Вернуться в главное меню загрузчика и загрузить систему. 6. Проанализировать последовательность загрузки системы. Для этого в процессе загрузки ОС нажимать комбинацию клавиш <**Ctrl**+**S**> (пауза), пока процесс загрузки ОС не остановится. Затем при помощи комбинации клавиш <**Shift**+**PgUp**> переместиться в начало вывода текстовых данных и проанализировать выходные данные. При помощи комбинации клавиш <**Shift**+**PgDn**>, перемещаясь вниз экрана, проанализировать выходные данные. Зафиксировать следующие параметры: Тип файловой системы загрузочного раздела (hd0,0): _____; Версия ядра Linux: _____; Частота процессора: _____; Объем доступной памяти: _____; Параметры и тип устройства scsi0: _____; Последовательность монтирования корневого раздела: _____;

Задачи	Описание действий
	Общая последовательность загрузки ОС: _____ _____ _____ _____. 7. При появлении предложения по запуску сервисов (**Starting service**) последовательно запустить все сервисы, нажимая на клавишу <**Y**> или <**Enter**>. 8. Дождаться появления начального экрана входа в систему
3. Освоить переключение между виртуальными консолями	1. Находясь в графическом экране входа в систему, нажать клавиши <**Ctrl**+**Alt**> и последовательно нажать клавиши <**Space**> и <**F1**> для перехода на 1-ю текстовую виртуальную консоль. 2. Ввести учетные данные пользователя **root**: login: **root** password: **P@ssw0rd** и выполнить вход в систему, нажав клавишу <**Enter**>. 3. Перейти на 2-ю виртуальную консоль, нажав клавиши <**Ctrl**+**Alt**>, и, последовательно нажав на клавиши <**Space**> и <**F2**>, выполнить повторный вход в систему от пользователя **root**. Запишите номер консоли, на которой последний раз выполнялся вход в систему: _____. 4. Перейти в графическую консоль, нажав клавиши <**Ctrl**+**Alt**>, и, последовательно нажав на клавиши <**Space**> и <**F7**>, выполнить повторный вход в систему от пользователя **root**. 5. Зафиксировать номер графической виртуальной консоли: _____
4. Запустить графическую программу **Terminal** и просмотреть текущие сеансы пользователей	1. Дождаться загрузки графического окружения и зайти в программу **Terminal**, нажав правой кнопкой мыши на рабочем столе и выбрав пункт **Open Terminal**. 2. В окне программы **Terminal** ввести команду **who** и проанализировать ее вывод: `# who` `root tty1 2009-04-28 21:05` `root tty2 2009-04-28 21:05` `root :0 2009-04-28 21:09` `root pts/1 2009-04-28 21:09 (:0.0)` Вывод:_____ _____ _____. 3. В окне программы **Terminal** перейти на 3-ю текстовую консоль при помощи команды **chvt**. 4. Выполнить вход в систему и ввести команду **last –x|head -20**. Определить предназначение команды **last**: _____ _____.

Задачи	Описание действий
	5. Выполнить переход на 8-ю, 9-ю и 10-ю виртуальные консоли. Сделать вывод о количестве доступных консолей: _____

Упражнение 2.2. Базовые операции с системой. Работа с интерфейсом командной строки

Сценарий: вы делаете свои первые шаги в работе с ОС Linux и хотите освоить базовые команды, такие как регистрация и завершение сеанса, перезагрузка, завершение работы в системе.

Задачи	Описание действий
1. Выполнить вход в систему в режим командной строки	1. Запустить виртуальную машину в **VMware Player** и дождаться появления окна ввода данных аутентификации. 2. Перейти в режим командной строки и выполнить вход в систему под пользователем **root**. 3. Определить параметры учетной записи пользователя при помощи команды **id** и зафиксировать следующее: Идентификатор пользователя (**uid**): _____; Основная группа пользователя:_____; Дополнительные группы пользователя: _____; Синтаксис команды **id**: _____
2. Посмотреть справку по вызову команды passwd и изменить пароль пользователя **root**	1. Посмотреть справку по команде **passwd** и ответить на следующие вопросы: Как определить статус пароля пользователя? _____ Как заблокировать учетную запись? _____ 2. Изменить пароль пользователя **root**. 3. Просмотреть статус пароля пользователя **ntp** и определить тип шифрования пароля: _____. 4. Разблокировать учетную запись пользователя **ntp** и повторно просмотреть статус пароля пользователя **ntp**. Сделать соответствующие выводы: _____. 5. Выполнить команду **finger -l ntp** и определить следующие параметры: Регистрационное имя пользователя: _____; Домашний каталог: _____; Имя пользователя: _____; Командная оболочка пользователя: _____. 6. Посмотреть справку по команде **nologin.** Выполнить попытку входа в систему под пользователем **ntp**. Сделать соответствующие выводы: _____
3. Освоить базовые операции с системой	1. Выполнить вход в систему на 4-ю текстовую консоль под пользователем **root**.

Задачи	Описание действий
	2. Переключиться на 5-ю текстовую консоль, выполнить вход в систему, а затем завершить сеанс при помощи команды **logout**. 3. Переключиться обратно на 4-ю текстовую консоль и просмотреть справку по команде **shutdown**. Какой ключ отвечает за перезагрузку системы? _____ Какой ключ отменяет ранее введенную команду? _____ Как изменить предупреждение о попытке завершения работы? _____ 4. Запланировать завершение работы через 5 минут и отменить его, используя команду **shutdown**, предварительно просмотрев справку по команде **shutdown**, выполнив команду **man shutdown**. 5. Выполнить вход в графическую консоль под пользователем **root** и перезагрузить систему при помощи команды **reboot**. 6. После перезагрузки выполнить вход в систему под пользователей **root** и запустить программу «**Terminal**». Выполнить команду **halt** и проанализировать дальнейшее поведение системы: _____ _____ _____

Упражнение 2.3. Работа в графическом режиме. Графический сервер Xorg

Сценарий: вам необходимо освоить работу в графическом режиме и понять принцип работы сервера **Xorg**. Для этих целей вы решили выполнить пробную настройку графического окружения, а также запустить несколько **X**-клиентов.

