

СОДЕРЖАНИЕ

Введение	4
Раздел 1. Введение в ОС Linux.	5
1.1. Основные понятия и определения.	5
1.2. Основные команды, конкатенация файлов.	6
1.3. Универсальный инструмент поиска <code>find</code>	10
1.4. Редактор <code>vi</code> – основные команды и их взаимосвязи.	15
Контрольные вопросы	18
Раздел 2. Регулярные выражения как средства текстового поиска.	19
2.1. Основные понятия и возможности.	19
2.2. Создание шаблонов для поиска выражений, стоящих в начале или в конце строки.	19
2.3. Создание шаблонов для поиска символов, встречающихся неопределенное число раз	21
2.4. Создание шаблонов для поиска специальных символов	22
2.5. Создание шаблонов для поиска символов из указанного набора или диапазона	22
2.6. Поиск символов, встречающихся заданное число раз	23
2.7. Инструментальное средство <code>grep</code>	24
Контрольные вопросы.	31
Раздел 3. Простейшие средства shell.	32
3.1. Пользовательская среда UNIX.	32
3.2. Командный интерпретатор shell.	32
3.3. Приемы экранирования.	34
3.4. Организация циклов и ветвлений.	35
3.5. Функции интерпретатора shell.	37
3.6. Применение shell-сценариев	43
Контрольные вопросы.	44
Раздел 4. Установка и обновление программного обеспечения в ОС Linux	46
4.1. Основные способы установки программного обеспечения.	46
4.2. Tar-gz-архивы и rpm-пакеты: понятие, создание и распаковка.	50
4.3. Компиляция ПО из исходных текстов.	54
4.4. Основные компоненты GNU- компилятора	55
4.5. Инсталляция пакетов ПО из исходных текстов	56
Контрольные вопросы.	57
Раздел 5. CRON-ДЕМОН.	58
5.1. Понятие демона в Linux-понимании.	58
5.2. Понятие <code>cron</code>	59
5.3. Создание файла <code>crontab</code> для пользователя.	62
5.4. Журнал событий <code>cron</code> -демона.	63
Контрольные вопросы.	63
Раздел 6. Администрирование и установка web-сервера	64
6.1. Директивы, используемые для конфигурирования web-сервера.	64
6.2. Настройка виртуальных серверов в файле <code>httpd.conf</code>	70
6.3. Инструмент настройки Apache.	71
6.4. Протоколирование.	72
6.5. Проверка работоспособности web-сервера.	74
6.6. Инсталляция SAMS	75
Контрольные вопросы.....	77
Список используемой литературы	78
Итоговый тест	79

ВВЕДЕНИЕ

Настоящее учебное пособие строится на материале, который читается студентам специальности 230101 в первой части дисциплины «Системное программное обеспечение» - операционная система UNIX. В ней рассматриваются основные понятия ОС Linux, принципы построения и инструментарий.

В 1984 году американский ученый Ричард Столлман (Richard Stallman) основал Фонд Свободного Программного Обеспечения (Free Software Foundation). Целью этого фонда было устранение всех запретов и ограничений по распространению, копированию, модификации и изучению программного обеспечения. В рамках Фонда Свободного Программного Обеспечения была начата разработка проекта GNU - проекта создания свободного программного обеспечения. Аббревиатура GNU - GNU's Not Unix означает то, что принадлежит проекту GNU, не является частью Unix. В рамках проекта GNU уже было разработано большое количество утилит разного рода. Но для превращения GNU в полноценную ОС не хватало ядра. Поэтому появление разработки Л. Торвальдса было очень своевременным. Она представляла собой только ядро операционной системы. Это ядро "упало на подготовленную почву", т.е. ознаменовало рождение операционной системы, распространяемой с открытыми исходными кодами.

Linux начала свое развитие с небольшого набора программ и за эти годы переросла в гибкую ОС, использующуюся для работы огромного количества программных сред и приложений. С момента выхода первой версии ядра в мире уже существует несколько десятков различных дистрибутивов Linux.

Учебное пособие рассматривает основы работы, принципы и технологии (инструментарий) в ОС Linux, базовые команды, в том числе команды поиска файлов с определенными характеристиками, основные команды редактора vi и их взаимосвязи. Во втором разделе рассмотрены регулярные выражения как средства текстового поиска, примеры создания шаблонов для поиска символов.

Третий раздел содержит комплекс упражнений с использованием командного языка shell. В четвертом разделе рассмотрены основные способы установки программного обеспечения, компоненты GNU- компилятора.

Пятый раздел полностью посвящен изучению cron-демона в Linux – пониманию. Вопросам администрирования и установки web-сервера посвящен шестой раздел. Приводятся тестовые задания.

