МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ГОСУДАРСТВЕННОЕ ВЫСШЕЕ УЧЕБНОЕ ЗАВЕДЕНИЕ
"НАЦИОНАЛЬНЫЙ ГОРНЫЙ УНИВЕРСИТЕТ"

ГЕОЛОГОРАЗВЕДОЧНЫЙ ФАКУЛЬТЕТ
Кафедра геоинформационных систем

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Методические указания и задания
к лабораторным работам
для студентов направления подготовки
6.050101 Компьютерные науки

Днепропетровск
НГУ
2013

Операционные системы. Методические указания и задания к лабораторным работам для студентов направления подготовки 6.050101 Компьютерные науки / И.Н. Гаркуша. – Д. : Национальный горный университет, 2013. – 79 с.

Автор
І.М. Гаркуша, канд. техн. наук, доцент.

Затверджено до видання редакційною радою ДВНЗ "НГУ" (протокол № 1 від 29.03.2013) за поданням методичної комісії напряму підготовки 6.050101 "Комп'ютерні науки" (протокол № 4 від 18.01.2013).

Методичні матеріали призначено для використання іноземними студентами напряму підготовки 6.050101 Комп'ютерні науки під час виконання лабораторних робіт і підготовки до модульних контролів за їх результатами з нормативної дисципліни "Операційні системи".

Розглянуто теоретичні відомості про команди Unix/Linux-подібних операційних систем, командний процесор Bash, а також особливості використання потужного засобу керування в новітніх операційних середовищах Microsoft – Windows PowerShell.

Рекомендації до кожної лабораторної роботи містять практичну частину та опис звіту, який студенти подають викладачеві на захист.

Матеріали включають список додаткової літератури, а також додатки, зміст яких сприяє швидкому оволодінню прийомами роботи в Unix/Linux-подібних операційних системах.

Рекомендації орієнтовано на активізацію виконавчого етапу пізнавальної навчальної діяльності іноземних студентів.

Відповідальний за випуск завідувач кафедри геоінформаційних систем, д-р техн. наук, проф. Б.С. Бусигін.

С О Д Е Р Ж А Н И Е

ВВЕДЕНИЕ

Операционная система (ОС; operating system, OS) – комплекс управляющих и обрабатывающих программ, выступающих в роли интерфейса между устройствами вычислительной системы и прикладными программами. Она предназначена для управления устройствами и вычислительными процессами с учетом эффективного распределения вычислительных ресурсов, организации надежных вычислений и защиты данных пользователей. Разработчикам программного обеспечения, ОС позволяет абстрагироваться от деталей реализации и функционирования устройств, предоставляя определенный набор функций.

Безусловным лидером среди настольных ОС является семейство систем компании Microsoft – Windows. Однако спектр существующих современных ОС достаточно широк и, например, в области серверных технологий Internet по оценкам ряда аналитиков в 2012 году лидером являлась ОС CentOS на базе Linux-ядра. Также множество серверных систем построены на базе различных Unix-подобных ОС, например, свободной FreeBSD или корпоративных коммерческих системах, таких как, HP-UX или IBM AIX. Более того, в последние несколько лет получили широкое развитие свободно распространяемые Linux-совместимые ОС.

Второе место после Windows среди настольных ОС по популярности занимает ОС компании Apple – OS X. За ними идут различные Linux-совместимые ОС, наиболее известные из которых: Ubuntu, Fedora, Mint, Mageia, openSUSE, Debian, Arch.

Поскольку ситуация на мировом рынке ОС изменилась и является очевидным тот факт, что знание только одной ОС значительно сужает профессиональные границы деятельности будущих специалистов, является актуальным изучение альтернативных ОС и их инструментов.

Лабораторный практикум направлен на начальное изучение работы с Linux-подобной ОС, а также изучение передовых технологий Microsoft по управлению ОС Windows. В ходе практикума предлагается изучить необходимые команды Unix/Linux, командный процессор Bash. Рассмотрена тема использования мощного средства управления в новых ОС Microsoft – Windows PowerShell, пришедшего на смену устаревшим командным процессорам Microsoft.

В конце методических указаний представлен список рекомендуемой литературы для повышения уровня знаний студентов.

Практикум разделен на два модуля. Первый включает выполнение и защиту студентом 1 – 3 лабораторных работ, второй – 4 и 5 работ.

Для выполнения лабораторного практикума студент кроме теоретической составляющей каждой работы обязан владеть знаниями, полученными из лекционного курса.

Лабораторная работа № 1
Установка и настройка ОС в среде Oracle Virtual Box

Цель работы: приобрести навыки в установке и настройке Linux-совместимых ОС при помощи среды Oracle Virtual Box.

Теоретическая часть

Oracle Virtual Box является программным продуктом виртуализации для ОС MS Windows, Linux-совместимых и Unix-подобных.

Изначально программа разработана компанией Innotek, затем приобретена Sun Microsystems и после поглощения корпорацией Oracle успешно развивается и распространяется как Open Source Software под лицензией GNU GPL ver.2.

Виртуализация является общим термином, охватывающим абстракцию ресурсов для различных аспектов вычислений. Выделяют программную виртуализацию (динамическую трансляцию, паравиртуализацию), аппаратную виртуализацию, виртуализацию на уровне ОС.

Виртуальная машина Virtual Box создает окружение, представляемое для "гостевой" ОС, как аппаратное. Однако на самом деле это программное окружение эмулируется ПО хостовой (основной) системы. Такая эмуляция должна быть достаточно надежной, чтобы драйверы гостевой системы могли стабильно работать. При использовании паравиртуализации, виртуальная машина не эмулирует аппаратное обеспечение, а, вместо этого, предлагает использовать специальное API.

Рассмотрим процесс установки и настройки ОС в среде виртуальной машины Virtual Box (далее VBOX) на примере ОС RFRemix (русский ремикс Fedora, Russian Fedora Remix).

Примечание: рассматриваемая версия VBOX – 4.2.6-82870.

Fedora (ранее название Fedora Core) – это дистрибутив ОС GNU/Linux, спонсируемый компанией Red Hat и поддерживаемый сообществом. По сути, проект Fedora является полигоном для тестирования новых технологий, включаемых в дальнейшем в продукты Red Hat и других производителей.

Существует большое количество дистрибутивов основанных на Fedora. К ним относится и RFRemix – Russian Fedora. Дистрибутив подключается к репозиториям ПО, содержащим специфичные для российского пользователя программы, некоторые исправленные версии, не включенные в Fedora и несвободные (проприетарные) программы и драйверы. Например, различные мультимедийные кодеки.

Сайт проекта RFRemix:

http://russianfedora.ru/

Сайт проекта VBOX:

http://www.oracle.com/technetwork/server-storage/virtualbox/downloads/index.html

Для создания новой виртуальной машины в среде VBOX выполните следующие действия:

1. Запустите Oracle VM VirtualBox менеджер.
2. Откройте меню *Файл \ Настройки* и выберите слева общие настройки. Укажите путь к папке для виртуальных машин, которая будет всегда доступна. Нажмите на кнопку *OK*.
3. Нажмите на кнопку *Создать* в главном окне программы.
4. В качестве имени новой виртуальной машины укажите *Fedora17* и нажмите копку *Next*.
5. В качестве объема памяти укажите значение *1536 MB* и нажмите на кнопку *Next*.
6. Укажите опцию *Не подключать виртуальный жесткий диск* и нажмите на кнопку *Создать*. Подтвердите создание виртуальной машины без жесткого диска.
7. В списке виртуальных машин выберите *Fedora17* и нажмите на кнопку *Настроить*.
8. В списке настраиваемых опций нажмите на *Система* и уберите флажок с опций загрузки *Дискета* и *Жесткий диск*.
9. Включите опцию *IO APIC*. Перейдите к закладке *Процессор* и укажите в качестве количества 2 процессора.
10. Перейти к опциям *Дисплей* и указать в качестве объема видео памяти значение *64 MB*. Установить опцию включения 3D-ускорения.
11. Перейти к опциям *Носители*, нажать на иконке оптического диска. Из правого списка приводов указать *Первичный мастер IDE* и, нажав на кнопку справа, выбрать образ оптического диска. Например:

    C:\Install\VBOX\RFRemix-17-i686-Live-XFCE.iso

12. Выбрать пункт *Контроллер: IDE* и нажать на кнопку добавления привода оптических дисков. В диалоговом окне нажать на кнопку *Выбрать образ*. В качестве образа указать файл:

    C:\Install\VBOX\ AddonRFRemix17.iso

13. Нажать на кнопку *OK* окна настроек.

После выполнения основных настроек можно приступить к запуску виртуальной машины. Для этого укажите требуемую виртуальную машину (в данном примере *Fedora17*) и нажмите на кнопку *Запустить* главного окна менеджера или осуществите двойной щелчок курсором мыши на названии виртуальной машины.

Когда ОС RFRemix 17 с графической оболочкой XFCE загружена, экран будет выглядеть так, как представлен на рис. 1.1 (на рисунке представлено запущенное приложение *Терминал*).

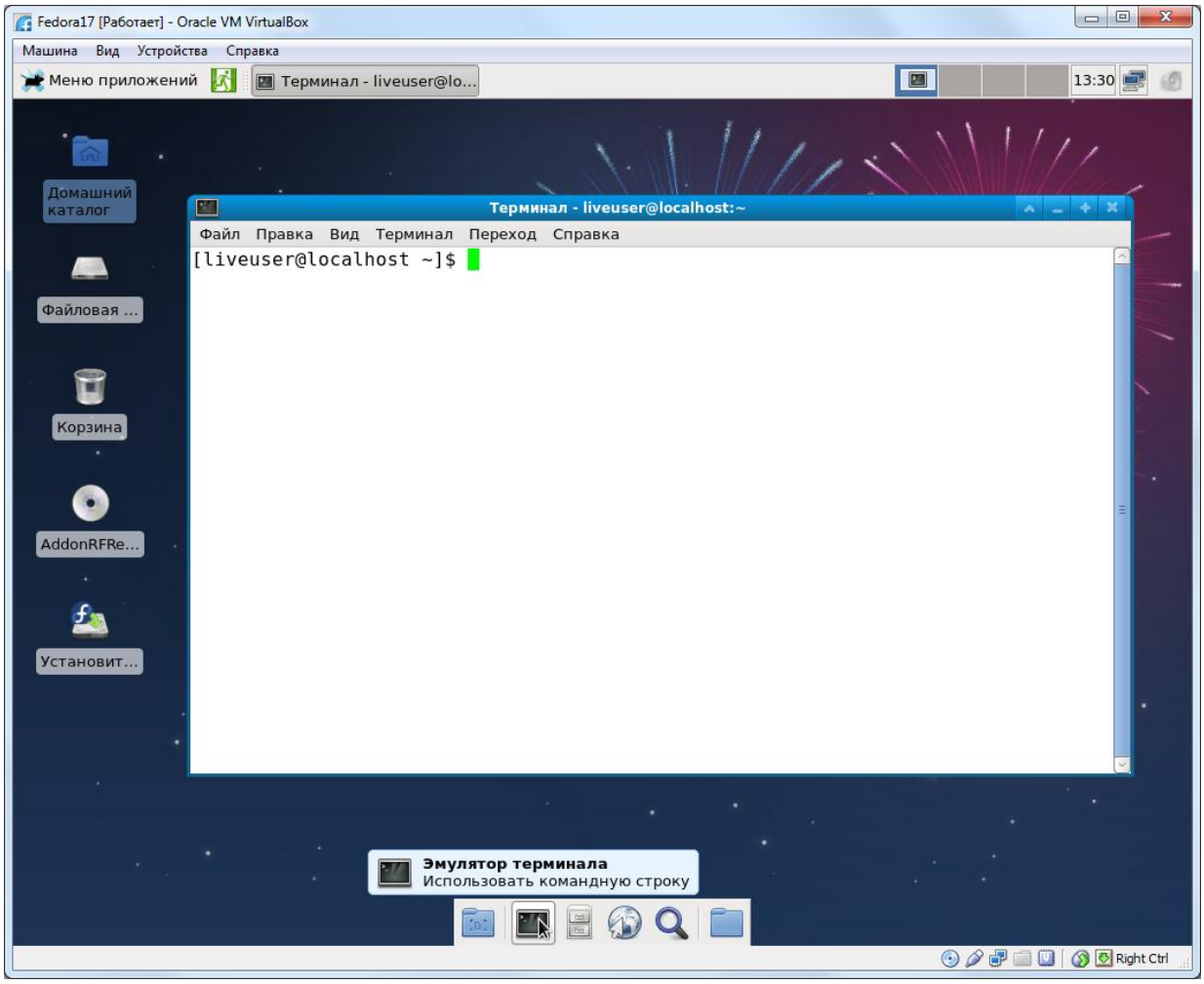

Рис. 1.1. Окно виртуальной машины с загруженной LiveCD-версией ОС RFRemix 17 XFCE

Интерфейс графической оболочки XFCE достаточно прост и интуитивно понятен. По умолчанию в верхней части экрана находится панель с меню приложений, кнопками управления и перечислением запущенных задач. В нижней части экрана находится вторая вспомогательная панель запуска (может отсутствовать в зависимости от настроек). Большую часть экрана занимает рабочий стол, с расположенными на нем значками запуска.

Узнать версию XFCE можно через меню приложений, выбрав пункт *Документация \ О Xfce 4*.

Для завершения работы в ОС выбирают из меню приложений пункт *Выйти \ Выключить*.

Состояние виртуальной машины может быть оперативно сохранено для последующего запуска. Для этого нужно нажать на кнопку закрытия окна виртуальной машины и в диалоге закрытия выбрать опцию *Сохранить состояние машины*.

Практическая часть

1. Выполните установку и настройку виртуальной машины с версией ОС RFRemix 17, как объясняется в теоретической части работы.

2. Измените настройку виртуальной машины таким образом, чтобы к контроллеру SATA был добавлен новый динамический виртуальный жесткий диск VDI (VirtualBox Disk Image). Для ускорения загрузки в дальнейшем с этого диска используйте в качестве места для хранения каталог C:\Install\VBOX. В качестве размера диска укажите 8 ГБ, в качестве имени диска используйте *Fedora17* (настройки выполнять в группе опций *Носители* при предварительно полностью выключенной виртуальной машине).

3. В группе опций *Система* установите опцию *Жесткий диск*.

4. Перезапустите виртуальную машину.

5. Используйте значок запуска с рабочего стола *Установить на жесткий диск*. При установке обязательно выполните следующие настройки.

Имя узла:	*host*
Часовой пояс:	*связан с городом Киев*
Системные часы:	*не используют UTC*
Пароль администратора:	***gisroot***

Схема размещения данных: *собственная* (стандартные разделы):

Точка монтирования:	/
Файловая система:	ext4
Размер:	6144 MB
Основной	

Точка монтирования:	/home
Файловая система:	ext4
Размер:	1024 MB
Основной	

Точка монтирования:	нет
Файловая система:	swap
Размер:	все доступное пространство
Основной	

6. После выполнения установки и перезагрузки виртуальной машины, приостановите ее начальную загрузку с LiveCD, нажав, например, клавишу перемещения курсора вниз. Отключите опцию загрузки с CD. Для этого закройте окно виртуальной машины без сохранения, перейдите к ее настройкам опций *Система* и отключите опцию загрузки с CD/DVD-ROM. В группе опций *Носители* выберите iso-образ ОС и в секции атрибутов укажите *Изъять диск из привода*. Повторите загрузку виртуальной машины.

7. Добавьте нового пользователя с именем *user* и паролем *user* (указать при этом, что он является *sudo*-пользователем). Проверьте правильность установленного системного времени и продолжите загрузку с подтверждением

об не отправке профиля установленной ОС.

8. После загрузки системы войдите под профилем пользователя *user*.

9. Установите дополнительное программное обеспечение с диска Addon. Для этого дважды щелкните по значку на рабочем столе *AddonRFRemix17*. В открывшемся файловом менеджере, в правой части где отображено содержимое Addon-диска, нажмите на свободном поле правую кнопку мыши и выберите пункт меню *Открыть терминал*. В терминале введите команды и дождитесь завершение установки пакетов программ (пакет *kernel-devel* устанавливается достаточно длительное время):

```
su
пароль администратора (пароль не отображается!)
sh  ./setup-c-devel.sh
```

10. После успешной установки дополнительных программ, перейдите к пункту меню в окне виртуальной машины *Устройства \ Установить дополнения гостевой ОС*. Когда появится в окне файлового менеджера новый диск, откройте его одним кликом мыши. Также как и в первом случае, откройте окно терминала для текущего каталога и выполните команды перехода в режим суперпользователя и запуска добавления дополнений для гостевой ОС:

```
su
пароль администратора
./VBoxLinuxAdditions.run
```

Дождитесь завершения процесса установки.

11. Перейдите в первую открытую консоль и выполнить команду exit, а затем команду:

```
sh  ./setup-share.sh
```

12. Закройте все окна в XFCE.

13. Выберите в окне виртуальной машины пункт меню *Устройства \ Общие папки* и с правой стороны нажмите на кнопку *Добавить общую папку*. Укажите путь к своей папке. В качестве имени папки укажите слово *share*. Включите опцию создания постоянной папки. Закройте диалог по кнопке *OK*.

14. Проверьте работу обмена данными через общую папку *share*, выполнив в окне терминала команду *./mountshare.sh*. Все содержимое папки должно отображаться в каталоге *share*. Для размонтирования *share* используйте вызов скрипта *./umountshare.sh* в окне терминала.

При защите работы представить установленную ОС на виртуальный жесткий диск, дать правильные ответы на следующие вопросы.