Задачи	Описание действий
1. Освоить работу в графическом окружении **GNOME**	1. Выполнить вход в графическую консоль под пользователем **root**. 2. Открыть программу **File Manager** и найти расположение файла **.bashrc**, используя встроенные средства поиска. 3. Открыть файл **.bashrc** в текстовом редакторе и добавить в него следующую строку: **alias mount_cd='mount -t iso9660 /dev/hdc /mnt'** Сохранить сделанные изменения. 4. Добавить на верхнюю панель рабочего стола апплет **System Monitor** и настроить его на отображение статистики по диску и сети. 5. Создать на нижней панели рабочего стола в левом углу область уведомлений и добавить в нее отображение часов.

Задачи	Описание действий
	6. Открыть программу **Terminal** и переключиться на вторую рабочую область, используя переключатель рабочих столов. 7. Изменить рисунок рабочего стола и расширение экрана (**1024×768**), предварительно добавив на верхнюю панель рабочего стола апплет запуска приложений (**Application Launcher**) и выполнив команду **system-config-display**. 8. Добавить на нижнюю панель ярлык запуска программы **Firefox**. 9. Создать правую панель рабочего стола и добавить на нее основное меню (**Custom Menu Bar**). Настроить данную панель на автоматическое скрытие с экрана. 10. Запустить программу **Terminal** и переключиться на 1-ю текстовую консоль
2. Понять принцип работы сервера **Xorg**	1. Выполнить вход в систему под пользователем **root**. 2. Выполнить команду **who** и ответить на следующие вопросы: На каком дисплее запущена программа **Terminal**? _____ Чем отличается консоль (**tty**) от псевдотерминала (**pts**)? _____ 3. Запустить второй сеанс сервера **Xorg** на 10-й виртуальной консоли в фоновом режиме, используя команду **X**. 4. Запустить программу **xterm** на втором экземпляре сервера Xorg: `# DISPLAY=__ xterm -geometry 800x600 &` Вместо подчеркивания вставить соответствующий номер дисплея. 5. Выполнить команду **who** и сделать соответствующий вывод относительно команды **xterm**: _____ _____. 6. Запустить программу **Firefox**, используя новый экземпляр сервера **Xorg**. 7. Перейти в графическую консоль и запустить дополнительный экземпляр сервера **Xorg**, используя команду **gdmflexyserver**. 8. Выполнить команду, находясь в текущей консоли: `# ps -ef|grep gdm` и определить, на каком терминале (**tty**) запустился дополнительный экземпляр сервера **Xorg**. Терминал, на котором запущен дополнительный экземпляр сервера **Xorg**: _____. 9. Перейти в новый сеанс менеджера дисплеев и выполнить вход в систему под пользователем **root**

Самостоятельные упражнения и дополнительные вопросы

1. Как можно просмотреть информацию о ранее выполненных перезапусках системы?
2. Опишите принцип работы системы X Window.
3. Какая виртуальная консоль используется по умолчанию для графического режима работы?
4. Каким образом в ОС Linux выполняется регистрация пользователей?

Практическая работа 3. Знакомство с файловой системой

Цель и результаты работы

Понять модель размещения файлов в ОС Linux и ориентироваться в иерархии файловой системы. Научиться определять типы файлов. Понимать разницу между относительными и абсолютными путями, а также научиться использовать подстановки имен файлов.

Предварительные требования

Данная работа подразумевает то, что вы уже имеете навыки работы в командной строке, полученные в предыдущей работе.

Упражнение 3.1. Иерархия файловой системы

Сценарий: осуществив загрузку ОС Linux в режим командной строки, вы решили выполнить несколько простых команд по работе с файлами и каталогами, с тем чтобы более подробно изучить особенности файловой системы ОС Linux.

Задачи	Описание действий
1. Определить домашний и рабочий каталоги пользователя	1. Выполнить вход в систему под пользователем **root** и перейти на 3-ю виртуальную консоль. 2. Определить текущий рабочий каталог пользователя, используя команду. 3. Перейти в корневой каталог и определить текущий рабочий каталог. 4. Перейти в домашний каталог пользователя. 5. Перейти на один уровень выше
2. Выполнить перемещение по каталогам файловой системы и отобразить их содержимое	1. Переместиться в корневой каталог и просмотреть его содержимое. 2. Переместиться в каталог **/var/log** и вернуться обратно в корневой каталог, используя относительный путь. 3. С помощью команды **ls** одновременно отобразить содержимое домашнего и корневого каталогов

Задачи	Описание действий
3. Определить типы файлов в различных каталогах файловой системы, используя команды **ls** и **file**	1. Определить типы файлов, содержащихся в каталоге **/dev**, используя команду **ls**. 2. Определить типы файлов, содержащихся в каталоге **/dev**, используя команду **file**. 3. Определить типы файлов **ifconfig** и **hosts**, содержащихся в каталогах **/sbin** и **/etc** соответственно. 4. Определить тип файла **/dev/sda1** при помощи команд **file** и **file -s**. Сделать соответствующие выводы: _____ _____. 5. Определить тип файла **/proc/cpuinfo** и сделать соответствующие выводы: _____ _____. 6. Определить тип файлов **/etc/init.d/yum-updatesd** и **/etc/rc.d/init.d/ yum-updatesd** при помощи команды **stat** и сделать соответствующий вывод: _____ _____
4. Выполнить операции по созданию, копированию, переименованию и удалению файлов	1. Создать каталог **/var/tmp/test/mytest** и переместиться в созданный каталог. 2. Создать в текущем рабочем каталоге, используя команду **touch**, три файла и определить их тип. 3. Скопировать в каталог **/var/tmp/test/mytest** файл **/etc/hosts** и вывести содержимое каталога **mytest** в расширенном формате и отсортированном порядке (по времени изменения). 4. Переименовать файл **1** в файл **hosts** и отобразить содержимое каталога. 5. Удалить файлы **2** и **3** из текущего рабочего каталога и просмотреть его содержимое. 6. Скопировать содержимое текущего рабочего каталога в домашний каталог пользователя **test**. 7. Удалить все содержимое каталога **/var/tmp/test/mytest**. 8. Удалить каталог **/var/tmp/test**

Упражнение 3.2. Монтирование файловых систем

Сценарий: в целях закрепления знаний по операциям монтирования файловых систем вы решили потренироваться, используя при этом имеющийся дистрибутив ОС Linux и удаленную файловую систему, доступную по **NFS**.

Задачи	Описание действий
1. Определить текущие смонтированные файловые системы	1. Открыть сеанс командной строки, запустив программу **Terminal**. 2. Определить смонтированные файловые системы, используя команду **mount**.

Задачи	Описание действий
	3. Проанализировать вывод команды **mount**, а именно: – смонтированное устройство _____; – точку монтирования _____; – тип файловой системы _____; – опции монтирования _____
2. Смонтировать файловую систему **ISO9660** в точку монтирования **/mnt**	1. Загрузить в устройство CD-ROM виртуальной машины образ ОС Linux (**rhel-5.3-server-i386-dvd.iso**), находящийся на хостовой машине. 2. Смонтировать файловую систему **ISO9660** в точку монтирования **/mnt** и просмотреть содержимое каталога **/mnt**. 3. Определить смонтированные файловые системы, используя команду **mount**
3. Смонтировать файловую систему, доступную по **NFS**, в точку монтирования /tmp/nfs	1. Создать каталог **/tmp/nfs**. 2. Смонтировать файловую систему **/inst** с удаленного хоста **192.168.1.210** в точку монтирования **/tmp/nfs**. 3. Определить смонтированные файловые системы, используя команду **mount**. 4. Создать в каталоге **/tmp/nfs** файл с произвольным именем и убедиться, что он создался
4. Размонтировать файловые системы, смонтированные в заданиях 2 и 3	1. Перейти в каталог **/mnt** и размонтировать устройство **/dev/hdc**. Сделать соответствующий вывод: _____ _____ _____. 2. Размонтировать файловые системы в точках монтирования **/mnt** и **/tmp/nfs**. 3. Определить смонтированные файловые системы, используя команду **mount**

Упражнение 3.3. Работа с группами файлов

Сценарий: вы планируете выполнить миграцию части ваших данных, расположенных на сервере ОС Linux, на новое оборудование. Перед вами стоит задача выполнить перенос только необходимых данных, расположенных в разных разделах файловой системы и относящихся к разным типам файлов.

Задачи	Описание действий
1. Определить расположение файлов для переноса и скопировать некоторые из них в указанный каталог	1. Выполнить вход в систему под пользователем **root** и перейти в текстовую виртуальную консоль. 2. Найти все скрытые файлы, присутствующие в домашнем каталоге пользователя, используя команду **ls** и шаблоны подстановки, и скопировать их в каталог **/opt/backup**. Отобразить при помощи команды **ls** все перенесенные файлы.

Задачи	Описание действий
	3. Найти все конфигурационные файлы в каталоге **/etc**, оканчивающиеся на строку **conf** и начинающиеся с символов «**y**», «**r**» или «**h**», при помощи утилиты **ls**, и скопировать их в каталог **/opt/backup/conf**. 4. Отобразить все файлы из каталога **/etc/rc.d/rc3.d**, начинающиеся с символов **S50**, **S51**, **S52** и т. д. до **S56** включительно. Скопировать найденные файлы в каталог **/opt/backup/rc3**. 5. В каталоге **/opt/backup** создать 50 подкаталогов с произвольным названием одной командой. 6. Скопировать все файлы из каталога **/usr/lib** в каталог **/opt/backup/lib**, имена которых не оканчиваются на символы **.a**, **.so** и **.la**. 7. Найти в каталоге все заголовочные файлы, название которых состоит из трех символов до точки, и скопировать их в каталог **/opt/backup/headers**. Поиск файлов выполнять при помощи классов **POSIX**

Самостоятельные упражнения и дополнительные вопросы

1. В каком файле содержится описание действий по монтированию файловых систем после загрузки?
2. Что описывает регулярное выражение?
3. Когда следует использовать регулярные выражения?
4. Опишите концепцию иерархии файловой системы ОС Linux.

Практическая работа 4. Основы работы с командной оболочкой

Цель и результаты работы

Цель данной работы заключается в получении практических навыков работы с командной оболочкой, а именно:

- определение переменных командного интерпретатора **bash**;
- поиск справочной информации по командам;
- использование перенаправления ввода-вывода;
- группировка команд;
- работа с историей команд;
- создание псевдонимов;
- настройка командного интерпретатора.

Упражнение 4.1. Работа с переменными окружения

Задачи	Описание действий
1. Определение текущих переменных окружения	1. Выполнить вход в систему под пользователем **root** и перейти в текстовую виртуальную консоль. 2. Определить текущие переменные окружения. 3. Вывести значение переменной **PATH** на терминал. 4. Добавить в переменную окружения **PATH** каталог **/opt/bin**. Скопировать в каталог **/opt/bin** исполняемый файл команды **date** и переименовать его в файл **new_date**. Перейти в каталог **/var** и выполнить команду **# new_date**. 5. Сделать переменную **PATH** обычной переменной и повторно выполнить команду **new_date**. Нажать два раза клавишу <**TAB**>. Сделать соответствующий вывод: _____. 6. Сделать переменную **PATH** переменной окружения и присвоить ей прежнее значение
2. Изменение переменных окружения и проверка результатов	1. Выполнить команду **date**. 2. Выполнить команду **date** в измененном окружении, в котором переменная **TZ** имеет значение **Asia/Novosibirsk**. Затем выполнить команду **date** еще раз. Сделать соответствующий вывод: _____ _____. 3. Присвоить переменной **my_var** значение 1, сделать ее переменной окружения и выполнить команду **set**. 4. Сделать переменную **my_var** обычной переменной и выполнить команду **set**. Сделать соответствующий вывод: _____ _____ _____ _____. 5. Создать простой сценарий, выполнив следующие команды: `# echo '#!/bin/bash' > sh.1; echo 'export VAR=1' >> sh.1` И просмотреть его содержимое при помощи команды `# cat sh.1` 6. Выполнить созданный сценарий и просмотреть значение переменной **VAR**. Сделать соответствующий вывод: _____ _____ _____