1. Что такое виртуальная машина?

2. Каким образом в VBOX осуществляется управление очередностью загрузки с носителей?

3. Какими способами в XFCE вызвать программу эмуляции терминала?

4. Как устанавливать расширения для гостевой ОС?

5. Как сохранять состояние виртуальной машины?

6. Как создавать собственное разбиение на разделы при установке ОС RFRemix 17?

Лабораторная работа № 2
Система помощи в Unix-подобных ОС

Цель работы: изучить основные возможности команды *man* по получению справочной информации в Unix-подобных ОС.

Теоретическая часть

Изучение теоретической части и выполнение задания, производятся в установленной, в ходе работы №1, ОС RFRemix. Описываемые в работе команды, вводятся в окне программы эмулятора терминала (см. работу №1).

Для получения детальной справочной информации по какой-либо команде (программе) Unix/Linux предназначена команда *man* (сокращение от “manual”). В простейшем виде ее вызов выглядит следующим образом:

man *команда/функция/программа*

где *команда/функция/программа* – название любой команды, функции или же программы ОС. Пробел между именем команды ОС и ее параметром (параметрами) обязателен.

Если справка по указанной информации установлена в одном из разделов справочной документации (как правило, это каталоги */usr/share/man*, */usr/local/man*, */usr/local/share/man*, */usr/man* и др.), то пользователю будет выдан результат.

При помощи следующей команды пользователь может всегда узнать, в каких каталогах файловой системы расположена документация команды *man*:

manpath

Если выводимая командой *man* информация не помещается в окне консоли, то, как правило, ее дальнейший вывод прерывается и может быть продолжен нажатием клавиши пробела. Выход из программы *man* осуществляется командой *q* (необходимо просто нажать на клавишу *q*).

Весь справочный материал разбит на разделы, порядок и названия которых различны для разных версий ОС. В табл. 2.1 перечислены традиционные разделы и их названия для двух основных ветвей Unix: BSD и System V. Часто во многих ОС эти разделы совпадают. Как правило, список разделов и их номера, а также всю информацию о команде *man* можно узнать, введя команду:

man man

При изучении справочной документации с использованием команды *man*, встречаются ссылки на другие команды, программы, описания файлов

конфигураций, функции языка С и разделы в следующем формате: *команда(номер)*. Например: getcwd(3). В данном случае это означает, что возможно получение справки по функции *getcwd* в разделе 3 следующим образом:

man *номер_раздела команда/функция/программа*

Например:

man 3 getcwd

Таблица 2.1

Название разделов справки

Содержание раздела	*BSD UNIX*	*UNIX System V*
Прикладные утилиты	1	1
Системные вызовы	2	2
Библиотечные функции	3	3
Специальные файлы, драйверы устройств и аппаратное обеспечение	4	7 или 4
Форматы различных конфигурационных и системных файлов и протоколов	5	4 или 5
Игры	6	6
Различные материалы, например, о типах файловых систем, определение типов данных и т.д.	7	5 или 7
Административные утилиты	8	8

У большинства программ/команд имеются ключи (параметры), которые можно указать во время запуска программ/команд. Так, весьма полезными ключами самой команды *man* являются ключи -f и -k. Многим программам с ключами соответствуют готовые командные файлы (командные скрипты или просто скрипты). Так, например, команде *man -f* соответствует скрипт *whatis*, команде *man -k* – скрипт *apropos*.

Команда *whatis* или *man -f* производит поиск краткого описания в базе данных *whatis* по каждому из указанных ключевых слов и отображения одной строки описания в окне терминала для каждого соответствия. Эта команда производит поиск только полностью совпадающих слов.

Если пользователю необходимо высветить все страницы справочного руководства, которые содержат указанное им слово, например, какую-либо команду, то для этих целей в Unix/Linux предназначена команда *apropos* или *man -k*. Команда производит поиск перечисленных строк в записях базы данных программы *whatis* и выдачу результатов в окно терминала. Команда *apropos* производит поиск строк, а не отдельных слов.

Примеры использования:

whatis chown
whatis passwd
man -f chown passwd
apropos passwd
man -k passwd single splash

Графической разновидность команды *man* является команда *xman* (по умолчанию может отсутствовать в системе).

Получить краткую справку по какой-либо команде Unix/Linux возможно при использовании ключа --help в составе нужной команды. Например:

```
mkdir --help
```

Команда *info* является GNU-средством просмотра гипертекста: отображает оперативную документацию, предварительно созданную из исходных файлов *Texinfo*. Информационные файлы (info-файлы) имеют иерархическую структуру, содержат меню и подтемы.

Примечание 2.1. В Linux большая часть программ предоставлена проектом GNU, развивающимся под управлением Фонда свободно распространяемого программного обеспечения. GNU – это рекурсивная аббревиатура: «GNU's not Unix» (GNU – это не Unix). Часто чтобы подчеркнуть эту особенность говорят не "Linux", а "GNU/Linux".

Примечание 2.2. Существуют и другие программные решения для отображения справочной документации, например, браузер *Konqueror*, входящий в состав графической оболочки KDE или программа gnome-help, входящая в состав другой графической оболочки GNOME. Например, следующая команда выведет в графическом окне справку по команде pwd:

```
gnome-help man:pwd
```

Кроме приведенных выше возможностей получения справки, пользователь также может использовать страницы документации, которые в большинстве Unix-подобных ОС располагаются в следующих каталогах:

```
/usr/doc
/usr/doc/Linux-HOWTOs
/usr/doc/Linux-mini-HOWTOs
/usr/doc/Linux-FAQs
/usr/src/linux/Documentation
/usr/local/doc
/usr/share/doc
/usr/share/info
/usr/local/share/doc
/usr/local/share/info
```

Возможно и другое расположение в системе. Кроме того, страницы руководств HOWTO (от английского *how to* – как сделать) и/или FAQ (Frequently Asked Question(s), часто задаваемые вопросы) могут отсутствовать. Это зависит от того, установил их суперпользователь (администратор, *root*) или нет.

Для чтения документации могут быть использованы различные программы-ридеры, в том числе, консольные. Например, можно использовать известную команду less для вывода информации из текстовых файлов на консоль с возможностью прокрутки текста:

```
less  /usr/share/doc/wine-core-1.5.16/README.ru
```

В случаях, когда пользователь имеет доступ к глобальной сети Internet, то весьма полезными для него может оказаться ряд специализированных ресурсов, список которых представлен в приложении А.

Практическая часть

1. Запустить VBOX, а в нем, установленную ранее, Linux ОС.
2. Войти в систему под именем/паролем: *user / user*.
3. В запущенной ОС, открыть окно терминала.
4. Ознакомиться с различными способами получения справочной информации на примерах, рассмотренных в теоретической части.

При защите работы дать правильные ответы на следующие вопросы:

1. Какие способы получения справки существуют в Unix/Linux?
2. Как получить справку, используя команду *man*?
3. Как получать справку из определенного раздела?
4. Какие изученные ключи команды *man* расширяют возможности поиска и каково назначение этих ключей?
5. Как получить короткую справку по команде?
6. В каких каталогах ОС Unix/Linux необходимо искать справочную документацию?

Лабораторная работа № 3
Основные консольные команды Unix-подобных ОС

Цель работы: изучить наиболее широко используемые консольные команды Unix-подобных ОС: *pwd*, *ls*, *ls -l*, *ls -la*, *mkdir*, *rm*, *mv*, *cd*, *vim*, *cat*, *cp*, *chmod*, *chown*, *chgrp*, *ps*, *top*, *free*, *df*, *who*, *uname*, *echo*, *less*.

Теоретическая часть

Краткое описание команд, используемых в лабораторной работе, представлено в табл. 3.1.

Таблица 3.1

Наиболее широко используемые
консольные команды Unix-подобных ОС

Команда	*Краткое описание*
pwd	Выводит полное имя текущего рабочего каталога
ls	Отображение списка файлов и каталогов
mkdir	Создание каталога
rm	Удаление файлов или каталогов
cd	Переход в заданный каталог (смена каталога)
cat	Конкатенация файлов или отображение их содержимого
cp	Копирование файлов/каталогов
mv	Перемещение/переименование файлов или каталогов
chmod	Изменение прав доступа к файлам/каталогам
chown	Изменение владельца-пользователя файла/каталога
chgrp	Изменение владельца-группы файла/каталога
ps	Информация о процессах в системе
top	Информация о процессах в системе, обновляемая в реальном времени
free	Вывод информации об используемой оперативной памяти
df	Вывод информации о распределении дискового пространства
who	Информация о пользователях, работающих в системе
uname	Вывод информации об узле (о компьютере и ОС)
echo	Вывод переменных окружения, вывод аргументов команды
vi	Консольный текстовый редактор
vim	Улучшенная версия консольного текстового редактора *vi*
nano	Крошечный консольный текстовый редактор
less	Просмотр файла

Для получения более детальной справки, например, по ключам команд, используйте команду *man* или приложения Б и В.

Консольный текстовый редактор vim

В состав Unix-подобных ОС, входит большое количество разнообразных

текстовых редакторов, в том числе консольных. Они особенно полезны в тех случаях, когда графическая подсистема по каким-либо причинам не доступна. Классическим консольным текстовым редактором является *vi*. Он обладает большим функциональным потенциалом. Все действия выполняются через комбинации клавиш и вводимые команды. Существует большое количество его улучшенных версий, одной из которых является редактор *vim*. Во многих Unix-подобных ОС команда *vi* является лишь ссылкой на одну из улучшенных версий редактора.

Рассмотрим основные приемы создания и редактирования текстовых файлов при помощи консольного редактора *vim*.

Для открытия текстового файла в редакторе вводят команду:

vim *./filename*

Если файл *filename* отсутствует в текущем каталоге, то будет открыт новый пустой файл. Указанный файл открывается в режиме просмотра. Для входа в режим вставки/редактирования текста в *vim* необходимо нажать на клавишу *i*.

После окончания редактирования нужно выйти из режима вставки, нажав клавишу *Esc*.

Операции выхода из редактора либо сохранения осуществляются из строки команд, расположенной в нижней части окна редактора (консоли). Для перехода в режим ввода команд, требуется ввести с клавиатуры символ “:” (двоеточие), за ним нужно вводить команды. Например, чтобы сохранить набранный текст и выйти из редактора вводится команда *wq* (команды сохранить и выйти):

:wq

Для выполнения команды нажимают клавишу *Enter*.

Примечание 3.1: все команды, начинающиеся с ":", должны завершаться нажатием клавиши *Enter*.

Если пользователь не делал никаких изменений, то выход из редактора осуществляется командой:

:q

Если по каким-либо причинам невозможно выйти из редактора в обычном режиме, то применяют принудительное завершение работы без сохранения изменений при помощи команды:

:q!

Если редактируемый файл требуется сохранить под другим именем, то вводят команду:

:w *имя_файла*

Например:

:w ./test.txt
:w /home/user/test.txt

Вариант с сохранением и последующим выходом:

:wq ./test.txt
:wq /home/user/test.txt

Если редактируемый файл имеет атрибут "только для чтения" и требуется принудительная запись в него информации, то применяют следующую команду:

:w!

Таблица 3.2

Часто используемые команды *vim* (*не все* (!) вводятся после символа ":")

Команда	*Описание*
yy	Копирование текущей строки в буфер
y<число>y	Копирование числа строк, начиная с текущей, в буфер
dd	Удаление строки в буфер редактора
d<число>d	Удаление числа строк в буфер редактора
dw	Удаление участка текста до конца слова в буфер редактора
x	Удаление символа под курсором
u	Отмена результата работы предыдущей команды
p	Вставка фрагмента текста из буфера редактора
shift+g	Быстрый переход в конец файла
/слово	Поиск слова в прямом направлении (в сторону конца файла)
?слово	Поиск слова в обратном направлении (к началу файла)
n	Повтор поиска в прямом направлении
shift+n	Повтор поиска в обратном направлении
:%s/было/стало/g	Быстрая замена слова "было" на слова "стало" во всем файле
ctrl+g	Информация о текущем положении курсора в файле
:!внешняя_команда	Выполнение внешней команды (например – **:!*pwd***)

Консольный текстовый редактор nano

Nano – консольный текстовый редактор для Unix-подобных ОС, основанный на библиотеке Curses, распространяемый под лицензией GNU GPL и являющийся свободным клоном текстового редактора Pico.

Для редактирования или создания файла с консоли вводят команду:

nano *имя_файла*

Для удобства использования в редакторе введены различные группы "горячих" клавиш. Описание некоторых из них, см. в табл. 3.3.

Таблица 3.3

Часто используемые комбинации клавиш редактора *nano*

Комбинация клавиш	*Описание*
Ctrl+g	Вызов системы помощи
Ctrl+x	Выход из редактора или из системы помощи
Ctrl+o	Сохранение файла
Ctrl+r	Загрузка файла и вставка его содержимого в текущий текст
Ctrl+w	Поиск слова в тексте
Esc+w	Повторить последний поиск
Ctrl+k	Вырезать текущую строку в буфер обмена редактора
Esc+6	Копировать текущую строку в буфер обмена редактора
Ctrl+u	Вставка строки текста из буфера обмена редактора
Esc+a	Установить метку начала выделения блока текста
Ctrl+c	Показать текущую позицию курсора
Esc+c	Постоянное отображение позиции курсора
Ctrl+y, Ctrl+v	Перемещение на предыдущую или следующую страницы
Esc+| , Esc+?	Перемещение в начало или в конец файла

Практическая часть

Используя материал теоретической части и команды получения справки, изученные в предыдущей работе, выполните следующие пункты практического задания.

1. Выполните команды: *pwd*, *ls*, *ls -l*, *ls -lh*, *ls -la*. Запротоколируйте и объясните полученный результат.

2. В домашнем каталоге создать каталог Text и сделать его текущим.

3. Используя редактор *vim*, создайте в каталоге Text небольшой текстовый файл приветствия с именем hello.txt.

4. При помощи команды *cat* вывести содержимое файла hello.txt в окно консоли. Запротоколируйте результат вывода команды *cat*.

5. Создайте копии файла hello.txt в текущем каталоге при помощи команд *cp* и *cat*, назвав новые файлы hello_cp.txt и hello_cat.txt соответственно.

Проверьте содержимое текущего каталога при помощи команды *ls -l*. Выведите в окно консоли содержимое новых файлов-копий. Запротоколируйте результаты.

6. Отмените у владельца-пользователя право на запись в файл hello_cp.txt, а всем остальным пользователям отмените право на запись в файл hello_cat.txt. Проверьте и запротоколируйте результат.

7. Последовательно при помощи одной команды допишите к концу файла hello_cat.txt содержимое всех файлов из каталога /etc, имена которых оканчиваются словом "release" или "version". Выведите содержимое файла hello_cat.txt на экран при помощи команды *less*. Запротоколируйте результат выполнения команд.

8. В каталоге Text создайте каталоги backup и temp.

9. Выполнить копирование одной командой сразу всех ранее созданных текстовых файлов в каталог backup. Проверьте и запротоколируйте результат.

10. Выполнить копирование каталога backup со всем его содержимым в каталог temp. Удалите файл hello_cp.txt из каталога backup, расположенного в каталоге temp. Проверьте и запротоколируйте результаты всех выполненных команд.

11. Выполните команду *ps -ef*, при этом перенаправьте ее вывод в файл ps_result.txt, который должен быть размещён в каталоге Text.

12. Ознакомьтесь с выводом команд *top*, *free*, *df* и объясните их.

13. Изучите ключи команд *who* и *uname*.

14. Переименуйте файл ./Text/hello.txt на ./Text/h.txt. Запротоколируйте выполнение команды.

15. Изучите и поясните в отчете справку по командам *echo*, *rm*, *chown* и *chgrp*, их назначение и приемы использования.

16. Используя редактор *nano*, создайте в каталоге Text файл, описывающий команды *chown* и *chgrp*.

Запротоколируйте для отчета выполненные операции и представьте описания команд *top*, *free*, *df*, *who*, *uname*, *echo*, *rm*, *chown* и *chgrp*.

Лабораторная работа № 4
Командный процессор Bash

Цель работы: изучить основные команды и инструкции командного процессора Bash. Овладеть техникой создания командных скриптов в среде Linux.

Теоретическая часть

Все командные оболочки, установленные в системе, указаны в файле */etc/shells*. Наиболее часто встречаются оболочки: bash, sh, csh, zsh, tcsh, ksh и др.

Каждому новому пользователю в Unix/Linux-подобных назначается командная оболочка (командный процессор) по умолчанию. Узнать, какая оболочка у конкретного пользователя можно при помощи команды (имя командного процессора указывается в конце выведенной строки-результата):

cat /etc/passwd | grep *имя_пользователя*

bash (Bourne again shell) – наиболее часто использующаяся командная оболочка (командный интерпретатор, командный процессор) Unix/Linux, является усовершенствованной и модернизированной вариацией командной оболочки Bourne shell (sh).

sh – ранняя командная оболочка Unix, разработанная Стивеном Борном из Bell Labs в 1978 году, выпущенная в составе Unix Version 7 и являющаяся стандартом де-факто для Unix-подобных ОС. В 1987 году она усовершенствована Брайаном Фоксом и существует под названием *bash*.

Проект *bash* развивается при поддержке Free Software Foundation в рамках проекта GNU. На момент 2012 года актуальной являлась версия 4.2. Узнать версию установленного bash можно введя команду:

echo $BASH_VERSION

Основное предназначение *bash* – выполнение команд пользователя. Вводимую команду *bash* воспринимает как некоторую программу, размещенную в одном из каталогов, указанных в переменной окружения PATH, или в указанном каталоге, или как внутреннюю команду. В противном случае выводится сообщение о невозможности выполнения команды.

Последовательность команд для автоматизации выполнения определенных задач можно группировать в сценарии. При запуске оболочки *bash* выполняется специальный сценарий *.bashrc*, находящийся в домашнем каталоге пользователя. В нем указывают команды, которые необходимо выполнить сразу после входа пользователя в систему. Файл является необязательным и может отсутствовать. Файл *.bash_history* хранит историю команд, вводимых пользователем в оболочке *bash*.

Команды интерпретатора *bash* могут быть объединены в сценарии (командные скрипты) и использованы совместно с другими скриптами или программами ОС. Для создания командных скриптов могут быть использованы любые редакторы простого текста, например, *vi*, *vim*, *nano*, *emacs*, *mcedit*, *leafpad*, *gedit*, *kwrite*, *kate*, *geany* или подобные. Небольшие сценарии могут быть набраны даже при помощи команды *cat* и перенаправления вывода с консоли в файл. Например:

cat > ./myscript.sh

Завершение создания файла в таком случае осуществляется нажатием комбинации клавиш Ctrl+D.

В командном сценарии *bash* первой строкой обязательно должна быть строка, не содержащая пробелов, следующего формата:

#!/bin/bash

За этой строкой следует текст кода сценария. Для запуска файла сценария ему необходимо установить право на выполнение. Например, если имеется файл скрипта *myscript.sh*, тогда для обеспечения его выполнения вводят команду:

chmod +x ./myscript.sh

Команда устанавливает право на выполнение для указанного файла. После этого можно вызвать командный скрипт:

./myscript.sh

Примечание 4.1: установка атрибута на выполнение возможна в случаях, когда файл расположен в разделах Linux файловых систем, например, таких как: ext2, ext3, ext4, ReiserFS и им подобным. Однако в случаях с файловыми системами Microsoft (FAT, FA32, NTFS), файлы уже обладают атрибутом на выполнение, но их выполнение из этих файловых систем, как правило, невозможно. Поэтому, если исполняемый файл находится не на Linux файловой системе, то его следует перенести на раздел Linux и только после этого выполнять.

Можно выполнить скрипт и без права на выполнение. Для этого надо вызвать напрямую интерпретатор и передать ему в качестве параметра имя файла командного скрипта:

bash ./myscript.sh

или

sh ./myscript.sh

Часто в Linux-подобных ОС команда *sh* является символической ссылкой (symbolic link) на *bash*. Поэтому вызов *sh* будет эквивалентен вызову *bash*. Проверить, является ли *sh* просто ссылкой на *bash* или это настоящий командный процессор *sh* можно при помощи команды:

ls -l /bin/sh

Комментарии в *bash*-скриптах начинаются с символа # и пробела.

Если требуется осуществить вывод какого-либо сообщения (строки) на консоль, то используют команду *echo*.

Для объявления переменной (используется только внутри файла-скрипта) используется конструкция: *переменная=значение*. Пробельные символы между переменной и символом присвоения и между символом присвоения и значением, отсутствуют.

Пример командного скрипта:

```
#!/bin/bash
#  This is my test bash-script
MYPROG=$HOME/myprograms/myprog1
echo 'HOME directory: '$HOME
echo 'My program:'
echo $MYPROG
echo 'Enter program name:'
read MYPROG
echo $MYPROG
CURDIR=`pwd`
echo 'Current directory: '$CURDIR
```

В примере показаны некоторые команды bash и работа с переменными.

Команда *read* осуществляет ввод с клавиатуры. Конструкция *переменная=`команда`* устанавливает в качестве значения вывод указанной в наклонных кавычках ` ` команды.

В Linux существуют специальные системные переменные окружения, содержащие служебные данные, например:

HOME – домашний каталог пользователя, запустившего сценарий;

PATH – пути поиска программ для запуска;

PWD – текущий каталог;

UID – ID пользователя, запустившего сценарий;

USER – имя пользователя, запустившего сценарий;

RANDOM – случайное число в диапазоне от 0 до 32767;

LANG – текущая языковая настройка терминала;

HOSTNAME – имя компьютера;

HISTFILE – имя файла журнала команд, с которым происходит работа;

PS1 – первичное приглашение командной строки; например:

[\u@\h \W]\$

Результат:

[igor@Fedora ~]$

где u – имя пользователя; h – имя хоста; W – рабочий каталог.

PS2 – вторичное приглашение новой строки для незаконченной команды, например:

>

Регистр вводимых переменных имеет значение. Например, следующие команды выводят на консоль совершенно разные результаты:

```
echo $PATH
echo $path
```

Для установки собственной переменной окружения используется команда export следующего вида (в этом случае переменная будет доступна всем дочерним процессам из bash):

```
export  MYVAR
```

или

```
export  MYVAR=значение
```

Обработка параметров командных файлов осуществляется при помощи обращения к специальным переменным:

$0 – содержит имя сценария;

$n – содержит значение параметра, где n – номер параметра, например, $1 – первый параметр;

$# – содержит количество переданных скрипту параметров командной строки.

Инструкции и команды bash

В *bash* доступны два условных оператора – *if* и *case*. Синтаксис:

```
if  условие_1  then
        команды_1
elif  условие_2  then
        команды_2
...
elif  условие_N  then
        команды_N
else
```

```
        команды_N+1
fi
```

Количество блоков *elif* не ограничено. Формат условий (символы пробелов обязательны; завершающая точка с запятой – обязательна!):

```
[ переменная выражение значение|переменная ];
```

где выражение:

```
= =          – равно
!=           – не равно
-lt          – меньше
-gt          – больше
-le          – меньше или равно
-ge          – больше или равно
-eq          – равно
-e файл      – условие истинно, если файл существует
-f файл      – условие истинно, если файл существует и он регулярный
-d каталог   – условие истинно, если каталог существует
-x файл      – условие истинно, если файл является исполнимым
!            – отрицание условия
&&           – логическое И
||           – логическое ИЛИ
```

Примеры условий:

```
#  переменная MAX равна 30
[  MAX==30  ]
#  переменная MAX не равна 30
[  MAX!=30  ]
#  MAX меньше 30
[  $MAX  -lt  30  ]
#  MAX меньше N
[  $MAX  -lt  $N  ]
```

Полный пример конструкции оператора if:

```
MAX=10
N=5
if [ $MAX -lt $N ]; then
        echo 'MAX < N'
else
        echo 'MAX > N'
fi
```

```
if [ -f ./"$1" ]; then
        echo ' '$1' - This is regular file!'
else
        echo ' '$1' - This is NOT regular file!'
fi
A=10
B=20
if [ $A -gt 0 ] && [ $B -gt 0 ]; then
        echo 'А и Б сидели на трубе.'
fi
```

Оператор *case* имеет следующий синтаксис:

```
case  переменная  in
        значение_1) команды_1 ;;
        …
        значение_N) команды_N ;;
        *) команды_по_умолчанию ;;
esac
```

Значение указанной переменной по очереди сравниваются с приведенными значениями (значение_1, …, значение_N). Если есть совпадение, то будут выполнены команды, соответствующие значению. Если совпадений нет, то будут выполнены команды по умолчанию. Пример:

```
case "$1" in
        start)
           echo "Запускаем процесс"
           ;;
         stop)
           echo "Останавливаем процесс"
           ;;
         *)
           echo "Используйте: $0 {start|stop}"
           #  Покидаем командный файл
           exit 1
           ;;
esac
```

Циклические операции в *bash* реализуются при помощи операторов: *for*, *while*, *until*, *select*. Синтаксис *for*:

```
for переменная in список ;
do
   команды ;
done
```

или

```
for (( выражение_1 ; выражение_2 ; выражение_3 )) ; do
        команды ;
done
```

Примеры:

```
for n in 1 2 3 4 5 ;
do
        echo "Start loop iteration -->" ;
        echo $n ;
done
for (( i=0; i<10; i++ )); do
        echo "Loop iteration #"$i ;
done
```

Синтаксис *while*:

```
while условие; do
        команды ;
done
```

Пример:

```
i=0
while [ $i -lt 10 ]; do
        echo "Loop iteration # $i" ;
        i=$(( $i+1 ));
done
```

Оператор $((арифметическое_выражение)) возвращает результат вычисления выражения.

В выражениях могут использоваться различные операции, по синтаксису соответствующие языку программирования С. Например:

++ операция инкремент;

– – операция декремент;

+= операция сложение с присвоением;

% операция взятия остатка от деления;

и т.п.

Например, цикл, тело которого выполняется десять раз, начиная от нуля, и

при этом на консоль будет выведена строка о номере итерации только при условии, что остаток от деления номера итерации на 2, будет равен нулю:

```
for (( i=0; i<10; i++ )); do
        if [ $((i % 2)) -eq 0 ]; then
                echo "Loop iteration #"$i ;
        else
                echo "---" ;
        fi
done
```

Команда *let* позволяет выполнять целочисленные операции. Например:

```
let "count=0" "i=i+1" "num=4"
echo $count
echo $i
echo $num
```

В bash встроены многие команды, сведения о которых можно получить при помощи команды *man bash*.

Встроенная команда *exit* с кодом возврата выполняет завершение командного скрипта. Например:

```
exit 1
```

Команда *test* с условием возвращает 0 или 1 в зависимости от выполнения условия. Например, проверка существования файла и его выполнение в случае успеха:

```
test -e ./myscript.sh && ./myscript.sh
```

Псевдонимы команд

Команда *alias* без аргументов выводит на экран список определенных псевдонимов. Псевдонимы могут быть полезны, когда пользователь часто выполняет какую-нибудь длинную команду или же нужно модифицировать выполнение команды по умолчанию, например, ввести предварительные запросы. Также, если определенная команда выполняется часто с заданным ключом, ей также можно назначить псевдоним. Это происходит по следующему синтаксису:

alias *псевдоним=команда_с_аргументами*

Например, введем новые две команды: myps и killp, которые позволят вывести подробно информацию обо всех процессах текущего пользователя (myps) и удалить процесс из памяти, посылая ему сигнал SIGKILL (killp).

Псевдонимы опишем в конце файла *.bash_profile*:

```
alias myps='ps -ef | grep $USER | more'
alias killp='kill -s SIGKILL '
```

Как правило, уже предопределенные в системе псевдонимы обычно описаны в файлах системного каталога /etc/profile.d или (реже) в файле /etc/profile.

Определенные псевдонимы будут доступны к использованию сразу после нового входа в систему пользователя.

Выполнение нескольких команд

При формировании команды можно использовать одновременно несколько команд, применяя специальные разделители.

Если между командами используется символ '|' , то это говорит о формировании так называемого конвейера команд, когда второй команде в качестве параметров передается результат первой команды, или на вход третьей команды передается результат второй, обрабатывающий в качестве параметра результат первой и т.д. Например, следующий конвейер позволяет вывести на консоль информацию обо всех процессах, затем отфильтровать только те, которые имеют отношение к текущему пользователю, а затем, если список вывода больше чем экран консоли, обеспечить скроллинг командой *more*:

```
ps -ef | grep $USER | more
```

Если между командами используется символы '||', то вторая команда, следующая за первой, выполнится только в случае невыполнения первой. Например, удалить файл abc.sh, расположенный в текущем каталоге и если его нет, вывести подробный список файлов:

```
rm ./abc.sh || ls -l
```

Если между командами используется символы '&&', то вторая команда, следующая за первой, выполнится только в случае успешного завершения первой. Например, создать каталог и перейти в него с выводом текущего местоположения:

```
mkdir ./test && cd ./test && pwd
```

Если между командами используется символ ';' , то формируется так называемый список команд, которые просто должны выполняться одна за другой как есть, вне зависимости от результатов выполнения предыдущей команды списка. Например, вывести список всех пользователей, работающих на

данный момент в системе, а затем список последних 10-ти вхождений в систему текущего пользователя (списки отделить один от другого пустой строкой):

```
who; echo; last $USER | head
```

Если требуется перенаправить вывод команд(ы) в файл, то используют символы перенаправления: '>' – создает новый файл (если файл существует, то его содержимое будет потеряно), '>>' – добавляет вывод команды в конец файла (если файла нет, то он будет создан). Например, перенаправить вывод всех перечисленных команд в файл logfile.txt:

```
(date; echo; free; echo; who; echo; last $USER | head) > ./logfile.txt
```

При перенаправлении можно ссылаться на файловые дескрипторы: 0 – стандартный ввод (*stdin*), 1 – стандартный вывод (*stdout*), 2 – стандартный вывод сообщений об ошибках (*stderr*). Например, вывести в файл find.txt результаты поиска (ошибки будут выведены на консоль):

```
find /etc -name '*.conf' > ~/find.txt
```

Перенаправить сообщения об ошибках в файл find_err.txt:

```
find /etc -name '*.conf' 2> ~/find_err.txt
```

Перенаправить весь вывод в файл find.txt:

```
find /etc -name '*.conf' > ~/find.txt 2>&1
```

Перенаправить стандартный вывод в файл find.txt, а поток сообщений об ошибках на нулевое устройство:

```
find /etc -name '*.conf' > ~/find.txt 2> /dev/null
```

Работа с архивами

В различных ОС существует множество программ архивирования данных. В Unix-подобных ОС получила широкое распространение утилита архивирования *tar* (tape archive). Первоначально эта программа разрабатывалась для создания архивов на магнитной ленте. Сейчас она используется для хранения нескольких файлов внутри одного файла, а также для распространения ПО и создания архива файловой системы. Одним из преимуществ формата tar при создании архивов является то, что в архив записывается информация о структуре каталогов, информация о владельце и

группе отдельных файлов, а также временные метки файлов. Одной из особенностей *tar* является то, что она не создает сжатых архивов, а использует для сжатия внешние утилиты, например, такие как *gzip, bzip2* или аналогичные.

Файлы, содержащие архивы *tar* имеют, как правило, расширение *.tar*. Если tar-файл сжат внешней утилитой, то к первоначальному расширению добавляется еще одно, свидетельствующее о том, какая утилита сжатия использовалась. Например, в случае gzip – это расширения: *.tar.gz, .tar.gzip, .tgz*; в случае с bzip2 – это расширения: *.tar.bz2, .tar.bzip2, .tbz2, .tbz, .tb2*.

Пусть, например, существует каталог /home/user/data, содержимое которого должно быть подвергнуто архивации; имя файла-архива: data.tar.gz. Тогда необходимо ввести следующую команду:

```
cd  /home/user ; tar  -czf  ./data.tar.gz  data
```

Опция *-c* говорит о создании архива, *-z* устанавливает использование утилиты *gzip*, а за ключом *-f* необходимо указать имя создаваемого файла-архива.

Если необходимо вывести список файлов и каталогов, размещенных в архиве без разархивирования, то используют опцию *-t*:

```
tar  -tf  ./data.tar.gz
```

Разархивирование данных происходит при указании опции *-x*. Если задана опция *-v*, то пользователь увидит вывод на экран обрабатываемых файлов архива, а при указании опции *-C* можно указать каталог для разархивирования. Если каталог не задан, то разархивирование будет произведено в текущий каталог. Примеры команды разархивирования:

```
tar  -zxf  ./data.tar.gz
tar  -zxvf  ./data.tar.gz  -C  /home/user/tmp/
```

Практическая часть

Написать командный bash-скрипт, позволяющий:

– создать в домашнем каталоге текущего пользователя каталог Images;

– создать в Images подкаталоги JPG, PNG, GIF;

– осуществить поиск всех файлов с расширениями JPG, jpg в каталоге /usr и его подкаталогах и найденные файлы сразу скопировать в каталог JPG;

– выполнить те же действия с файлами PNG и GIF, копируя их в соответствующие каталоги;

– после поиска и копирования файлов, создать архив содержимого каталога Images при помощи программы *tar*. Выходное имя файла-архива: images.tar.gz;

– осуществить рекурсивное удаление каталога Images.

При подготовке bash-скрипта использовать возможности команды *find* (ее опций *-iname* и *–exec*; см. приложение В). Для более детальной информации использовать справку по команде *find*. Пути к создаваемым каталогам должны храниться в соответствующих переменных: *DATADIR*, *JPGDIR*, *PNGDIR*, *GIFDIR*. После завершения создания архива, удаление каталога Images должно быть осуществлено только при условии успешного создания файла-архива. При этом в случае успеха скрипт возвращает в среду выполнения код нуля, в случае неудачи – код единицы.

В отчет включить код скрипта с комментариями. Продемонстрировать работу скрипта.

Лабораторная работа № 5
Microsoft Windows PowerShell

Цель работы: приобрести навыки работы в командной оболочке Microsoft Windows PowerShell, изучив основные команды и приемы работы.

Теоретическая часть

Начало работы

Windows PowerShell – расширяемое средство автоматизации от Microsoft, состоящее из оболочки с интерфейсом командной строки и сопутствующего языка сценариев. Windows PowerShell построен на базе Microsoft .NET Framework и интегрирован с ним (как минимум требует предустановленную в ОС библиотеку .NET Framework версии 2.0). Дополнительно, PowerShell предоставляет удобный доступ к технологиям COM (Component Object Model – объектная модель компонентов), WMI (Windows Management Instrumentation – инструментарий управления Windows) и ADSI (Active Directory Service Interfaces – интерфейсы службы Active Directory), равно как и позволяет выполнять обычные команды командной строки для создания единого окружения, в котором администраторы смогли бы выполнять различные задачи на локальных и удаленных системах.

Версия 1.0 выпущена в 2006 году и доступна для установки для сред Windows XP SP2, SP3, Windows Server 2003, Windows Vista.