Упражнение 4.2. Поиск команд и справочной информации

Задачи	Описание действий
1. Поиск расположения команд и справочной информации по ним	1. Выполнить вход в систему под пользователем **root** и перейти в текстовую виртуальную консоль. 2. Определить текущие переменные окружения. 3. Найти расположение команды **echo**, используя команду **type –a**, и убедиться, что данная команда входит в переменное окружение. Сделать дополнительный вывод относительно команды **echo**: _____ _____ _____. 4. Просмотреть справку по команде **echo** в командной строке, не используя руководства **man** и **info**, и определить предназначение ключа **–n**: _____ _____ _____. 5. Найти информацию по встроенной в интерпретатор команде **echo** в руководстве **man**. 6. Найти команду, которая выполняет формирование и выполнение другой команды, используя в качестве строки для поиска строку “**construct argument lists and invoke utility**”. Посмотреть 1-й пример выполнения данной команды. 7. Выполнить команду из 1-го примера, указанного на странице **XARGS(P)** руководства **man**. Описать последовательность выполнения команд в командной строке: _____ _____. 8. Выполнить ту же команду, но без круглых скобок. Описать последовательность выполнения команд в командной строке: _____ _____
2. Работа с руководством **man**	1. Найти справку по конфигурационному файлу **passwd** в руководстве **man**. 2. Найти в руководстве **man** все команды, предназначенные для удаленного копирования файлов.

Задачи	Описание действий
	3. Установить максимальный приоритет для 5-й секции руководства **man** при просмотре страниц руководства. И проверить сделанные изменения в командной строке. 4. Определить, каким образом можно узнать расположение файла произвольного справочного руководства **man**. 5. Определить месторасположение файлов руководства **man** для команды **ldd**. Просмотреть конфигурационный файл руководства **man** и сделать соответствующие выводы относительно файлов страниц руководства: _____ _____ _____ _____. 6. Определить следующие параметры конфигурации страниц руководства **man** и заполнить следующие поля: Название и значение переменной, указывающей расположение страниц руководства: _____; Название и значение переменной, определяющие тип используемой программы-просмотрщика: _____
3. Работа с руководством **info**	1. Открыть справочное руководство **info** по команде **xargs**. 2. Определить названия предыдущей и последующей команд по отношению к команде **xargs** в руководстве **info** при помощи клавиш **n** и **p**. 3. Найти вторую гиперссылку в тексте руководства **info** (/) и переместиться по ней (<**TAB**>). 4. Просмотреть справку по командам работы с руководством **info**, используя клавиши **?** и <**Enter**>. Определить, какой комбинацией клавиш вызывается журнал просмотренных страниц руководства. 5. Определить опцию команды **info**, которая позволяет использовать для работы с руководством клавиши редактора vi. 6. Просмотреть руководство **info** по команде **info**. Используя клавишу **u**, вернуться в начальное меню

Упражнение 4.3. Настройка командного интерпретатора

Задачи	Описание действий
1. Найти файлы, являющиеся конфигурационными файлами для интерпретатора **bash**. Настроить псевдонимы команд	1. Выполнить вход в систему под пользователем **root** и перейти в текстовую виртуальную консоль. 2. Определить конфигурационные файлы **Bash**. 3. Определить, в каком из конфигурационных файлов интерпретатора присутствуют записи о псевдонимах команд.

Задачи	Описание действий
	4. Определить все используемые псевдонимы в настоящий момент. 5. Добавить псевдоним монтирования привода компакт-дисков, так чтобы монтирование выполнялось одной командой с одним-единственным параметром – точкой монтирования. 6. Смонтировать привод компакт-дисков при помощи созданного псевдонима, убедиться, что монтирование выполняется, а затем удалить псевдоним
2. Изменение настроек командного интерпретатора	1. Определить значение переменной первичного приглашения **bash**. 2. Найти справку по формату первичного приглашения **Bash** и изменить его (команда vi <путь_к_файлу>) в соответствующем конфигурационном файле так, чтобы после отображения рабочего каталога отображалось текущее время с точностью до секунд. Перезапустить сеанс пользователя. Изменить первичное приглашение, используя команду export, на первоначальный вариант. Сделать соответствующие выводы: _____ _____ _____
3. Поиск и выполнение команд из истории	1. Определить максимально допустимое количество запоминаемых команд. 2. Определить расположение файла с историей команд. 3. Вывести последние 5 выполненных команд, используя команду **fc**. 4. Вывести все команды, начиная с последней, начинающейся со слова man. 5. Выполнить команду с номером 10. 6. Найти ранее выполненную команду, начинающуюся со слов **cd ~**. Использовать возможность поиска в командной строке при помощи сочетаний клавиш <**Ctrl**+**r**>
4. Выполнить перенаправление ввода-вывода	1. Выполнить одновременное перенаправление вывода команды **date** на экран и в файл **date.out**. 2. Выполнить перенаправление из файла **date.out** на экран

Самостоятельные упражнения и дополнительные вопросы

1. Какое справочное руководство содержит больше информации?
2. Зачем нужно перенаправление?
3. Что такое переменное окружение?
4. Какие конфигурационные файлы интерпретатора bash считываются при открытии дополнительного сеанса пользователя?

Практическая работа 5. Работа с файлами и каталогами

Цель и результаты работы

Цель данной работы заключается в получении практических навыков работы с файлами и каталогами, а именно:

- умение выполнять поиск файлов согласно заданным критериям поиска;
- работать с группами файлов;
- использовать перенаправление ввода-вывода;
- работать с архивными файлами.

Упражнение 5.1. Ротация журнальных файлов

Сценарий: перед вами стоит задача выполнить ротацию[1] журнальных файлов различных сервисов, находящихся в каталоге **/var/log**, таким образом, чтобы файлы, время модификации которых составляет более 24 часов, архивировались, сжимались и переносились на заданный раздел (**/var/log/archivelog**) для последующего хранения. В процессе ротации команда должна создать отчет в файле (**/var/log/archivelog/report**), в котором должен содержаться список обработанных файлов.

Результатом работы должен быть каталог, включающий сжатый архивный файл, в котором содержатся системные журналы, а также отчет, содержащий список обработанных файлов.

Задачи	Описание действий
1. Найти расположение команды **find**, определить ее синтаксис	1. Выполнить вход в систему под пользователем **root** и переключиться в текстовый режим работы. 2. Определить расположение команды find, используя утилиту **locate**. Расположение команды find: _____. 3. Определить синтаксис вызова команды **find**. Синтаксис вызова: _____
2. Найти справочную информацию по использованию ключей **-exec** и **-mtime** команды **find**, используя руководства **man** и **info**	1. Вывести справочную информацию по команде **find**, используя руководство **man**, и найти описание использования ключей **-exec** и **-mtime**. Ключ **-exec** используется для: _____; Ключ **-mtime** используется для: _____. 2. Найти информацию по применению ключей **-mtime** и **-exec**, используя руководство **info**. В главном меню программы **info**, используя клавишу </> и введя в строку поиска слово **find** или пролистав экран клавишей <**Space**>, найти пункт * **find**:

[1] Под ротацией понимается процесс архивирования файлов.

Задачи	Описание действий
	(find)Invoking find, перейти по ссылке, нажав клавишу **Enter**. При необходимости возврата к предыдущей или следующей странице используйте клавиши <**p**> и <**n**> соответственно. Для перехода в начало страницы используйте клавишу <**b**>. Для возврата в главное меню используйте клавишу <**d**>. После того как необходимая информация будет найдена, выйти из программы **info**
3. Найти справочную информацию по командам архивирования и сжатия файлов	1. Найти справочную информацию по команде **gzip**, используя команду **man**. 2. Найти справочную информацию по команде **tar**, используя команду **man**, и обратить внимание на использование ключей **-f** и **-r**. Ключ **-f** используется для: _____; Ключ **-r** используется для: _____
4. Найти альтернативный способ использованию ключа **-exec** в команде **find**	1. В руководстве **info** найти раздел по команде **find** и в описании использования ключа **-exec** найти альтернативный способ выполнения команд над найденными файлами (использование команды **xargs)**. 2. Найти в руководстве **info** пример использования команды **xargs**
5. Выполнить ротацию журнальных файлов, находящихся в каталоге /**var**/**log**, согласно заданию	1. Создать каталог, куда будет производиться перенос файлов. 2. Найти файлы, используя команду **find**, время модификации которых – более чем 24 часа, сжать их, используя команду **gzip**, переместить их в каталог /**var**/**log**/**archivelog** и создать соответствующую запись в файле /**var**/**log**/**archivelog**/**report**: `# archive=/var/log/archivelog/logs-$(date +%d.%m.%g).tar` `# logdir=/var/log` `# report=/var/log/archivelog/report` `# find $logdir -type f -mtime +0 -exec tar -v -r -file=$archive {} \; >$report && gzip -v $archive` Команда в 1-й строке выполняет: _____ _____ _____; Команда в последней строке выполняет: _____ _____ _____. 3. Убедиться, что файл, содержащий журнальные файлы, успешно создался. 4. Удалить копии исходных файлов и проверить, что файлы были удалены. 5. Восстановить заархивированные файлы на прежние места

Задачи	Описание действий
6. Найти более совершенный метод ротации системных журналов ОС Linux	1. Используя команду **apropos**, найти команды, в описании которых содержится слово **rotate** (ротация). 2. Просмотреть справку по команде **logrotate** и определить ее основной конфигурационный файл. Абсолютный путь конфигурационного файла программы **logrotate**: _____; Относительный путь конфигурационного файла программы **logrotate** (относительно текущего рабочего каталога): _____

Упражнение 5.2. Просмотр файлов

Задачи	Описание действий
1. Просмотреть несколько файлов, определить необходимую информацию, выполнить указанные действия	1. Выполнить вход в систему под пользователем **root** и переключиться в текстовый режим работы. 2. Скопировать с виртуальной машины преподавателя файл **README**, используя клиента **lftp**. 3. Открыть командой **less** файлы **/etc/rc.d/rc.sysinit**, **/etc/fstab** и загруженный файл **README**. 4. Перейти к просмотру 3-го файла и переместиться на 3 страницы вперед, определить 3-й аргумент в команде **m4**. 3-й аргумент в команде **m4**: _____. 5. Перейти на одну страницу назад и определить аргумент команды **cd**. Аргумент команды **cd**: _____. 6. Выполнить поиск описания механизма access.db. Механизм access.db позволяет: _____ _____ _____ _____. 7. Переместиться в начало файла и найти 11-е вхождение строки +————+. Определить текст следующей строки. Текст следующей строки: _____. 8. Перейти к просмотру второго файла и настроить отображение номеров строк в этом файле. 9. Определить запись в 7-й строке данного файла. Запись в 7-й стоке: _____. 10. Перейти к 1-му файлу и переместиться в его конец. 11. Выполнить команду **date**, не выходя из файла. 12. Просмотреть название редактируемого файла. Название файла: _____. Размер текущего экрана в строках: _____.

Задачи	Описание действий
	Переместиться на 360-ю строку в файле. Сделать соответствующий вывод: _____ _____ _____.
	13. Перейти на 560-ю строку текущего файла и сделать пометку. Вернуться в начало файла, а затем к сделанной пометке

Практическая работа 6. Редактирование и обработка текстовых файлов

Цель и результаты работы

Цель данной работы заключается в получении практических навыков обработки текста. Данная работа позволит:

- научиться редактировать файлы в редакторе **vim**;
- научиться выводить текст файлов в требуемом формате;
- научиться сравнивать текстовые файлы;
- обрабатывать текст при помощи программы **awk**;
- обрабатывать текст при помощи программы **sed**.