Версия 2.0 интегрирована в виде компонента систем Windows 7, Windows Server 2008 R2 и поставляется с графической оболочкой для языка Windows PowerShell Integrated Scripting Environment (ISE), содержащей встроенный отладчик, подсветку синтаксиса, автозавершение команд.

Версия 3.0 интегрирована в виде компонента систем Windows 8, Windows Server 2012 и также содержит ISE.

В работе рассмотрена работа со **второй версией** Windows PowerShell.

Для запуска Windows PowerShell ISE нажмите *Пуск \ Выполнить* и выполните команду:

```
powershell_ise
```

Эти же действия могут быть выполнены через соответствующий ярлык в программной группе меню *Пуск*.

Чтобы узнать версию установленной Windows PowerShell (далее PS), в ее консоли введите команду:

```
$host
```

или

```
$host.version
```

На рис. 5.1 представлено главное окно среды ISE.

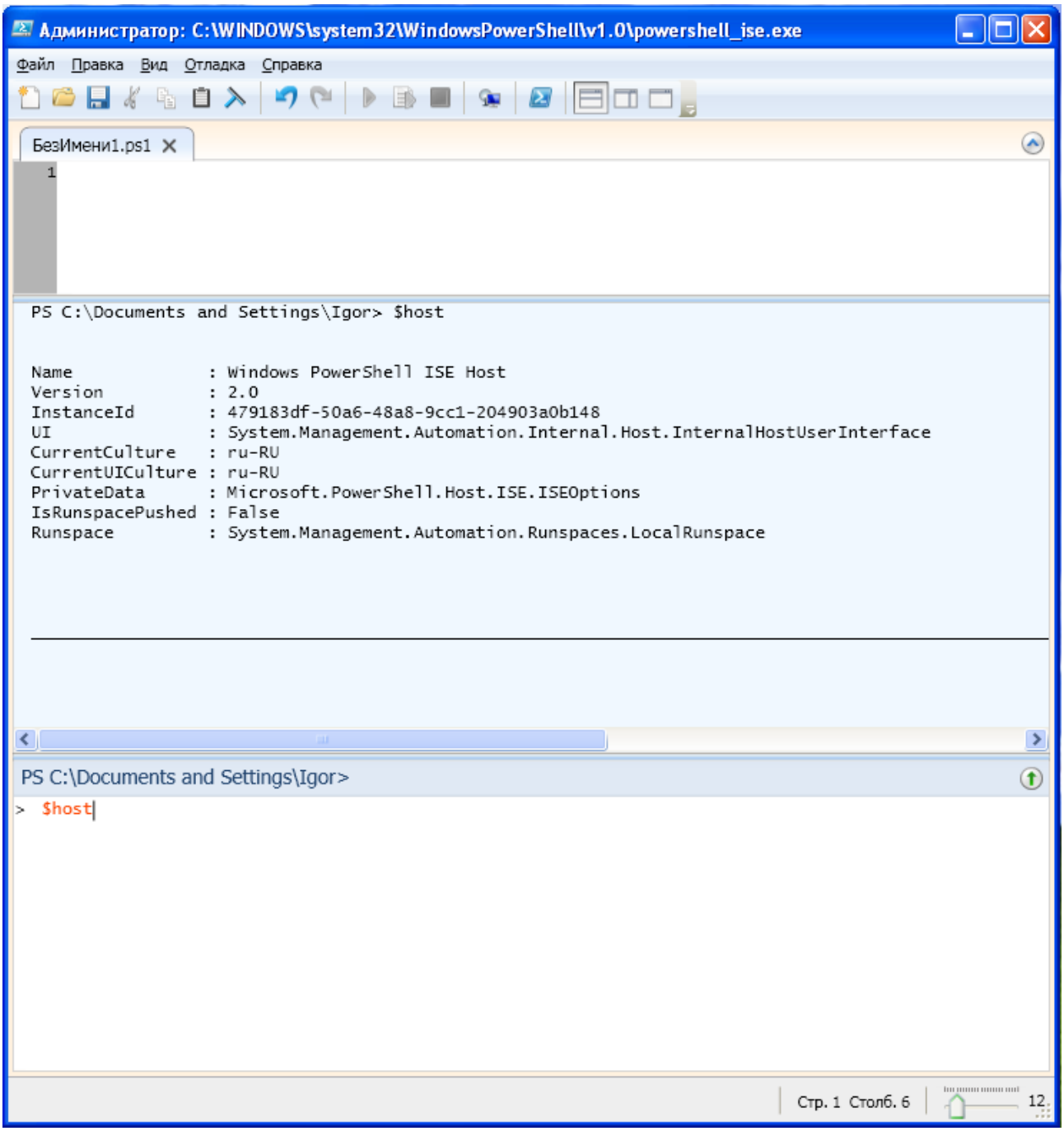

Рис. 5.1. Windows PowerShell ISE

PS содержит более 130 команд, называемых командлетами (cmdlets). Они следуют определенным правилам именования:

1. Команды PS состоят из глагола и существительного (в единственном числе), разделенных тире, записываемых на английском языке. Например, команда вызова справки: *Get-Help* (регистр значения не имеет).

2. Перед параметрами ставится символ тире. Например, команда более детального вывода справочной информации:

Get-Help -Detailed

3. В PS многим командам соответствуют псевдонимы. Например, команде *Get-Help* соответствуют псевдонимы *help* (классический Windows) и *man* (классический Unix).

Для автоматического завершения команд используется клавиша TAB.

Команда вывода всех команд PS:

```
Get-Help  *
```

Параметр *-Detailed*, как правило, используют совместно с определенной командой. Например, следующие команды выводят пользователю сокращенную и детальную информацию о команде *Get-Process*:

```
Get-Help  Get-Process
Get-Help  Get-Process  -Detailed
```

Команды вывода полной справки и примеров выглядят так:

```
Get-Help  Get-Process  -examples
Get-Help  Get-Process  -full
```

При использовании команд может использоваться конвейер (формируется через символ ‘|’). Так, например, следующая команда выведет полный список команд PS, а затем перенаправит этот список в указанный файл при помощи командлета *Out-File*:

```
Get-Help  *  |  Out-File  c:\temp\help.txt
```

Считается, что на диске пользователя каталог *C:\Temp* существует.

Эту же команду можно записать в традиционном стиле:

```
Get-Help  *  >  c:\temp\help.txt
```

Для запуска одного или нескольких процессов на локальном компьютере используют команду *Start-Process* или ее псевдонимы *saps* или *start*. Например, для открытия сохраненного файла help.txt можно использовать приложение WordPad:

```
Start-Process  wordpad  c:\temp\help.txt
```

или

```
saps  wordpad  c:\temp\help.txt
```

или

```
start  wordpad  c:\temp\help.txt
```

Для выполнения команд в комплексе они группируются в командные скрипты – файлы с расширением *.ps1*. Например, следующий командный скрипт формирует файл help.txt и загружает его содержимое в WordPad:

```
Get-Help * | Out-File c:\temp\help.txt
start wordpad c:\temp\help.txt
```

Текст скрипта записывается в верхней части, в редакторе ISE (рис. 5.1).

В скриптах могут быть задействованы переменные. Например, предыдущий скрипт может быть переписан следующим образом (полное имя файла хранится в переменной *f*):

```
$f="c:\temp\help.txt"
Get-Help * | Out-File $f
start wordpad $f
```

Выполнение скриптов

Для выполнения скриптов PS необходимы определенные установки политик системы безопасности Windows. Проверить, какая действует политика выполнения для текущего сеанса, можно при помощи командлета *Get-ExecutionPolicy*. Изменить политику можно командлетом *Set-ExecutionPolicy*.

Допустимые значения политики выполнения:

– *Restricted* – не загружает файлы конфигурации и не выполняет скрипты;

– *AllSigned* – требует, чтобы все скрипты и файлы конфигурации были подписаны доверенным издателем, в том числе скрипты, подготовленные на локальном компьютере;

– *RemoteSigned* – требует, чтобы все скрипты и файлы конфигурации, загруженные из Internet, были подписаны доверенным издателем;

– *Unrestricted* – загружает все файлы конфигурации и выполняет все скрипты. Если запущен неподписанный скрипт, загруженный из Internet, то программа просит ввести разрешение перед запуском;

– *Bypass* – ничего не блокируется и никакие предупреждения и запросы не появляются;

– *Undefined* – удаляет текущую назначенную политику выполнения из текущей области (параметр не удаляет политику выполнения, заданную в области групповой политики).

По умолчанию действует политика *Restricted*. Примеры команд:

```
Set-ExecutionPolicy -scope CurrentUser -ExecutionPolicy AllSigned -force
Get-ExecutionPolicy -list
```

Работа с процессами

Для вывода списка всех работающих в системе процессов применяется команда *Get-Process* (псевдоним *ps*). Для сортировки выводимого списка-результата (по умолчанию по возрастанию) используют другой командлет *Sort-Object* (или по убыванию с параметром *-Descending*). В качестве параметра указывают свойство объекта, например, процент загрузки процессора – CPU:

```
Get-Process | Sort-Object CPU
```

При помощи параметра *FileVersionInfo* (псевдоним *fvi*) получают информацию о версии файла-процесса и его размещении на диске. Например:

```
ps -fileversioninfo
```

или

```
ps -fvi
```

Для вывода информации о процессах с удаленного компьютера используют команду с параметром *ComputerName* (псевдоним *cn*). При этом пользователь, выполняющий команду, должен обладать соответствующим доступом к удаленному компьютеру. Например:

```
ps -ComputerName compname
```

или

```
ps -cn compname
```

Для извлечения очень подробной информации о процессах используют дополнительный командлет *Format-List* (псевдоним *fl*). Например, первая команда перенаправляет в файл process.txt подробные сведения обо всех загруженных процессах локального компьютера, вторая – выводит информацию только о процессе *explorer*:

```
ps | Format-List * | Out-File c:\temp\process.txt
ps explorer | fl *
```

При выполнении команды *ps* можно указывать различные условия, ограничивающие вывод информации. Это достигается, например, при помощи командлета *Where-Object* (псевдонимы *where* или *?*), создающего фильтр, который определяет, какие объекты будут переданы по командному конвейеру. Например, необходимо вывести все процессы локального компьютера, потребляющие в текущий момент времени более 20 Мбайт памяти:

```
Get-Process | Where-Object {$_.WorkingSet -gt 20000000}
```

или

```
ps | ? {$_.WS -gt 20000000}
```

или

```
ps | ? {$_.WS -gt 20*1024*1024}
```

где *gt* – означает условие “больше”. Другие возможные варианты:
eq – “равно”,
lt – “меньше”,
ne – “не равно”,
le – “меньше или равно”,
ge – “больше или равно”.

Если производится сравнение строк и требуется учитывать проверку регистра символов, то перед требуемым обозначением операции используют символ “*c*”. Например, следующая команда сравнивает две строки и возвращает значение *True* или *False* в зависимости от результата:

```
# возврат True
“this”  -eq  “THIS”

# возврат False
“this”  -ceq  “THIS”
```

Условные выражения и перехват исключений

При помощи инструкции *If* можно выполнять определенные блоки кода, когда заданное условие имеет значение *True*. Можно задавать одно или несколько дополнительных условий. Синтаксис инструкции *If*:

```
if (<условие_1>)
{ <список_инструкций_1> }
[
   elseif (<условие_2>)
   { <список_инструкций_2> }
]

[
   else
   { <список_инструкций_3> }
]
```

Пример 5.1. Вывод сообщения в случае, если значение переменной *a* больше значения 2:

```
$a=3
if ( $a -gt 2 ) { echo "Значение $a больше чем 2" }
```

Пример 5.2. Определение существования указанного файла:

```
$f="c:\temp\commands.txt"
if ( test-path -path $f )
{
   echo "Файл $f существует"
}
else
{
   help * > $f
   if ( test-path -path $f ) { echo "Создан файл $f" }
   else { echo "Ошибка создания файла $f" }
}
```

Часто при выполнении каких-либо операций необходимо реагировать на исключительные ситуации. Например, невозможно создать файл, если указан неверно путь к нему. В примере 5.2 будет выдана ошибка, если каталог *C:\Temp* отсутствует. Поэтому более корректно использовать также специальные конструкции обработки исключений: блоки *try*, *catch*, *finally*. Например:

```
$f="c:\temp\commands.txt"
if ( test-path -path $f )
{
   echo "Файл $f существует"
}
else
{
   try
   {
      help * > $f
   }
   catch [System.IO.IOException]
   {
      echo "Ошибка создания файла $f!"
   }
   finally
   {
      if ( test-path -path $f ) { echo "Создан файл $f" }
   }
}
```

В таблице 5.1 представлены некоторые часто используемые классы исключений.

В случаях, когда список альтернативных условий в инструкции *If* велик, прибегают к инструкции *Switch*. Она поочередно сопоставляет выражение с каждым условием. Если обнаружено совпадение, то выполняется действие, связанное с этим условием.

Таблица 5.1

Часто используемые классы исключений

Имя класса исключения	*Описание*
System.Exception	Представляет ошибки, происходящие во время выполнения приложения.
System.ArithmeticException	Исключение выбрасывается для ошибок арифметических действий, а также операций приведения к типу и преобразования. Наследники класса: System.DivideByZeroException System.NotFiniteNumberException System.OverflowException
System.IO.IOException	Исключение, создаваемое при возникновении ошибки ввода-вывода. Наследники класса: System.IO.DirectoryNotFoundException System.IO.EndOfStreamException System.IO.FileNotFoundException System.IO.FileLoadException System.IO.PathTooLongException
System.FormatException	Исключение, выбрасываемое, если формат аргумента не соответствует спецификациям параметра вызываемого метода.
System.IndexOutOfRangeException	Исключение происходит при попытке обращения к элементу массива с индексом, находящимся вне границ массива.

Упрощенная форма инструкции *Switch*:

```
switch( "string" | number | variable )
{
        "string" | number | variable | {<выражение>}  { <список_инструкций> }
        [ default { <список_инструкций> } ]
}
```

Например:

```
$day = "day3"
switch ($day){
   day1 {"Понедельник"; break}
   day2 {"Вторник"; break}
   day3 {"Среда"; break}
   day4 {"Четверг"; break}
   day5 {"Пятница"; break}
   day6 {"Суббота"; break}
```

```
    day7 {"Воскресенье"; break}
    default {"Слишком много дней"}
}
```

Циклические операции

Достаточно часто при обработке массивов данных возникает необходимость циклической обработки. Для этого применяют инструкции *For*, *Foreach*, *While*, *Do-While*, *Do-Until*.

Синтаксис инструкции *For*:

for (<инициализация>; <условие>; <повторение>)
{ <список_инструкций> }

Синтаксис инструкции *Foreach*:

foreach ($<элемент> in $<коллекция>) { <список_инструкций> }

Синтаксис инструкции *While*:

while (<условие>) { <список_инструкций> }

Синтаксис инструкций *Do-While* (выполняется пока условие остается истинным) и *Do-Until* (выполняется пока условие ложно):

do { <список_инструкций> } while (<условие>)
do { <список_инструкций> } until (<условие>)

Пример 5.3. Цикл вывода значений от 1 до 10 без перевода на новую строку через пробел:

```
for( $i=1; $i -le 10; $i++) { Write-Host -NoNewline $i" " }
```

Пример 5.4. Цикл вывода значений от 1 до 10 с переводом на новую строку:

```
for( $i=1; $i -le 10; $i++) { echo $i }
```

Пример 5.5. Бесконечный цикл вывода на консоль. Прервать его пользователь может по комбинации клавиш Crtl+C:

```
for(;;) { echo "i" }
```

Пример 5.6. Вывести количество элементов массива, расположенных до нулевого элемента:

```
$x = 1, 2, 78, 0
$count = $i = 0
do {
   $count++;
   $i++;
} while ($x[$i] -ne 0)
$count
```

Пример 5.7. Вывод всех элементов массива:

```
$data = 1, 2, 78, 0
foreach( $v in $data ) { Write-Host -NoNewline $v" " }
echo ""
```

Пример 5.8. Скрипт вывода на экран всех имен файлов из каталога *C:\Temp*, чей размер больше либо равен 3 Мбайт:

```
# $max - filter value size in MB
echo "======================================"
$max = 3
$dir = "c:\temp"
Write-Host "Find in directory" $dir
$count = 0
$size = 0
foreach ($file in Get-ChildItem -path $dir)
{
   if ($file.length -ge $max*1024*1024)
   {
     $fname = "File Name: "+$file.Name
     $size += $file.length
     $fsize = ($file.length / 1024 / 1024).ToString("F0")
     $fsize = "File size: "+$fsize+" MB"
     Write-Host ($fname, $fsize, "") -separator "`n"
     $count++
   }
}
Write-Host "Total Founds:"$count
Write-Host "Total size:"($size / 1024 / 1024).ToString("F0") "MB"
echo "======================================"
```

При формировании строки в последнем примере используется спецсимвол перевода на новую строку: `n.

В PS распознаются следующие специальные символы:

`0 – Null

`a – Предупреждение (звуковой сигнал)

`b – Возврат курсора
`f – Перевод страницы
`n – Новая строка
`r – Возврат каретки
`t – Горизонтальная табуляция
`v – Вертикальная табуляция

Перечисленные спецсимволы вводятся с учетом регистра!

Пользовательские объекты в PS

Для создания экземпляра объекта .NET Framework в PS используется командлет *New-Object*. Командлету сообщается тип создаваемого объекта, например, *PSObject*, который обеспечивает согласованное представление любого объекта в среде PS. В параметре *Property* командлета необходимо описать так называемую хеш-таблицу – пары "ключ-значение" (сокращенный синтаксис – @{}). Пары "ключ-значение", определенные в хеш-таблице и переданные через параметр *Property*, преобразуются в свойства и значения в новом *PSObject.*

Например, необходимо осуществить вывод в виде таблицы информации о файле, содержащей имя и его размер в килобайтах. Тогда при помощи командлетов *New-Object* и *Format-Table* это можно реализовать следующим образом:

```
Get-Help * > C:\temp\commands.txt
$f = Get-ChildItem -Path C:\temp\commands.txt
$NewObject = New-Object PsObject -Property @{
        FileName=$f.Name; FileSize=($f.Length/1024).ToString("F0")+" KB" }
$NewObject | Format-Table -auto
```

Вначале извлекаем информацию о файле при помощи командлета *Get-ChildItem*. Затем создаем новый объект при помощи командлета *New-Object*, содержащий два строковых свойства: *FileName* и *FileSize*. Передаем этим свойствам требуемые значения, исходя из решаемой задачи, и при помощи командлета *Format-Table* выводим значения свойств в виде таблицы на экран.

Для упрощения создания объекта можно также использовать командлет *Select-Object*. Например, все указанные выше действия можно представить так:

```
Get-Help * > C:\temp\commands.txt
$f = Get-ChildItem -Path C:\temp\commands.txt

$obj = 0 | Select-Object `
   @{name="FileName";expression={$f.Name}},
   @{name="FileSize";expression={($f.Length/1024).ToString("F0")+" KB"}}
$obj | Format-Table -auto
```

Создаваемые объекты можно накапливать в массив при помощи операции +=.

Вот как будет выглядеть описанный выше пример 5.8 с применением технологии создания объектов и вывода информации в табличном виде (новые строки кода выделены жирным курсивом):

```
# $max - filter value size in MB
echo "====================================="
$max = 3
$dir = "c:\temp"
Write-Host "Find in directory" $dir
$count = 0
$size = 0
$NewObject =@( )
foreach ($file in Get-ChildItem -path $dir)
{
  if ($file.length -ge $max*1024*1024)
  {
   $fname = "File Name: "+$file.Name
   $size += $file.length
   $fsize = ($file.length / 1024 / 1024).ToString("F0")
   $fsize = "File size: "+$fsize+" MB"
    $NewObject += New-Object PSObject -Property @{FileName=$fname;FileSize=$fsize}
   $count++
  }
}

$NewObject | Format-Table -auto
Write-Host "Total Founds:"$count
Write-Host "Total size:"($size / 1024 / 1024).ToString("F0") "MB"
echo "====================================="
```

Для вывода информации о методах и свойствах объекта используют командлет *Get-Member* (псевдоним *gm*). Например:

```
$f = Get-ChildItem -Path C:\temp\commands.txt
echo "`nDecription File Object:"
$f | Get-Member

$obj = 0 | Select-Object @{n="FileName";e={""}}, @{n="FileSize";e={""}}
echo "`nDecription My Object:"
$obj | gm
```

Работа с файлами и каталогами

В PS имеются несколько командлетов, управляющих файлами и каталогами. Например:

– *Get-ChildItem* – извлекает элементы и их потомки из заданных

местоположений (псевдоним *ls*);

– *Get-Item* – получает элемент, находящийся в заданном местоположении (псевдоним *gi*);

– *New-Item* – создает новый элемент и задает его значение (псевдоним *ni*);

– *Copy-Item* – копирует элемент из одного местоположения в другое внутри одного пространства имен (псевдоним *cp*);

– *Move-Item* – перемещает элемент из одного местоположения в другое (псевдоним *mv*);

– *Remove-Item* – удаляет заданные элементы (псевдоним *rm*);

– *Rename-Item* – переименовывает элемент в пространстве имен поставщика PS (псевдоним *ren*);

– *Get-Content* – извлекает содержимое элемента, находящегося в заданном местоположении, например, текст из файла (псевдоним *cat*).

Пример 5.9: создание в каталоге *C:\Temp* файлов *testfile1.txt, testfile2.txt* и каталога *scripts*. Второй текстовый файл содержит короткий текст. После создания осуществляется вывод на экран содержимого каталога и файла с текстом:

```
clear
rm c:\temp\testf*.txt
rm c:\temp\scripts
new-item -path c:\temp -name testfile1.txt -type "file"
new-item c:\temp\testfile2.txt -type "file" -value "This is a text string."
new-item -type directory -path c:\temp\scripts
ls c:\temp -include testf*.txt, c:\temp\sc*
cat c:\temp\testfile2.txt
```

Разница между командлетами *Get-Item* и *Get-ChildItem*: *Get-Item* не имеет параметра *Recurse*, так как он извлекает только элемент, а не его содержимое. Для рекурсивного извлечения содержимого элемента используется командлет *Get-ChildItem*.

Если необходимо произвести в текстовом файле поиск строк по определенному шаблону, то используют командлет *Select-String*. Например, пусть необходимо вывести на экран только те строки из файла *C:\Temp\commands.txt*, в которых содержится слово *about*:

```
ls C:\temp\commands.txt | Select-String about
```

или

```
ls C:\temp\commands.txt | Select-String -pattern "about"
```

Усложним задачу. Пусть, например, необходимо сформировать массив, с номерами найденных строк в указанном файле. Это можно выполнить, например, следующим образом:

```
$lines = (ls C:\temp\commands.txt | Select-String -pattern "about")
$mas = 0..($lines.length-1)
for($i = 0; $i -le $lines.length-1; $i++) { $mas[$i] = $lines[$i].LineNumber }
```

Использование математических функций

Для использования в PS аппарата математических функций чаще используют класс *Math* пространства имен *System* библиотеки .NET Framework. Например, в классе существуют следующие методы:

Abs(Decimal), Abs(Double), Abs(Int16), Abs(Int32), Abs(Int64), Abs(SByte), Abs(Single) – возвращают абсолютную величину значения соответствующего типа.

Acos(double), *Asin*(double), *Atan*(double), *Atan2*(double y, double x), *Cos*(double), *Cosh*(double), *Sin*(double), *Sinh*(double), *Tan*(double), *Tanh*(double) – тригонометрические функции.

Pow(double *x*, double *y*) – возведение значения *x* в степень *y*.

Sqrt(double) – корень квадратный числа.

Log(double) – натуральный логарифм (по основанию *e*).

Log(double *a*, double *newBase*) – вычисление логарифма по основанию *newBase* числа *a*.

Log10(double) – вычисление логарифма по основанию 10.

Exp(double *d*) – возведение *e* в степень *d*.

В классе существуют перегруженные методы *Min*, *Max*, *Round*, *Floor*, *Truncate*. Существуют и другие методы.

В класс введены две константы:

E (2.7182818284590452354) и *PI* (3.14159265358979323846).

Таким образом, класс *Math* содержит достаточно много математических методов для расчета сколь угодно сложных математических выражений. В PS это выглядит, например, так:

```
$Num = 49.68124
$res1=[System.Math]::Round($Num, 2)
$res2=[Math]::Truncate($Num)
$res3=[Math]::Sqrt($Num)
echo ""
$res1, $res2, $res3
```

Практическая часть

Создать в папке пользователя скрипт на PS, выполняющий следующие действия:

1. В домашнем каталоге пользователя создает каталог Data.

2. В каталоге Data формирует файл commands.txt с перечнем всех команд PS.

3. Делает выборку из файла commands.txt всех частей about-инструкций и формирует в каталоге Data отдельные файлы с полным описанием каждой about-инструкции. При выборке строк использовать свойство *Line* и его метод

Split с передачей в качестве параметра символа-разделителя, например, пробела.

В каталоге пользователя создать второй скрипт для вычисления математического выражения согласно варианту задания, выданного преподавателем (табл. 5.2 – 5.4).

В отчете привести коды написанных скриптов с комментариями, а также результаты их работы.

Устно дать ответы на следующие вопросы.

1. Каким образом определить версию Windows PowerShell?
2. Что такое командлет?
3. Как вызвать справку по командлету или инструкции?
4. Какие условия необходимы для выполнения скриптов на Windows PowerShell?
5. Как получить сведения о загруженных процессах?
6. Какие в PS существуют операторы условий?
7. Как построить условные выражения на PS?
8. Когда и как необходимо применять инструкцию *Switch*?
9. Как обрабатывать исключительные ситуации?
10. Какие существуют инструкции для построения циклов? Привести примеры.
11. Какие бывают спецсимволы в PS?
12. Как осуществлять вывод на консоль нескольких строк через разделитель?
13. Как осуществлять вывод на консоль данных в табличном представлении?
14. Как создавать свои объекты .NET Framework в PS?
15. Какие известны командлеты для управления файлами и каталогами?
16. Как вызывать математические функции в скриптах на PS?

Таблица 5.2

Варианты заданий

Вариант задания	*Вариант выражения*	*Вариант входных данных*	*Вариант задания*	*Вариант выражения*	*Вариант входных данных*
1	1	1	11	1	2
2	2	2	12	2	1
3	3	1	13	3	2
4	4	2	14	4	1
5	5	1	15	5	2
6	6	2	16	6	1
7	7	1	17	7	2
8	8	2	18	8	1
9	9	1	19	9	2
10	10	2	20	10	1

Таблица 5.3

Входные данные

Вариант входных данных	a_i
1 (тип данных int)	5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55
2 (тип данных double)	0.3, 0.7, 0.9, 1.3, 1.7, 1.9, 2.3, 2.7, 2.9, 3.3, 3.7

Таблица 5.4

Варианты выражений

Вариант выражения	*Выражения*
1	$S = \frac{\sum_{i=1}^{5} \sin\lvert 1 - \ln a_i \rvert}{\sum_{i=1}^{11} \sin^2 18 a_i^3}$
2	$S = \frac{\sum_{i=1}^{11} \sin\left(1 - 3 * \sin^3 a_i\right)}{3 * \sum_{i=1}^{11} \cos^2 a_i^5}$
3	$S = \frac{\sum_{i=1}^{11} \cos\lvert 1 - \ln^2 a_i \rvert}{\sum_{i=1}^{11} \sin^2 18 a_i^{a_i}}$
4	$S = \frac{\ln\left\lfloor \sum_{i=1}^{11} \left(\lvert \sin a_i \rvert^{\sin a_i} + \lvert \cos a_i \rvert^{\cos a_i} \right) \right\rfloor}{3 * \sum_{i=1}^{11} \sin^2 a_i^{\sin a_i}}$
5	$S = \frac{\sin^2 \left\lfloor \sum_{i=1}^{11} \ln\left(\lvert 1 - \cos^2 a_i \rvert \right) \right\rfloor}{\sum_{i=1}^{11} \cos\left(\lvert 1 - \sin a_i \rvert \right)}$
6	$S = \sum_{i=1}^{11} \frac{1 - 2\cos^2 a_i}{\lvert \sin a_i \rvert * \lvert \cos a_i \rvert}$
7	$S = \sum_{i=1}^{11} \left(1 + \cos a_i + \cos \frac{a_i}{2} \right)$
8	$S = \sum_{i=1}^{11} \frac{\cos 2a_i}{1 - \sin 2a_i}$
9	$S = \sqrt{\frac{\sum_{i=1}^{11} (a_i - \tilde{a})^2}{10}}, \tilde{a} = \frac{1}{11} \sum_{i=1}^{11} a_i$
10	$S = \sum_{i=1}^{11} \left(1 + \sin a_i + \sin \frac{a_i}{2} \right)$

СПИСОК ЛИТЕРАТУРИ

1. Таненбаум Э. Современные операционные системы. 3-е изд. – СПб.: Питер, 2010. – 1120 с.

2. Гордеев А.В. Операционные системы: Учебник для вузов. 2-е изд. – СПб.: Питер, 2009. – 416 с.

3. Сетевые операционные системы / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер, 2007. – 544 с.

4. Робачевский А.М., Немнюгин С.А., Стесик О.Л. Операционная система UNIX. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2010. – 656 с.

5. Лукас М. FreeBSD. Подробное руководство: 2-е изд. – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 864 с.

6. Операционная система Linux: курс лекций. Учеб. Пособие для студентов вузов, обучающихся по специальностям в области информ. технологий / Г.В. Курячий, К.А. Маслинский. – М.: Интернет-Ун-т информ. технологий, 2005. – 392 с.

7. Колисниченко Д.Н., Аллен Питер В. Linux: полное руководство. – СПб.: Наука и Техника, 2006. – 784 с.

8. Колисниченко Д.Н. Самоучитель Linux. Установка, настройка, использование. – 5-е изд. – СПб.: Наука и Техника, 2009. – 368 с.

9. Стахнов А.А. Linux: 3-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2009. – 1056 с.

10. Глушаков С.В., Сурядный А.С. Linux для дома и офиса: Учебный курс. – Харьков: Фолио, 2002. – 389 с.

11. Сивер Э., Спейнауэр С., Фиггинс С., Хекман Д. Linux. Справочник. – Пер. с англ. – СПб.: Символ-Плюс, 2001. – 912 с.

12. Орлов А.А. 99 советов по Linux. – СПб.: БХВ-Петербург, 2007. – 352 с.

13. Попов А.В. Введение в Windows PowerShell. – СПб.: БХВ-Петербург, 2009. – 464 с.

14. Кох Ф. Windows PowerShell. Введение в технологии языка сценариев для пользователей без базовых знаний. – Берн: Microsoft, 2007. – 44 с.

15. Торрес Дж. Скрипты для администратора Windows. Специальный справочник. – СПб.: Питер, 2002. – 336 с.

Приложение А

Популярные Internet-адреса серверов Unix/Linux

Internet-адрес	*Описание ресурса*
1	*2*
http://www.opennet.ru/	Новости, статьи, программы, форум, советы и документация по Unix/Linux
http://www.linuxcenter.ru/	On-line магазин дистрибутивов, книг и журналов по Unix/Linux
http://www.linux.ru/	Крупный портал по Linux-системам: новости, статьи, конференции, блоги, документация, вход на википедию Linux
http://www.linux.kiev.ua/	Новости и форум по Linux
http://www.linux.ru/forum/	Мегафорум по Linux-системам
http://www.linuxforum.ru/	Мегафорум по Linux-системам
http://www.linux.org.ru/	Русская информация об ОС Linux: форум, дистрибутивы
http://www.rhd.ru/	Сайт российского представителя Red Hat Инвента с русскоязычной документацией
http://linuxgid.ru/	Российский образовательный Linux-портал
http://distrowatch.com/	Сайт, отслеживающий популярность дистрибутивов
http://www.kernel.org/	The Linux Kernel Archives
http://www.gnu.org/	Сайт проекта GNU
http://sourceforge.net/	Проекты различного свободного ПО для Linux, Windows и др. систем
http://www.linuxfoundation.org/	The Linux Foundation (Некоммерческий консорциум развития Linux)
http://www.fsf.org/	The Free Software Foundation (Фонд свободно распространяемого ПО)
http://www.icewalkers.com/	Linux Software – удобный поисковый портал для загрузки ПО Linux: от прикладных программ до различных ОС Linux
http://www.redhat.com/	Дистрибутив Red Hat
http://www.slackware.com/	Дистрибутив Slackware
http://www.vectorlinux.com/	Дистрибутив Vector Linux
http://www.suse.com/	Дистрибутив SUSE
http://software.opensuse.org/	Дистрибутив openSUSE
http://fedoraproject.org/	Дистрибутив Fedora
http://www.debian.org/	Дистрибутив Debian
http://www.freebsd.org/	Дистрибутив Unix FreeBSD
http://ubuntu.ru/	Дистрибутив Ubuntu (русское сообщество)

1	*2*
http://www.pcbsd.org/	Дистрибутив Unix PC-BSD
http://www.altlinux.ru/	Российский дистрибутив ALT Linux
http://ualinux.com/	Украинский Linux-портал проекта Ubuntu DesktopPack
http://linuxsam.org.ua/	Украинский информационно-образовательный Linux-портал
http://lin.in.ua/	Украинский Linux-портал
http://grusha.org.ua/	Украинский дистрибутив Grusha Linux
http://deepstyle.org.ua/	Украинский дистрибутив DeepStyle Linux
http://ubuntu-box.at.ua/	Украинская дополненная версия Ubuntu Linux
http://www.zenwalk.org/	Дистрибутив Zenwalk Linux
http://www.slax.org/	Дистрибутив Slax
http://www.linuxmint.com/	Дистрибутив Linux Mint
http://www.knoppix.net/	Live CD/DVD дистрибутив Knoppix
http://www.calculate-linux.ru/	Российский дистрибутив Calculate Linux
http://www.rosalab.ru/	Российская линейка дистрибутивов Rosa Linux
http://www.agilialinux.ru/	Российский дистрибутив Agilia Linux
http://www.apple.com/osx/	Проект Apple OS X
http://rpmfind.net/	Поиск RPM-пакетов
http://rpm.pbone.net/	Поиск RPM-пакетов
http://slackfind.net/ru/	Поисковый сервис по пакетам Slackware Linux
http://www.debian.org/distrib/packages	Поисковый сервис по пакетам Debian Linux
http://packages.ubuntu.com/ru/	Поисковый сервис по пакетам Ubuntu Linux
http://pkgs.org/	Удобная поисковая система Linux программ
http://ati.amd.com/products/index.html	Поддержка драйверов видеокарт ATI
http://www.nvidia.com/object/unix.htm	Поддержка драйверов видеокарт NVIDIA
http://intellinuxgraphics.org/	Поддержка драйверов видеосистем INTEL
http://www.linuxant.com/drivers/	Драйвера к чипсетам Conexant (например, к soft-модемам)

Приложение Б

Быстрое введение в основы Unix/Linux[1]

Краткая история появления Unix

В 1965 году Bell Telephone Laboratories (подразделение AT&T) совместно с General Electric Company и Массачусетским институтом технологии (MIT) начали разрабатывать новую ОС, названную MULTICS (MULTiplexed Information and Computing Service). Перед участниками проекта стояла цель создания многозадачной ОС разделения времени, способной обеспечить одновременную работу нескольких сотен пользователей. От Bell Labs в проекте приняли участие два сотрудника – Кен Томпсон (Ken Tompson) и Дэннис Ритчи (Dennis Ritchie). Хотя система MULTICS так и не была завершена (в 1969 году Bell Labs вышла из проекта), она стала предтечей ОС, впоследствии получившей название Unix (читается *ю́никс*). Однако Томпсон, Ритчи и ряд других сотрудников продолжили работу над созданием удобной среды программирования. Используя идеи и разработки, появившиеся в результате работы над MULTICS, они создали в 1969 небольшую ОС, включавшую файловую систему, подсистему управления процессами и небольшой набор утилит. Система была написана на ассемблере и применялась на компьютере PDP 7. Эта операционная система получила название Unix, созвучное MULTICS и придуманное другим членом группы разработчиков, Брайаном Керниганом (Brian Kernighan).