Упражнение 6.1. Редактирование и обработка текстовых файлов

Задачи	Описание действий
1. Создать необходимые файлы в редакторе **vim**	1. Выполнить вход в систему под пользователем **root** и переключиться в текстовый режим работы. 2. Создать три файла следующего содержания. В скобках указано название файлов: **1-й файл (names):** name surname tel email ages Alex Ott (812)1233421 ott@gmail.com 25 Vasya Ivanov (495)6122331 vivanov@mail.ru 32 Petya Shestopalov 2344212 petr@redhat.com 28 Ilja Kudirov (820)1233421 ilk@pochta.ru 27 Файл содержит следующие поля: 1 – Имя, 2 – Фамилия, 3 – Телефон, 4 – Почтовый ящик, 5 – Возраст. Все поля разделены между собой табуляцией. **2-й файл (proff):** name surname proff Alex Ott doctor Vasya Ivanov teacher Petya Shestopalov engineer Ilja Kudirov freelancer

Задачи	Описание действий
	Файл содержит следующие поля: 1 – Имя, 2 – Фамилия, 3 – Профессия. Все поля разделены между собой табуляцией. **3-й файл (files):** name surname docs mp3 video Alex Ott 123 1 122 Vasya Ivanov 400 50 14 Petya Shestopalov 20 23 0 Ilja Kudirov 12 49 0 Файл содержит следующие поля: 1 – Имя, 2 – Фамилия, 3 – Количество документов, 4 – Количество музыкальных файлов, 5 – Количество видеофайлов. Все поля разделены между собой табуляцией. Все файлы создавать и редактировать в редакторе **vim**. Редактирование файлов выполнять одновременно в трех горизонтальных областях экрана, при необходимости используя визуальный режим, команды замены слов (**cw**, **dw**, **cW**, **d$**) и копирования/вставки текста (**y**, **p**). Переключение между областями экрана выполнять при помощи клавиш <**Ctrl**+**w**>
2. Освоить работу с основными утилитами форматирования и вывода текста	1. Вывести на экран содержимое трех созданных файлов при помощи команды **cat**, включая отображение символов табуляции и конца строки. 2. Исправить ошибки, допущенные в процессе редактирования файлов. Какими символами отображаются символы табуляции и конца строки? _____ 3. Настроить редактор **vim** на отображение символов табуляции и конца строки. Сделать эту настройку постоянной. 4. Вывести на экран отсортированный 3-й файл. Сортировку выполнять при помощи команды **sort**, сортируя по количеству музыкальных файлов в порядке убывания. Вывод производить без первой строки файла. 5. Удалить во всех файлах первую строку, используя редактор **vim**. 6. Определить, отсортирован ли 3-й файл, используя команду **sort**. Выполнить сортировку данного файла по умолчанию и вывести результат на экран. Затем повторно проверить, отсортирован ли файл. Сделать соответствующие выводы: _____ _____ _____ _____. 7. Вывести первые две строки отсортированного 1-го файла по 2-му полю.

Задачи	Описание действий
	8. Объединить файлы 1 и 2 в новый файл, используя команду **join**, и перенаправить вывод в новый файл **data**. До перенаправления убрать дублируемые поля, используя команду **cut**. 9. Отобразить список файлов в текущем рабочем каталоге в одну колонку, используя команды **ls** и **paste**
3. Освоить работу с утилитой **awk**	1. Вывести все поля из 1-го файла на экран и подсчитать их количество, используя при этом утилиту **awk**. Использовать для выполнения задачи встроенную команду **print** и переменную **NF**. 2. Вывести строку 1-го файла, в 3-м поле которой содержится код города 820. Код города задать как внешнюю переменную в команде **awk**. Для выполнения задачи использовать встроенный оператор **if**. 3. Написать простой awk-сценарий с названием **audio_count**, который подсчитывает суммарное количество аудиофайлов, указанных в 3-м файле. Результатом работы сценария должен быть вывод всех строк файла на экран, а также строки следующего вида: `Total number of audio files: N` 4. Модифицировать данный сценарий, чтобы отображалась только последняя строка. 5. Вывести имена всех людей из 1-го файла, возраст которых больше 27 лет, используя команды **cat** и **awk**
4. Освоить работу с утилитой **sed**	1. Отобразить записи из 1-го файла, в которых присутствует слово **Petya**, используя команду **sed**. 2. Удалить все записи из 2-го файла, в которых содержатся слова **engineer** и **teacher**. Входные данные для команды **sed** передавать со стандартного канала ввода. 3. Добавить после 1-й строки во все три файла пустую строку (**\n**), используя редактор **sed**. 4. Изменить почтовый ящик во 2-й записи первого файла на **vivanov@yandex.ru**. 5. Написать простой sed-сценарий с названием **new_line**, который вставляет после каждой строки пустую строку. Проверить данный сценарий на любом файле

Самостоятельные упражнения и дополнительные вопросы

1. Сколько режимов работы имеет редактор vi?
2. В каком файле хранятся персональные настройки редактора vim каждого пользователя?

3. С чем работает утилита awk?
4. Как применить изменения, сделанные редактором sed, непосредственно к файлу?

Практическая работа 7. Работа с регулярными выражениями

Цель и результаты работы

Цель данной работы заключается в получении практических навыков работы с утилитами семейства **grep** при использовании регулярных выражений.

Упражнение 7.1. Использование регулярных выражений с утилитами семейства grep

Задачи	Описание действий
1. Освоить работу с утилитами **grep**, **fgrep и egrep**	1. Выполнить вход в систему под пользователем **root** и переключиться в текстовый режим работы. 2. Найти все файлы в каталоге **/etc**, в состав которых входит слово **fgrep**. Вывести только имена найденных файлов. 3. Выполнить поиск всех пользователей, содержащихся в файле **/etc/passwd**, у которых в качестве командного интерпретатора указана программа **nologin**. Выделить найденные совпадения цветом. 4. Найти 10 последних записей в журнальном файле **/var/log/messages**, относящихся к текущей дате. 5. Составить шаблон поиска любого IP-адреса при помощи регулярных выражений и проверить его работоспособность при помощи утилиты **grep**. Здесь и далее используйте выделение совпавшего образца цветом. 6. Составить шаблон поиска любого MAC-адреса при помощи регулярных выражений и проверить его работоспособность при помощи утилиты **grep**. 7. Составить шаблон поиска почтовых адресов в формате **строка1@строка2.домен** для доменов **.com** и **.ru** при помощи регулярных выражений и проверить его работоспособность при помощи утилиты **grep**

Практическая работа 9. Работа с пользователями и группами

Цель и результаты работы

Цель данной работы заключается в получении практических навыков управления учетными записями пользователей, а также запуском программ от имени других пользователей. Данная работа позволит:

- научиться редактировать файлы в редакторе **vim**;
- научиться выводить текст файлов в требуемом формате;
- научиться сравнивать текстовые файлы;
- обрабатывать текст при помощи программы **awk**;
- обрабатывать текст при помощи программы **sed**.

Упражнение 9.1. Создание нового пользователя

Сценарий: в вашу организацию в отдел разработки поступает новый сотрудник (**Andrew Ivanov**) на временную должность, в которой он будет работать в течение месяца. Вам поставлена задача настроить учетную запись нового сотрудника, а также установить ограничения на запуск некоторых команд. До того, как сотрудник начнет работать, необходимо заблокировать его учетную запись. Настройка учетной записи пользователя должна быть выполнена, исходя из следующих требований:

- логин пользователя: **aivanov**;
- пароль пользователя: **P@ssw0rd**;
- максимальное время действия пароля пользователя: **14 дней**;
- группы, в которых состоит пользователь: **users**, **devel**;
- домашний каталог пользователя: **/home**;
- командный интерпретатор пользователя: **/bin/tcsh**;
- запрещаемые команды: разрешить запускать команду **rpm** только с ключом **-q**.

Задачи	Описание действий
1. Создать и настроить учетную запись пользователя согласно описанным требованиям	1. Выполнить вход в систему под пользователем **root** и переключиться в текстовый режим работы. 2. Определить текущие настройки, которые использует команда **useradd**: `# cat /etc/default/useradd` Каким образом присваиваются идентификаторы новым пользователям: _____. Где создаются домашние каталоги пользователей: _____. Какие ограничения будут иметь создаваемые учетные записи: _____. 3. Проверить наличие групп **users** и **devel**. В случае их отсутствия создать необходимые группы, используя команду **groupadd**. 4. Создать учетную запись пользователя согласно требованиям и проверить результат сделанных действий, используя команды **su**, **id** и **chage**. 5. Задать пароль для учетной записи пользователя **aivanov**,используя команду **passwd**
2. Наложить ограничение на запуск команд	1. Отредактировать файл **/etc/sudoers**, внеся в него запись, которая позволит запускать пользователю **aivanov** команду **rpm** только с ключом **-q**. Редактирование выполнять при помощи команды **visudo**.

Задачи	Описание действий
	2. Выполнить вход в консоль под пользователем **aivanov** и попробовать запустить команды **sudo rpm -qa** и **sudo rpm -h**. Сделать соответствующие выводы: _____ _____ _____.
3. Заблокировать учетную запись пользователя до момента выхода его на работу	1. Заблокировать учетную запись пользователя при помощи команды **passwd**

Самостоятельные упражнения и дополнительные вопросы

1. Где содержатся пароли локальных пользователей?
2. Каким образом можно выполнить команду из-под другого пользователя?
3. Где содержатся персональные конфигурационные файлы интерпретатора Bash?

Практическая работа 11. Знакомство с процессами

Цель и результаты работы

Цель данной работы заключается в получении практических навыков управления процессами. Данная работа позволит:

- научиться определять зависимости между процессами и их состояние;
- определять присутствующие в системе процессы и используемые ими ресурсы;
- научиться управлять процессами: остановка, перевод в фоновый режим работы, завершение работы.

Упражнение 11.1. Определение параметров запущенных процессов

Задачи	Описание действий
1. Определить запущенные процессы и их характеристики при помощи утилиты **ps**	1. Выполнить вход в систему под пользователем **root** и переключиться в текстовый режим работы. 2. Определить все присутствующие в системе процессы, используя команду **ps**. 3. Определить родительский процесс для процесса **crond**, используя команды **ps** и **pstree**. 4. Отобразить дерево процессов, используя команду **pstree**. В выводе отобразить аргументы команд.

Задачи	Описание действий
	5. Вывести на экран все процессы в формате **PID**, **PPID**, **Команда**, **Аргументы**, используя команду **ps**. 6. Определить состояние процессов **sshd** и **ps**. Состояние процесса **sshd**: _____; Состояние процесса **ps**: _____. 7. Определить первые пять идентификаторов процессов и названия соответствующих команд, которые больше всех используют ресурсы виртуальной памяти (**SIZE**), и вывести их на экран, используя команды **ps**, **sort** и др.
2. Определить параметры процессов при помощи утилиты **top**	1. Выполнить команду **top** и определить следующие параметры: Количество работающих процессов: _____; Аргументы процесса **sshd**: _____. 2. Добавить в вывод команды **top** поле **GROUP**. 3. Подсветить красным цветом все процессы и сохранить сделанные изменения в конфигурационном файле утилиты **top**. 4. Скрыть в выводе команды **top** все неработающие процессы
3. Выполнить операции по управлению процессами	1. Создать произвольный файл в редакторе vim. 2. В процессе редактирования остановить задачу, переведя процесс в состояние STOP. 3. Просмотреть текущие процессы пользователя, используя команду **ps -s**. Просмотреть текущие задачи при помощи команды **jobs** (с указанием идентификатора процесса). Сделать соответствующие выводы: _____ _____. 4. Перевести остановленную задачу в фоновый режим и запустить утилиту **top**. 5. Завершить работу команды **vim** посредством подачи сигнала **-9** и выйти из программы **top**. Выполнить команду **jobs**. Сделать соответствующие выводы: _____ _____

Самостоятельные упражнения и дополнительные вопросы

1. Что такое процесс?
2. Опишите принцип многозадачности в ОС Linux.
3. Как завершить работу процесса, зная его название?
4. Как определить идентификатор процесса?
5. Как определить размер памяти, занимаемый процессом?
6. Как определить состояние процесса? Чего ожидает процесс в состоянии SLEEP?

Практическая работа 12. Программирование в командной оболочке Bash

Цель и результаты работы

Цель данной работы заключается в получении практических навыков разработки сценариев командной оболочки **Bash**. Данная работа поможет вам:

- разобраться в структуре сценариев командного интерпретатора;
- научиться реализовывать базовые алгоритмы программирования;
- освоить способы решения различных задач при помощи сценариев;
- научиться отлаживать созданные сценарии.