В 1973 году ядро ОС было переписано на языке высокого уровня C, – неслыханный до этого шаг, оказавший громадное влияние на популярность Unix. Это означало, что теперь система Unix может быть перенесена на другие аппаратные платформы за считанные месяцы, кроме того, значительная модернизация системы и внесение изменений не представляли особых трудностей.

Краткая история появления Linux

В январе 1991 года 21-летний студент Линус Торвальдс, изучающий компьютерные науки в Университете г. Хельсинки, покупает ПК 386 с 33 МГц-процессором, чтобы играть в Prince of Persia, и начинает писать Unix-подобную ОС для 386, используя книги Энди Таненбаума и Мориса Баха. В июне Ричард Столлмен публикует вторую версию своей сотрясающей основы GNU (General Public License), которая разрешает пользователям брать чужой код, коль скоро они выпускают плоды своих трудов под той же лицензией. В августе Торвальдс

[1] – Часть информации приложения Б взята из книги [4].

в группе новостей Usenet comp.os.minix сообщает миру, что пишет некую ОС, но она не будет «большой и профессиональной, как GNU». Рабочее название – Freax. В сентябре 1991 первая версия (0.01) того, что теперь называется Linux, выпущена с аппаратной поддержкой для финских клавиатур, а в декабре к версии 0.11 проекта Торвальдса в ОС добавлена поддержка гибких дисков. Торвальдс теперь рассматривает ее как самостоятельную систему, независимую от Minix (проект Таненбаума).

Официальная дата рождения Linux – 25 августа 1991 года.

Основы

Сегодня Unix используется на самых разнообразных аппаратных платформах – от персональных рабочих станций до мощных серверов с тысячами пользователей. И прежде всего потому, что Unix – это многозадачная многопользовательская система, обладающая широкими возможностями.

С точки зрения пользователя в ОС Unix существуют два типа объектов: файлы и процессы. Все данные хранятся в виде файлов, доступ к периферийным устройствам осуществляется посредством чтения/записи в специальные файлы. Когда вы запускаете программу, ядро загружает соответствующий исполняемый файл, создает образ процесса и передает ему управление. Более того, во время выполнения процесс может считывать/записывать данные в/из файла. С другой стороны, вся функциональность ОС определяется выполнением соответствующих процессов.

В Unix файлы организованы в виде древовидной структуры (дерева), называемой файловой системой (*file system*, *FS, ФС*). Каждый файл имеет имя, определяющее его расположение в дереве ФС. Корнем этого дерева является корневой каталог (*root directory*), имеющий имя «/». Имена всех остальных файлов содержат путь – список каталогов (ветвей), которые необходимо пройти, чтобы достичь файла. В Unix все доступное пользователям файловое пространство объединено в единое дерево каталогов, корнем которого является каталог «/». Таким образом, полное имя любого файла начинается с «/» и не содержит идентификатора устройства (дискового накопителя, CD-ROM или удаленного компьютера в сети), на котором он фактически хранится.

В большинстве случаев единое дерево, такое, каким его видит пользователь системы, составлено из нескольких отдельных ФС, которые могут иметь различную внутреннюю структуру, а файлы, принадлежащие этим ФС, могут быть расположены на различных устройствах.

Имя файла является атрибутом ФС, а не набора некоторых данных на диске, который не имеет имени как такового. Каждый файл имеет связанные с ним метаданные (хранящиеся в индексных дескрипторах – *inode*), содержащие все характеристики файла и позволяющие ОС выполнять операции, заказанные прикладной задачей: открыть файл, прочитать или записать данные, создать или удалить файл. В частности, метаданные содержат указатели на дисковые блоки хранения данных файла. Имя файла в ФС является указателем на его

метаданные, в то время как метаданные не содержат указателя на имя файла.

Типы файлов

В Unix существуют 6 типов файлов, различающихся по функциональному назначению и действиям ОС при выполнении тех или иных операций над файлами:

– обычный файл (*regular file*);
– каталог (*directory*);
– специальный файл устройства (*special device file*);
– FIFO или именованный канал (*named pipe*);
– связь (*link*);
– сокет (*socket*);

Обычный файл представляет собой наиболее общий тип файлов, содержащий данные в некотором формате. Для ОС такие файлы представляют собой просто последовательность байтов. Вся интерпретация содержимого файла производится прикладной программой, обрабатывающей файл. К этим файлам относятся текстовые файлы, бинарные данные, исполняемые программы и т.п.

Каталог. С помощью каталогов формируется логическое дерево ФС. Каталог – это файл, содержащий имена находящихся в нем файлов, а также указатели на дополнительную информацию – метаданные, позволяющие ОС производить операции над этими файлами. Каталоги определяют положение файла в дереве ФС, поскольку сам файл не содержит информации о своем местонахождении. Любая программа (задача), имеющая право на чтение каталога, может прочесть его содержимое, но только ядро имеет право на запись в каталог.

Специальный файл устройства обеспечивает доступ к физическому устройству. В Unix различают символьные (*character*) и блочные (*block*) файлы устройств. Доступ к устройствам осуществляется путем открытия, чтения и записи в специальный файл устройства. Символьные файлы устройств используются для небуферизированного обмена данными с устройством. В противоположность этому блочные файлы позволяют производить обмен данными в виде пакетов фиксированной длины – блоков. Доступ к некоторым устройствам может осуществляться как через символьные, так и через блочные специальные файлы.

FIFO или именованный канал – это файл, используемый для связи между процессами. FIFO впервые появились в UNIX System V и большинство современных систем поддерживают этот механизм.

Связь. Как уже говорилось, каталог содержит имена файлов и указатели на их метаданные. В то же время сами метаданные не содержат ни имени файла, ни указателя на это имя. Такая архитектура позволяет одному файлу

иметь несколько имен в ФС. Имена жестко связаны с метаданными и, соответственно, с данными файла, в то время как сам файл существует независимо от того, как его называют в ФС. Такая связь имени файла с его данными называется *жесткой связью* (*hard link*). Например, с помощью команды *ln* можно создать еще одно имя (*second*) файла, на который указывает имя *first*:

ln first /home/user/second

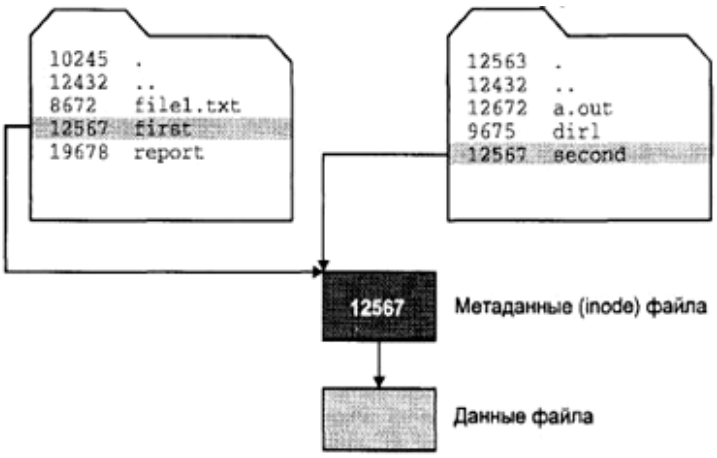

Рис. Б.1. Жесткая связь имен с данными файла [4]

Жесткие связи абсолютно равноправны. В списках файлов каталогов, которые можно получить с помощью команды *ls*, файлы *first* и *second* будут отличаться только именем. Все остальные атрибуты файла будут абсолютно одинаковыми. С точки зрения пользователя – это два разных файла. Изменения, внесенные в любой из этих файлов, затронут и другой, поскольку оба они ссылаются на одни и те же данные файла. Можно переместить один из файлов в другой каталог – все равно эти имена будут связаны жесткой связью с данными файла. Легко проверить, что удаление одного из файлов (*first* или *second*) не приведет к удалению самого файла, т.е. его метаданных и данных (если это неспециальный файл устройства). По определению жесткие связи указывают на один и тот же индексный дескриптор inode. Жесткая связь является естественной формой связи имени файла с его метаданными и не принадлежит к особому типу файла.

Особым типом файла является *символическая связь*, позволяющая косвенно адресовать файл. В отличие от жесткой связи, символическая связь адресует файл, который, в свою очередь, ссылается на другой файл. В результате, последний файл адресуется символической связью косвенно.

Данные файла, являющегося символической связью, содержат только имя целевого файла. Пример создания символической связи:

ln -s first /home/user/sfirst

Сокеты предназначены для взаимодействия между процессами. Интерфейс сокетов часто используется для взаимодействия различных локальных и удаленных процессов в сети TCP/IP. В системах класса BSD Unix на базе сокетов реализована система межпроцессного взаимодействия, с помощью которой работают многие системные сервисы, например, система печати.

Структура ФС Unix

Использование общепринятых имен основных файлов и структуры каталогов существенно облегчает работу в ОС, ее администрирование и переносимость. Эта структура используется в работе системы, например при ее инициализации и конфигурировании, при работе почтовой системы и системы печати. Нарушение этой структуры может привести к неработоспособности системы или отдельных ее компонентов.

Корневой каталог «/» является основой любой ФС Unix. Все остальные файлы и каталоги располагаются в рамках структуры, порожденной корневым каталогом, независимо от их физического местонахождения.

/bin – в нем находятся наиболее часто употребляемые команды и утилиты системы, как правило, общего пользования.

/dev – содержит специальные файлы устройств, являющиеся интерфейсом доступа к периферийным устройствам. Он может содержать несколько подкаталогов, группирующих специальные файлы устройств одного типа.

/etc – в этом каталоге находятся системные конфигурационные файлы и многие утилиты администрирования. Среди наиболее важных файлов – скрипты инициализации системы.

/lib – в каталоге находятся библиотечные файлы языка С и других языков программирования. Часть библиотечных файлов также находится в каталогах /usr/lib, */usr/local/lib*.

/lost+found, /home/lost+found – каталог "потерянных" файлов. Ошибки целостности файловой системы, возникающие при неправильном останове Unix или аппаратных сбоях, могут привести к появлению т.н. "безымянных" файлов – структура и содержимое файла являются правильными, однако для него отсутствует имя в каком либо из каталогов. Программы проверки и восстановления ФС помещают такие файлы в этот каталог под системными числовыми именами.

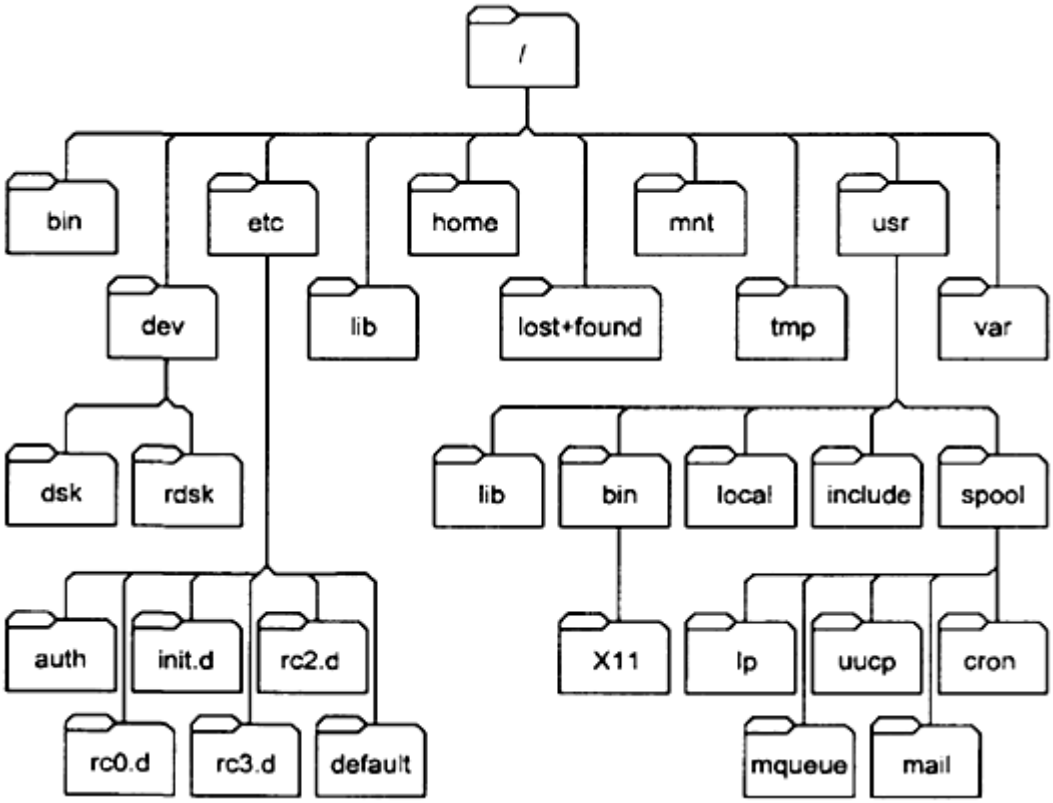

Рис. Б.2. Пример дерева ФС Unix [4]

/mnt, /media, /run/media – каталоги для временного связывания (монтирования) физических ФС к корневой для получения единого дерева логической ФС.

/home, /usr/home – каталог для размещения домашних каталогов пользователей.

/usr – в этом каталоге находятся подкаталоги различных сервисных подсистем – системы печати, электронной почты и т.д., исполняемые файлы утилит Unix, дополнительные программы, используемые на компьютере, файлы заголовков, электронные справочники и т.д.

/opt – каталог для дополнительного по отношению к /usr размещения программ.

/var – в разновидностях UNIX System V этот каталог является заменителем каталога /usr/spool, используемого для хранения временных файлов различных сервисных подсистем – системы печати, электронной почты и т.д.

/tmp – каталог хранения временных файлов, необходимых для работы различных подсистем Unix. Обычно этот каталог открыт на запись для всех пользователей системы.

Кроме вышеназванных, могут существовать и другие каталоги в корне дерева. В различных разновидностях Unix/Linux список таких каталогов может различаться или варьироваться. Узнать детально о каталогах той или иной Unix-like ОС можно при помощи команды:

man hier

Понятия программы и процесса

Процессы в Unix играют ключевую роль. От оптимальной настройки подсистемы управления процессами и числа одновременно выполняющихся процессов зависит загрузка ресурсов процессора, что в свою очередь имеет непосредственное влияние на производительность системы в целом.

Обычно *программой* называют совокупность файлов, будь то набор исходных текстов, объектных файлов или собственно выполняемый файл. Для того чтобы программа могла быть запущена на выполнение, ОС сначала должна создать *окружение* или *среду выполнения задачи*, куда относятся ресурсы памяти, возможность доступа к устройствам ввода/вывода и различным системным ресурсам, включая услуги ядра. Это окружение (среда выполнения задачи) получило название *процесса*. Можно представить процесс как совокупность данных ядра системы, необходимых для описания образа программы в памяти и управления ее выполнением. Можно также представить процесс как программу в стадии ее выполнения, поскольку все выполняющиеся программы представлены в Unix в виде процессов. Процесс состоит из инструкций, выполняемых процессором, данных и информации о выполняемой задаче, такой как размещенная память, открытые файлы и статус процесса.

Типы процессов

Системные процессы в Unix являются частью ядра и всегда расположены в оперативной памяти. Выполняемые инструкции и данные этих процессов находятся в ядре системы. Таким образом, они могут вызывать функции и обращаться к данным, недоступным для остальных процессов. Примером важнейшего системного процесса является *init*. Хотя *init* не является частью ядра, его работа жизненно важна для функционирования всей системы в целом.

Демоны – это неинтерактивные процессы, которые запускаются таким же образом, как и обычные программы. Однако они выполняются в фоновом режиме. Обычно демоны запускаются при инициализации системы (но после инициализации ядра) и обеспечивают работу различных подсистем Unix: системы терминального доступа, системы печати, системы сетевого доступа, сетевых услуг и т.п. Демоны не связаны ни с одним пользовательским сеансом работы и не могут непосредственно управляться пользователем. Большую часть

времени демоны ожидают пока тот или иной процесс запросит определенную услугу, например, доступ к файловому архиву или печать документа.

К *прикладным процессам* относятся все остальные процессы, выполняющиеся в системе. Как правило, это процессы, порожденные в рамках пользовательского сеанса работы. Важнейшим пользовательским процессом является основной командный интерпретатор (командный процессор) (login shell), который обеспечивает работу пользователя в Unix. Он запускается сразу же после регистрации пользователя в системе, а завершение работы login shell приводит к отключению от системы.