Упражнение 12.1. Создание сценариев

Задачи	Описание действий
1. Создать и протестировать сценарии командного интерпретатора **Bash**	1. Выполнить вход в систему под пользователем **root** и переключиться в текстовый режим работы. Открыть редактор **vim**. 2. Создать сценарий, принимающий в качестве аргумента любой файл и выводящий содержимое указанного файла, вставляя в начало каждой строки файла ее номер в формате **номер**: строка... 3. Создать сценарий, проверяющий, содержится ли указанная команда в переменной окружения PATH. Результатом работы сценария должна быть строка вида: *команда* found in PATH или *команда* not found in PATH. В качестве разделителя полей использовать переменную окружения IFS (по умолчанию является символом пробела). Проверку содержания команды реализовать в виде функции. Сценарий должен принимать только один аргумент, не содержащий пустого значения, и выдавать сообщение об ошибке в случае нарушения данных условий. 4. Найти ошибки в следующем сценарии и исправить их: `#!/bin/bash` `# Копирует все файлы из текущего каталога` `# в каталог, указанный в командной строке.` `if [-z "$1"]  # Выход, если каталог назначения не задан.` `then` `  echo "Порядок использования: `basename $0 directory-to-copy-to"`

Задачи	Описание действий
	`` exit 65 fi ls . |xargs -i -t cp -rf ./{} $1 # Этот сценарий является точным эквивалентом # cp * $1 # если в именах файлов не содержатся пробельные символы. exit0 ```

Практическая работа 13. Работа с дисковым пространством

Цель и результаты работы

Цель данной работы заключается в получении практических навыков работы с файловой системой. Данная работа поможет вам научиться:

- создавать и форматировать разделы;
- создавать файловые системы и изменять их размер;
- определять свободное дисковое пространство;
- проверять файловую систему на наличие программных ошибок.

Упражнение 13.1. Создание и изменение параметров разделов

Задачи	Описание действий
1. Создать разделы и файловые системы на устройствах согласно требованиям	1. Выполнить вход в систему под пользователем **root** и переключиться в текстовый режим работы. 2. Отформатировать устройство **/dev/sdb**, создав на нем 3 раздела: 1-й раздел – тип раздела Linux; размер 50 Мб; 2-й раздел – тип раздела Linux; размер 20 Мб; 3-й раздел – тип раздела Linux Swap; размер 30 Мб. 3. Создать на отформатированных разделах файловые системы: 1-й раздел – ext3; 2-й раздел – ext2; 3-й раздел – swap. 4. Смонтировать первые два раздела в точки монтирования **mnt1** и **mnt2**. Раздел подкачки активизировать, используя команду **swapon**. Оценить сделанные изменения, используя команды **mount** и **free**
3. Выполнить копирование файлов на созданные разделы	1. Заполнить второй раздел наполовину, используя команду **dd if=/dev/zero off=/mnt2/file2 bs=____ count=____**. Контроль заполнения производить при помощи утилит **df** и **du**

Задачи	Описание действий
4. Изменить размеры указанных файловых систем без потери данных	1. Изменить размер 2-го раздела до 15 Мб. 2. Проверить, что размер 2-го раздела изменился и данные из этого раздела не были потеряны. Сделать соответствующие выводы: _____ _____ _____ _____

Самостоятельные упражнения и дополнительные вопросы

1. Опишите общий принцип создания и редактирования разделов.
2. Для чего используются метки разделов?
3. Как определить тип файловой системы и просмотреть ее параметры?
4. Перечислите основные моменты, на которые следует обращать внимание при создании файловой системы.

Практическая работа 14. Сетевые клиенты

Цель и результаты работы

Цель данной работы заключается в получении практических навыков работы с сетевыми возможностями ОС Linux. Данная работа поможет вам научиться:

- настраивать сетевые интерфейсы;
- определять открытые сетевые сокеты и процессы, которые их используют;
- подключаться к удаленным хостам по протоколу SSH.

Упражнение 14.1. Настройка и проверка сетевых параметров

Задачи	Описание действий
1. Настроить сетевые интерфейсы	1. Выполнить вход в систему под пользователем **root** и переключиться в текстовый режим работы. 2. Используя команду **ifconfig**, настроить интерфейс **eth3** следующим образом: IP: 10.0.0.4 MASK: 255.255.255.254 GATEWAY: 10.0.0.0.1/24 3. Проверить настройки интерфейса, выполнив посылку ICMP-запросов на шлюз по умолчанию
2. Определить активные сетевые сервисы	1. Используя команду **netstat**, определить: Открытые TCP-порты: _____; Количество принятых пакетов: _____; Таблицу маршрутизации: _____

3. Выполнить удаленное подключение и передачу данных	1. Используя команду **ssh**, выполнить подключение к локальному хосту через интерфейс **eth3**. 2. Выполнить копирование произвольных файлов на локальный хост при помощи команд **scr** и **sftp**. 3. Выполнить копирование произвольных файлов при помощи команды **lftp** на виртуальную машину преподавателя

Самостоятельные упражнения и дополнительные вопросы

1. Где содержатся настройки сетевых интерфейсов?
2. Как определить маршрут по умолчанию?
3. Откуда в выводе команды **netstat** берутся названия сервисов?

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЬЯНС-КНИГА» наложенным платежом, выслав открытку или письмо по почтовому адресу: **123242, Москва, а/я 20** или по электронному адресу: **orders@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в Internet-магазине: **www.alians-kniga.ru**.

Оптовые закупки: тел. **(495) 258-91-94, 258-91-95**; электронный адрес **books@alians-kniga.ru**.

Войтов Никита Михайлович

Основы работы с Linux

Учебный курс

Главный редактор *Мовчан Д. А.*
dm@dmk-press.ru
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Подписано в печать 13.07.2010. Формат 70×100 $^{1}/_{16}$.
Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 21,75. Тираж 1000 экз.
№

Web-сайт издательства: www.dmk-press.ru