Пользовательские процессы могут выполняться как в интерактивном, так и в фоновом режиме, но в любом случае время их жизни (и выполнения) ограничено сеансом работы пользователя. При выходе из системы все пользовательские процессы будут уничтожены.

Атрибуты процесса

Процесс в Unix/Linux имеет несколько атрибутов, позволяющих ОС эффективно управлять его работой, важнейшими из которых являются следующие:

– идентификатор процесса – PID (*Process Identifier*);

– идентификатор родительского процесса – PPID (*Parent PID*);

– приоритет процесса (*Nice Number*);

– терминальная линия (*TTY*);

Каждый процесс имеет *уникальный идентификатор* PID, позволяющий ядру системы различать процессы. Когда создается новый процесс, ядро присваивает ему следующий свободный (т.е. не ассоциированный ни с каким процессом) идентификатор. Присвоение идентификаторов происходит по возрастающий, т.е. идентификатор нового процесса больше, чем идентификатор процесса, созданного перед ним. Если идентификатор достиг максимального значения, следующий процесс получит минимальный свободный PID и цикл повторяется. Когда процесс завершает свою работу, ядро освобождает занятый им идентификатор.

Идентификатор родительского процесса PPID – идентификатор процесса, породившего данный процесс.

Приоритет процесса – относительный приоритет процесса, учитываемый планировщиком при определении очередности запуска. Фактическое же распределение процессорных ресурсов определяется приоритетом выполнения, зависящим от нескольких факторов, в частности от заданного относительного приоритета. Относительный приоритет не изменяется системой на всем протяжении жизни процесса (хотя может быть изменен пользователем или администратором) в отличие от приоритета выполнения, динамически обновляемого ядром.

Терминальная линия – терминал или псевдотерминал, ассоциированный с процессом, если такой существует. Процессы демоны не имеют

ассоциированного терминала.

Статус процесса

Команда *ps* (*process status*) позволяет вывести список процессов, выполняющихся в системе, и их атрибуты. Команда имеет множество ключей запуска.

У команды *ps* есть множество различных опций, влияющих на выводимую информацию. Например, опция *-a* позволяет показывать информацию обо всех запущенных процессах, а не только тех, которыми владеет пользователь. Опция *-u* показывает имя пользователя, владеющего процессом и информацию об используемой памяти. Опция *-x* показывает информацию о процессах-демонах. Опция *-ww* указывает команде *ps* показать всю командную строку, вместо обрезания ее, когда она станет слишком длинной, чтобы уместиться на экране.

Пример вывода информации о запущенных в данный момент времени процессах командой *ps -ef* (символ '?' в столбце *TTY* говорит о том, что данный процесс не связан ни с каким терминалом):

```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 00:52 ?        00:00:01 init [4]
root         2     1  0 00:52 ?        00:00:00 [migration/0]
root         3     1  0 00:52 ?        00:00:00 [ksoftirqd/0]
.....................................................................................................
root      3185  3174  0 00:53 ?        00:00:00 /usr/sbin/smbd -D
root      3204     1  0 00:53 ?        00:00:00 /opt/kde/bin/kdm -nodaemon
root      3228     1  0 00:53 tty1     00:00:00 /sbin/mingetty tty1
root      3230     1  0 00:53 tty2     00:00:00 /sbin/mingetty tty2
.....................................................................................................
root      3290  3204  0 00:53 ?        00:00:00 -:0
user      3338  3290  0 00:53 ?        00:00:00 /bin/sh /opt/kde/bin/startkde
root      3384     1  0 00:53 ?        00:00:00 start_kdeinit --new-startup +kcminit_startup
user      3385     1  0 00:53 ?        00:00:00 kdeinit Running...
user      3388     1  0 00:53 ?        00:00:00 dcopserver [kdeinit] --nosid
user      3390  3385  0 00:53 ?        00:00:00 klauncher [kdeinit] --new-startup
user      3392     1  0 00:53 ?        00:00:00 kded [kdeinit] --new-startup
.....................................................................................................
user      3756  3748  0 01:27 pts/1    00:00:00 -bash
user      3780  3756  0 01:28 pts/1    00:00:00 ps -ef
```

Процесс, имеющий в команде символы ':0' обозначает, как правило, запуск графической подсистемы *X Window*.

Пользователь может сформировать собственный, удобный для него вывод командой ps. Например, следующая команда выводит имя пользователя, от имени которого запущен процесс, PID процесса, процент использования процессорного времени, время начала исполнения процесса, времени

функционирования процесса, укороченной до 8-ми символов команды запуска процесса. Вывод сортируется по полю используемого процессорного времени (-%cpu – вывод по убыванию значений, %cpu – вывод по возрастанию значений).

ps -eo user,pid,%cpu,stime,etime,fname --sort=-%cpu

Пример вывода:

USER	PID	%CPU	STIME	ELAPSED	COMMAND
root	1282	99.8	12:19	03:16:53	captmon2
igor	2734	22.2	12:21	03:14:48	plugin-c
igor	2155	3.2	12:20	03:15:46	firefox-
root	1457	2.4	12:19	03:16:49	Xorg
igor	5634	1.1	13:00	02:35:26	soffice.
igor	1757	0.9	12:19	03:16:32	pulseaud
igor	1805	0.6	12:19	03:16:30	compiz
igor	18909	0.2	15:30	05:48	chestnut
igor	1784	0.1	12:19	03:16:31	wnck-app
igor	6593	0.0	13:12	02:23:31	gnome-te

…

Следует отметить, что команда ps в Linux-подобных может допускать использование опций BSD-стиля или стиля классических Unix-подобных. Например, может быть удобным использование следующего вида форматированного вывода (AIX формат), который будет легко проанализирован, например, при помощи утилиты awk (пример см. ниже):

ps -eo "%U:%p:%C:%t:%c" --sort=-%cpu

где %U – поле USER; %p – поле PID; %C – поле %CPU; %t – поле ELAPSED; %c – поле COMMAND. Пример вывода:

USER	: PID	:%CPU	: ELAPSED	:COMMAND
root	: 1282	:99.8	: 04:12:32	:captmon2
igor	: 2734	:20.7	: 04:10:27	:plugin-containe
igor	: 2155	: 2.6	: 04:11:25	:firefox-bin
root	: 1457	: 2.2	: 04:12:28	:Xorg

...

Следует отметить, что набор опций команды *ps* отличается между версиями Linux-подобных, BSD-подобных и классическими Unix-подобными.

Для получения более детальной информации используйте команду:

man ps

Другой командой вывода сведений о работающих процессах является команда *top*. Эта команда показывает запущенные процессы и обновляет экран каждые несколько секунд, что позволяет наблюдать за работой компьютера в

реальном времени. Вывод разбит на два раздела. Заголовок (первые пять строк) показывает PID последнего запущенного процесса, среднее значение загрузки системы, время работы системы с последней перезагрузки и текущее время. Другие цифры относятся к количеству запущенных процессов, количеству занятой памяти, памяти подкачки и время, занимаемое различными состояниями процессора. Команда *top* показывает также величину занятой процессором памяти. Команда автоматически обновляет экран каждые две или три секунды. Это значение можно изменить опцией *s*. Выход из режима просмотра команды *top* осуществляется по клавише *q*. По клавише *h* на экран выводится справка об опциях.

Если в ОС установлена графическая оболочка *KDE*, то, как правило, будет доступна программа *KSysGuard*, позволяющая проводить мониторинг запущенных процессов через графический интерфейс. Эта программа может быть запущена командой:

ksysguard &

Часто в KDE для запуска этой программы закреплена последовательность «горячих» клавиш Ctrl+Esc.

В случае установленной графической оболочки *GNOME* мониторинг запущенных процессов через графический интерфейс осуществляют, например, командой:

gnome-system-monitor &

Сигналы. Завершение работы процесса

Взаимодействовать с запущенными процессами, в том числе и с демонами, можно при помощи *сигналов*. Сигналы являются способом передачи от одного процесса другому или от ядра ОС какому-либо процессу уведомления о возникновении определенного события. Сигналы можно рассматривать как простейшую форму межпроцессного взаимодействия. В то же время, сигналы больше напоминают программные прерывания, – средство, с помощью которого нормальное выполнение процесса может быть прервано.

Сигналы имеют специальное значение, многие обрабатываются приложениями, реакция которых на эти сигналы должна быть описана в документации. Пользователь имеет право посылать сигналы только тем процессам, владельцем которых он является.

Два сигнала могут быть использованы для завершения процесса: *SIGTERM* и *SIGKILL*. *SIGTERM* это корректный способ завершить процесс. В некоторых случаях процесс может проигнорировать *SIGTERM*, если выполняет задачу, которая не может быть прервана.

Более детальную справку по сигналам можно получить по команде: *man 7 signal* (для Linux-подобных ОС).

Для удаления процесса используется команда *kill*, отсылающая сигнал процессу. Примеры:

kill -s SIGTERM 7611
kill 7611

где 7611 – PID процесса, который требуется завершить

Некоторые демоны, входящие в состав Unix/Linux, запускаются из *inetd*. *inetd* – это демон (или суперсервер Internet), прослушивающий все порты, используемые другими службами, настроенные на запуск этим демоном, и порождающий копии соответствующих демонов при попытке подключения к ним. Демоны, запускаемые из *inetd*, могут быть отключены путем комментирования соответствующих строк в файле */etc/inetd.conf*. Для быстрого перезапуска *inetd* можно использовать команду (в Linux):

kill -HUP $(cat /var/run/inetd.pid)

Или использовать следующий прием, получая PID демона *inetd* следующим образом:

ps -ef | grep inetd
/sbin/kill -s HUP 7870

где 7870 – это PID демона inetd.

Остальные службы, запускаемые во время загрузки системы, запускаются из скриптов инициализации, размещенных, например в Linux, в каталоге */etc/rc.d*. Их запуск можно отключать тремя способами:

– воспользоваться командами управления скриптов инициализации;

– снять разрешения на выполнение для соответствующего скрипта;

– комментировать в скриптах соответствующие строки (как правило, используется реже всего).

В современных Linux-совместимых ОС в последнее время получили широкое распространение системы инициализации на базе технологий *UpStart* и *Systemd*.

Информация о ФС, монтирование

Ядро Linux является единственным процессом, имеющим непосредственный доступ к аппаратуре – все остальные процессы обращаются к устройствам только через ядро. В ядре Linux можно выделить несколько важных подсистем: подсистему управления памятью, планировщик задач, подсистему VFS – виртуальную ФС и драйверы.

VFS и драйверы ФС являются одной из важнейших составляющих ядра.

При обращении к файлу, в какой-либо ФС, VFS будет переадресовывать обращение на соответствующую функцию, если таковая была зарегистрирована драйвером. Информация о списке обслуживаемых ядром ФС находится в файле */proc/filesystems.* Его просмотр осуществляется командой:

cat /proc/filesystems

Чтобы сделать доступной ФС, расположенную на каком-либо блочном устройстве, используют *операцию монтирования*. Суть операции монтирования заключается в том, что ядро ассоциирует некоторый каталог, называемый точкой монтирования, с блочным устройством и драйвером ФС.

Некоторые ФС не нуждаются в блочном устройстве, поскольку хранят свои данные исключительно в памяти, например, файловая система *procfs*, через файлы которой можно получить доступ к различным системным параметрам и таблицам.

Просмотреть таблицу смонтированных ФС можно через файл */proc/mounts* командами:

cat /proc/mounts

или

mount

Монтирование ФС осуществляется суперпользователем или пользователями, наделенными соответствующими правами при помощи команды *mount*. Для детализации информации об опциях команды mount используют команду:

man mount

Из-за того, что в VFS присутствует понятие кэширования, перед отключением системы необходимо делать обязательный сброс изменений на диск. Сброс кэша на диск осуществляется в момент размонтирования ФС при помощи команды *umount*. Для принудительного сбрасывания на диски всех изменений в любой момент используют команду *sync*.

Размонтировать ФС можно только тогда, когда ни один процесс не удерживает в открытом состоянии файлы с этой ФС, а также в памяти не находится ни один процесс, работающий с каталогом ФС. При неудачном размонтировании может появиться сообщение: *device is busy* (устройство занято).

Команды монтирования и размонтирования

Примеры ФС: EXT2, EXT3, EXT4, UFS, ReiserFS, ZFS, XFS, JFS, NCPFS (для доступа к серверам Novell NetWare), SMBFS, CIFS (для доступа к серверам Windows), NFS (для доступа к ФС Unix/Linux систем), UMSDOS, VFAT, NTFS, HPFS, ISO9660, UDF и другие. Документацию о различных файловых системах,

поддерживаемых, например, в Linux можно взять из каталога:

/usr/src/linux/Documentation/filesystems.

Пример монтирования USB-flash-диска:

ОС FreeBSD 6.2:	*mount -t msdos /dev/da0s1 /mnt/flash*
Linux-подобная ОС:	*mount -t vfat /dev/sda1 /mnt/flash*

Примеры монтирования CD/DVD-диска:

ОС FreeBSD 6.2:	*mount /dev/acd0 /cdrom*
ОС FreeBSD 9.0:	*mount -t cd9660 /dev/cd0 /mnt/dvd*
Linux-подобная ОС:	*mount /dev/hdc /mnt/cdrom*
	mount /dev/sr0 /mnt/cdrom

Пример монтирования дискеты с ФС FAT16, содержащей имена файлов или каталогов в кириллической кодировке для пользователя *user* (Linux-подобная ОС):

mount -t vfat -o iocharset=koi8-r,codepage=1251,uid=user,gid=user /dev/fd0 /mnt/floppy

Пример монтирования NFS (Linux-подобная ОС):

mount -t nfs server:/export/home /mnt/nfs_target

Пример монтирования CIFS (Linux-подобная ОС, команда в одну строку):

mount -t cifs //10.0.0.201/share /home/user/share -o username=user%mypasswd, iocharset=koi8-u, codepage=1251,uid=user,gid=user

Примеры размонтирования:

umount /mnt/floppy
umount /mnt/cdrom
umount /mnt/flash

Детально о процессе автомонтирования ФС в момент загрузки ОС при помощи файла /etc/fstab см. в справке или дополнительной литературе.

Приложение В

Некоторые консольные команды Unix/Linux

Замечание: большинство приведенных ниже команд содержат различные ключи. Кроме того, в различных версиях Unix/Linux-подобных ОС ключи и их значение у команд могут отличаться. Крайне рекомендуется перед выполнением команд с ключами для рекурсивного использования или другими возможностями, ознакомиться с ними при помощи команды: *man команда*.

Команды *ls, ls -l, ls -la, ls -ld, ls -li, ls -i, ls -d*

Команда выводит список файлов и каталогов текущего каталога. Опция *-l* дает возможность выполнить детальный вывод, а опция *-la* дает возможность просмотра информации по скрытым файлам. Опция *-d* позволяет выдавать имена каталогов, как будто они обычные файлы, т.е. без показа их содержимого. Опция *-i* предваряет вывод для каждого файла его номером *inode*. Пример вывода команды *ls -l*

```
drwxr-xr-x      2     root    root    112     2007-04-22 18:08    ifplugd/
-rw-r--r--      1     root    root    4476    2006-10-19 09:38    inetd.conf
 lrwxrwxrwx     1     root    root    12      2007-10-13 10:43    init.d -> rc.d/init.d//

1       2       3       4       5       6           7           8
```

1 – тип файла (см. ниже);
2 – права доступа (6-ть символов, см. ниже);
3 – количество жестких связей;
4 – владелец-пользователь;
5 – владелец-группа;
6 – размер в байтах;
7 – дата и время создания или последней модификации;
8 – имя файла (каталога).

Типы файлов:

b – блочный файл устройства;
c – символьный файл устройства;
d – каталог;
l – файл символической связи;
s – сокет;
p – канал, FIFO;
- – обычный файл.

Права доступа:

rwx rwx rwx
1 2 3

1 – права владельца-пользователя;
2 – права владельца-группы;
3 – права других пользователей;
- – отсутствие прав (каких именно – зависит от знакоместа).
r – право на чтение;
w – право на запись;
x – право на выполнение (или открытие каталога)

Команда *cd*

Команда смены текущего каталога. Примеры:

cd ~
cd $HOME
cd /
cd /usr/bin
cd ..
cd ../..

Команда *pwd*

Команда определения текущего каталога. Пример:

pwd

Команда *which*

Команда отображает полный путь к указанным командам или сценариям. Команда обладает рядом ключей. Например, при помощи ключа *-a* команда выводит все совпадающие исполняемые файлы по содержимому в переменной окружения PATH, а не только первый из них. Примеры:

which java
which -a java

Команда *touch*

Команда изменяет время последнего доступа и/или время последней модификации каждого заданного файла. Если заданный файл не существует, то он создается (если не задана опция *-c*). Например:

touch ./mynewfile

Команда *cp*

Команда копирования файлов (каталогов). С ключом -r выполняет рекурсивное копирование каталога (со всеми его подкаталогами). Примеры:

cp filename newcopyfilename

cp filename1 filename2 filename3 /home/igor/temp
cp filename /home/igor/temp*
cp /etc/samba/ /home/igor/temp*
cp -r /home/user1/docs /mnt/share

Команда *mkdir*

Команда создает новый каталог. Пример:

mkdir mynewdir

Команда *mv*

Команда перемещения/переименования файлов (каталогов). Примеры:

mv filename newfilename
mv filename /home/user1/targetdir

Команда *rmdir*

Удаление пустого каталога. Пример:

rmdir mydeldir

Команда *rm*

Команда удаления файлов/каталогов. С опцией -r выполняет рекурсивное удаление каталога (со всеми подкаталогами). С опцией -f игнорирует имена несуществующих файлов (каталогов) и не выдает интерактивных запросов пользователю. Примеры:

rm filename
*rm filename**
rm -f filename
rm -r dirname

Команда *cat*

Команда вывода содержимого файла на консоль (по умолчанию). Для перенаправления вывода используйте символы '>' (для создания нового файла) или '>>' (для добавления в конец файла). Примеры:

cat filename
cat filename > newfile
cat filename >> addtofile
cat filename1 filename2
cat filename1 filename2 > totalfile

Команда *more*

Команда обеспечивает постраничный вывод на экран. Как правило, применяется с другими командами при организации конвейера команд. Примеры:

more ./readme
ls -l | more
ps -ef | more

cat bigfile | more
more bigfile

Команда *less*

Консольная программа для просмотра содержимого текстовых файлов с возможностью прокрутки. Гораздо быстрее, чем vi, осуществляет просмотр больших файлов. Имеет ряд опций для дополнительных возможностей. Примеры:

less ./readme

Команда *ln*

Команда создает связь (по умолчанию жесткую). С указанием ключа *-s* – символическую связь. Примеры:

ln mysourcefile newlinkfile
ln -s mysourcefile newslinkfile

Команда *chmod*

Команда изменения прав доступа к файлу/каталогу. Примеры:

chmod g+w ./test.txt
chmod g+w,o-r ./test.txt
chmod a+r ./test.txt
chmod a+rwx ./test.txt

Команда *chown*

Команда смены владельца-пользователя (возможно и владельца-группы) файла/каталога. При указании ключа *-R* выполняется рекурсивное изменение прав на все подкаталоги указанного каталога. Примеры:

chown igor ./test.txt
chown -R user2 ./otherdirectory
chown igor:users ./test.txt

Команда *chgrp*

Команда смены владельца-группы файла/каталога. При указании ключа *-R* выполняется рекурсивное изменение прав на все подкаталоги указанного каталога. Примеры:

chgrp users ./test.txt
chgrp -R users ./otherdirectory

Команда *uname*

Команда выводит информацию о системе. С опцией *-a* выводит полную информацию. Пример: *uname -a*

Команда *eject*

Команда извлечения размонтированного CD/DVD-диска из устройства.

Пример:

eject

Команда *startx*

Команда загружает графическую оболочку, если вход в систему был произведен в текстовом консольном режиме и настроены скрипты инициализации графического старта. Оболочка будет загружена при правильной настройке графической подсистемы *X Window*. Пример:

startx

Команда *logout*

Команда завершает работу в сеансе Unix/Linux. Пример:

logout

Команда *reboot*

Команда вызывает перезагрузку системы. Она является укороченной версией команды *shutdown -r*. Пример:

reboot

Команда *halt*

Команда вызывает останов системы (завершает работу ОС и, если это возможно, питание компьютера будет выключено). Она является укороченной версией команды *shutdown -h*. Пример:

halt

Команда *df*

Команда информирует о наличии свободного места на дисковых устройствах. При использовании с ключом *-h* выводит в формате удобном для понимания человеком. Примеры:

df
df -h

Команда *du*

Команда оценивает, сколько занимают места файлы/каталоги на дисках. Команда обладает множеством ключей. Например, при использовании ключа *-h* команда осуществляет вывод в формате, понятном человеку. При использовании ключа *-s* команда выдает только суммарную информацию. При использовании ключа *-c* команда выдает статистику о занимаемом месте каталогов и файлов, а также итоговое суммарное значение. Примеры:

du /etc
du -h /etc
du -hs /etc

du -hc /etc
du -h /etc | sort
du -hsc /usr/ | sort -r*

Команда *free*

Команда информирует о состоянии оперативной памяти и использовании раздела подкачки. При использовании с ключом *-k* выводит объемы в КБ, с ключом *-m* – в МБ, с ключом *-g* – в ГБ. Примеры:

free
free -m

Команда *last*

Команда выводит на экран информацию о дате и времени регистрации в системе пользователя, показывает имя терминала, откуда проводилась регистрация, имя хоста пользователя, время и дату последней регистрации. Терминал с именем :0 обозначает графическую подсистему *X Window*. Примеры:

last
last igor
last tty1
last pts/0

Команда *find*

Утилита поиска файлов в Unix/Linux-подобных ОС. Может производить поиск в одной или нескольких директориях с использованием критериев, заданных пользователем. По умолчанию возвращает все файлы в текущей директории. Использует большое количество разнообразных опций. Наиболее известные опции:

name – поиск по имени файла;

iname – поиск по имени файла без учета регистра;

type – тип искомого: f – файл; d – каталог; l – ссылка (link);

user – владелец (имя пользователя или UID);

group – владелец (группа пользователя или GID);

perm – указывает права доступа;

size – размер (указывается в 512-байтных блоках или байтах, символ 'c' за числом указывает на признак байтов);

atime – время последнего обращения к файлу;

ctime – время последнего изменения владельца или прав доступа к файлу;

mtime – время последнего изменения файла;

delete – удалять найденные файлы;

print – показывает на экране найденные файлы;

exec command {} \; – выполняет над найденным файлом указанную команду;

prune – используется, когда необходимо исключить из поиска

определенные каталоги.

Примеры:

1. Найти все файлы, начиная с текущей директории, название которых начинается на 'my':

find . -name 'my'*

2. Найти все файлы, начиная с корневой директории, название которых начинается на 'my':

find / -name 'my'*

3. Поиск в директориях /usr/local и /opt/local определенных файлов:

find /usr/local /opt/local -name 'my'*

4. Поиск в текущем каталоге файлов 'my*' или (опция -o) файлов 'qu*' (если требуется логическое И, тогда используют опцию -a):

find . \(-name 'my' -o -name 'qu*' \) -print*

5. Вывести все файлы из текущего каталога и ниже, измененные в течение последних 10-ти минут (с расширением '.xml'):

find . -mmin -10
find . -mmin -10 -name '.xml'*

6. Вывести все файлы из текущего каталога и ниже, размер которых превышает 500 МБ:

find . -size +500M

7. Найти файлы жесткой связи с одинаковым значением inode:

find /usr -samefile /usr/bin/gcc
find /usr -inum 1196018

8. Найти файлы-архивы сжатые gzip и вывести их содержимое в файл archives.txt, расположенный в домашнем каталоге пользователя:

find /usr -iname '.tar.gz' -exec file '{}' \; -exec tar -tf '{}' >> $HOME/archives.txt \;*

Команда *grep*

Утилита командной строки, находящая на вводе строки, отвечающие заданному регулярному выражению. Используется как без дополнительных опций, так и с ними. Примеры:

1. Вывод из файла myscript всех строк, содержащих слово 'echo':

grep 'echo' ./myscript

2. Вывод на экран строк, содержащих имя командного процессора Bash:

ps -ef | grep bash

3. Из буфера сообщений ядра, вывести только те, которые имеют отношения к шине PCI с игнорированием регистра символов:

dmesg | grep 'PCI' -i

4. Вывод на экран имен файлов из текущей директории, которые содержат определенное слово:

*grep -l 'echo' **

5. Вывод на экран имен файлов из текущей директории и ниже, которые содержат определенное слово и не являются бинарными:

*grep -r -I -l '<HTML>' **

6. Вывод на экран только строк с комментариями из заданного файла:

grep -e ^[#] ./myscript

7. Вывод на экран всех строк без строк с комментариями из заданного файла:

grep -v -e ^[#] ./myscript

8. Поиск в текущем каталоге и ниже всех текстовых файлов, которые содержат подстроку 'STRING':

find . -type f -name '.txt' -exec grep -i -H 'STRING' {} \;*

Команда *sed*

Утилита командной строки, представляющая потоковый текстовый редактор, применяющий различные предопределенные текстовые преобразования к последовательному потоку текстовых данных. Примеры:

1. Выполнить глобальную замену символа ',' на '.' в определенном файле и вывести результат замены на экран (или в новый файл):

sed -e 's/,/./g' ./geodata.dat
sed -e 's/,/./g' ./geodata.dat > ./geodata2.dat

2. Выполнить глобальную замену слова 'oldstuff' на 'newstuff' в определенном файле и вывести результат замены в новый файл:

sed -e 's/oldstuff/newstuff/g' ./infotable.txt > ./newinfotable.txt

Команда *sort*

Команда предназначена для сортировки строк текстовых файлов. Например, пусть существует файл textfile.txt следующего содержания:

1 Ivanov A.
3 Petrov V.
4 Sidorov M.
2 Ivanov K.

Тогда результат команды *sort ./textfile.txt* будет следующим:

1 Ivanov A.
2 Ivanov K.
3 Petrov V.
4 Sidorov M.

Использование ключа *-b* позволяет игнорировать пробелы в начале сортируемых полей или начале ключей. С ключом *-r* (reverse) сортировка выполняется в обратном порядке. Используя ключ *-u* получают уникальную сортировку (игнорируются повторяющиеся строки). По умолчанию команда выводит результат сортировки в поток стандартного вывода. При использовании ключа *-o* можно указать файл, в который будет произведен вывод результата сортировки. На сортировку влияют установки локали.

Примеры:

sort ./textfile.txt
sort -r ./textfile.txt
sort -o ./sorttextfile.txt ./textfile.txt
du -h /etc | sort

Команды *head* и *tail*

Команды выводят указанное количество строк (по умолчанию 10) с начала указанного файла (head) или с конца (tail). Указать количество выводимых строк можно при помощи опции -n. Примеры:

cat /var/log/messages | tail -n 20
du -sc /usr/share/ | sort -nr | head -n 20*

Команда *tar*

Архиватор. Одним из преимуществ формата tar при создании архивов

является то, что в архив записывается информация о структуре каталогов, о владельце и группе отдельных файлов. Сам tar не создает сжатых архивов, а использует для сжатия внешние утилиты, такие как gzip, bzip2, lzma, lzoop, lzip, 7-zip. Примеры:

1. Архивирование каталога ./tmp в архив tmp.tar.gz и сжатие его архиватором gzip:

tar -czvf ./tmp.tar.gz tmp

2. Распаковка архива в текущий каталог или в заданный каталог:

tar -xzvf ./tmp.tar.gz
tar -xzvf ./tmp.tar.gz -C /home/igor/packtmp

3. Вывод содержимого архива на экран:

tar -tf ./tmp.tar.gz

4. Вывод содержимого архива на экран с информацией о правах и атрибутах:

tar -tvf ./tmp.tar.gz

5. Архивирование с использованием 7-zip:

tar -cvf - ./tmp | 7za a -si ./tmp.tar.7z

6. Распаковка с использованием 7-zip:

7za x -so ./tmp.tar.7z | tar xf -

7. Вывод содержимого tar.7z-архива на экран:

7za x -so ./tmp.tar.7z | tar -tf -

Команды *lscpu, lspci, lsusb, lsmod, lsblk, lsof*

Команда *lscpu* показывает информацию об архитектуре процессора. *lspci* – показывает список всех устройств на шине PCI. *lsusb* –показывает список всех USB-устройств. *lsmod* –показывает статус модулей ядра Linux. *lsblk* –выводит список блочных устройств. *lsof* –выводит список открытых файлов. Примеры:

lscpu
lspci
lspci -tv
lsusb
lsusb -tv

lsmod
lsmod | more
lsblk
lsof
lsof | more
lsof | grep user | more

Команда *modinfo*

Команда выводит информацию об указанном модуле ядра. Например:

modinfo video

Команда *awk (иногда отсутствует в дистрибутивах по умолчанию)*

Awk – утилита, предназначенная для простых вычислительных манипуляций над данными. Это полноценный интерпретируемый скриптовый язык обработки текстовой информации с С-подобным синтаксисом. Он обладает достаточно большими возможностями. Далее будут рассмотрены только некоторые из них.

Awk обрабатывает каждую строку входного потока и разбивает их на поля, которые в тексте отделены друг от друга пробельными символами. В качестве разделителей могут быть использованы и другие символы. Awk анализирует и обрабатывает каждое поле в отдельности, что делает его идеальным инструментом для работы со структурированными текстовыми файлами, например, представляющих информацию в табличном виде.

Внутри сценариев командной оболочки, код awk заключается в одинарные кавычки и фигурные скобки. Например, пусть имеем файл textfile.txt с содержимым, рассмотренным выше в команде sort. Тогда следующая команда выведет на консоль только фамилии людей (содержимое второго поля), перечисленных в этом файле:

awk '{print $2}' ./textfile.txt

При формировании вывода данных полей можно использовать спецсимволы форматирования языка С. Например, использование табуляций:

awk '{print $2 "\t\t" $3}' ./textfile.txt

Представленная команда выводит содержимое второго поля, затем генерирует два символа табуляции и затем выводит данные третьего поля файла textfile.txt.

Таким образом, $1 – первое поле, $2 – второе поле и т. д. Для ссылки на всю строку целиком используется $0. Строка может содержать до 100 полей. Строка может содержать максимально до 256 символов.

Внутри awk могут использоваться переменные (как числовые, так и строковые). Например, следующий фрагмент кода показывает использование некоторой переменной *i*, являющейся номером строки, и выводимой совместно с данными второго и третьего полей:

awk '{i++; print i "\t\t" $2 "\t\t" $3}' ./textfile.txt

Конструкция в фигурных скобках выполняется каждый раз для новой строки входного потока (или файла), однако значение переменной *i* сохраняется, и каждый раз увеличивается. Изначальное значение переменной равно нулю.

Если нужно вывести в поток вывода информацию после обработки, то используют так называемый селектор END. В следующем примере значение переменной *i* будет выведено после вывода всех данных поля 2 и 3 – значение будет соответствовать количеству обработанных строк файла textfile.txt:

awk '{i++; print $2 "\t\t" $3} END { print "COUNT = " i }' ./textfile.txt

Такой же результат даст следующая команда:

awk '{ print $2 "\t\t" $3 } END { print "COUNT = " NR }' ./textfile.txt

NR – это предопределенная в awk переменная, которая равна номеру обрабатываемой записи (строки).

Часто удобно совместить вывод с командой echo, чтобы сформатировать заголовок выводимой информации. Например:

echo -e "Number\t\tName" ; awk '{ print $1 "\t\t" $2 " " $3 }' ./textfile.txt

Чтобы выполнить какие-либо действия до начала анализа строк, используется селектор BEGIN. Например, преобразованная предыдущая команда может быть записана следующим образом:

awk 'BEGIN { print "Number\t\tName" } { print $1 "\t\t" $2 " " $3}' ./textfile.txt

Следующая команда устанавливает в качестве разделителя полей символ *':'*.

ps -e | awk '{ print $3 }' | awk 'BEGIN { FS = ":" } { print $3 }'

Вначале осуществляется вывод команды *ps -e*. Его результат (значение поля TIME – третье поле) подается на вход утилиты awk. Результат обработки – только значение этого поля, подается для обработки опять на вход awk и теперь в качестве разделителя используется символ *':'*. Итогом выполнения будет извлечение поля секунд (тоже третье поле) из поля TIME команды *ps -e*.

Отметим, что для установки символа-разделителя полей можно использовать ключ *-F* утилиты awk. Таким образом, предыдущая команда эквивалентна следующей:

ps -e | awk '{ print $3 }' | awk -F: '{ print $3 }'

При использовании утилиты можно использовать операторы управления, такие как: *if*, *while*, *for* и другие. Например, необходимо вывести на консоль информацию о первых 20-ти процессах, потребляющих наибольшее количество процессорного времени и работающих более 3-х часов в системе:

```
ps -eo "%U:%p:%C:%t:%c" --sort=-%cpu | \
head -n 20 | \
awk -F: '{ if(NR==1) print $0; if(NF==7 && $4>3) print $0 }'
```

где внутренняя переменная NF показывает количество полей в записи (строке).

Пример использования в awk *for* для форматированного вывода:

```
ps -eo user,pid,time,stime,etime,comm | \
head | \
awk '{ for(i=1;i<NF;i++) { s=s$i"|\t" } { s=s$NF } { print s; s="" } }'
```

Часто awk удобно использовать совместно с командой sort. Например:

```
du -sc /usr/share/* | sort -nr | head | awk '{ print $1/1024 " MB\t" $2 }'
```

Приведенная команда позволяет вначале определить объем дискового пространства, занимаемого каталогами, которые расположены в каталоге /usr/share (объем выводится в килобайтах). Затем список сортируется в порядке убывания подсчитанных значений, а затем выводятся только первые девять наиболее больших каталогов и в качестве первой строки итоговое значение. Эта информация подается на вход утилиты awk, которая преобразует значения в удобные для понимания человеком (пересчитывает в мегабайты) с выводом аббревиатуры.

Более детально про использование awk см. по следующим ссылкам:
http://lib.ru/MAN/DEMOS210/awk.txt
http://www.ibm.com/developerworks/ru/library/l-awk1/index.html
http://ru.wikipedia.org/wiki/AWK

Гаркуша Ігор Миколайович

ОПЕРАЦІЙНІ СИСТЕМИ

Методичні вказівки та завдання
до лабораторних робіт
для студентів напряму підготовки
6.050101 Комп'ютерні науки

(Російською мовою)

Видано в авторській редакції

Підп. до друку 05.04.2013. Формат 30х42/4.
Папір офсет. Ризографія. Ум. друк. арк. 4,7.
Обл.-вид. арк. 5,6. Тираж 50 пр. Зам. №

Державний ВНЗ “Національний гірничий університет”
49005, м. Дніпропетровськ, просп. К. Маркса, 19.