Пособие рассчитано на студентов специальности 230101 и слушателей высших учебных заведений, обучающихся по техническим дисциплинам. Может быть использовано при проведении практических занятий по дисциплине «Системное программное обеспечение», курсовом проектировании и самостоятельном решении задач разработки современного системного программного обеспечения.

РАЗДЕЛ 1. ВВЕДЕНИЕ В ОС LINUX

1.1. Основные понятия и определения

Linux - операционная система, имеющая открытый исходный код, создавалась как часть проекта GNU, распространяется она по лицензии GPL.

Возникновение Linux связывают с 1991 годом, когда молодой финский программист Линус Торвалдс (от его имени и происходит название Linux) начал работу над первой версией системы. После своего возникновения (и до настоящего времени) Linux переживает настоящий расцвет популярности. Связано это в первую очередь с тем, что ядро операционной системы Linux, как и подавляющее большинство написанных под неё программ обладают двумя важными отличительными признаками: бесплатность и открытость исходного кода.

В настоящее время вокруг Linux сложилось гигантское сообщество программистов, которые постоянно занимаются совершенствованием Linux, разработкой новых версий и разновидностей операционной системы, написанием самых разнообразных программ, работающих под Linux. Особенно сильны позиции этой операционной системы в области серверного программного обеспечения - подавляющее большинство серверов Internet работают именно под управлением операционных систем Linux.

Перечислим основные свойства операционной системы Linux:

1) *настоящая многозадачность* - система устроена так, что под каждую задачу, выполняемую пользователем, выделяется определенное количество ресурсов. Ресурсы компьютера, такие как, например, оперативная память, не передаются приоритетной задаче (как это делается в Windows), а используются параллельно несколькими приложениями. Это повышает производительность системы и снижает риск ее «зависания». В случае отказа приложения выполнять команды оно не мешает работать и не «утягивает» за собой всю систему. Кроме того, «зависшую» задачу почти всегда можно снять командой *kill*;

2) *поддержка различных типов файловых систем* – на компьютере параллельно с Linux можно установить еще несколько операционных систем на одном жестком диске, причем данные каждой из них будут доступны из Linux;

3) *поддержка различных аппаратных платформ* - система может функционировать как на IBM-совместимом компьютере с процессорами большинства производителей - Intel, AMD, Via, так и на компьютерах с другими процессорами, например, ARM (основой некоторых карманных компьютеров, что позволило использовать Linux, например, в КПК Sharp Zaurus и его модификациях), Sun Sparc и других;

4) *невысокие системные требования* - минимальным системным требованиям для Linux удовлетворяет компьютер с процессором Intel 386 и 4 мегабайтами оперативной памяти. Однако в данном случае работа с Linux будет аналогична работе в DOS и осуществляется только из командной строки.

Для запуска файловой оболочки, например, Midnight Commander, потребуется уже 8 Мбайт памяти. Интересно, что для работы в графическом режиме X Window достаточно процессора Intel 486 и 4 мегабайт ОЗУ. Тем не менее, для работы в интегрированной среде KDE или Gnome необходимо 32 и более мегабайта оперативной памяти.

Сторонники Linux утверждают, что программное обеспечение с открытым кодом - это будущее экологичных информационных технологий. Одной из таких сторонниц является Tina Gasperson из журнала "Datamation". Она считает, что использование операционных систем с открытым кодом доступа очень экологично, и те компании, которые используют ОС Linux, сильно продвинулись в "озеленении" своих департаментов. Она выделяет следующие преимущества: за счет виртуализации серверов экономится электроэнергия, потребляемая компанией, работающей на ОС Linux, и рабочее пространство; документация по программному обеспечению (если она вообще есть) распространяется в электронном, а не бумажном виде, что позволяет экономить бумагу и сохранять леса; Linux, по сравнению с другими ОС, требует менее мощное оборудование для работы, что позволяет дополнительно сократить расходы на электроэнергию и на постоянное обновление "железа"; программное обеспечение с открытым кодом каждый может адаптировать "под себя".

1.2. Основные команды, конкатенация файлов

Ввод команд в Linux выглядит примерно так же, как в DOS и других операционных системах, ориентированных на ввод в командной строке. ОС Linux, как и UNIX, чувствительна к регистру, поэтому если система не воспринимает какую-либо команду, следует проверить, в правильном ли регистре ввели ее.

1.2.1. Вызов истории команд

В ОС Linux есть средство повторного обращения к уже выполненным командам, которое не прерывается даже при выключении компьютера. Предыдущая команда вызывается после нажатия клавиши <Up>, а для ее выполнения надо нажать <Enter>. Для вывода всего списка примененных команд следует воспользоваться командой *history*:

```
$ history
```

Результат выполнения команды:

```
1 clear
```

```
2 adduser
```

```
3 history
```

Чтобы выполнить команду из хронологического списка, необходимо перелистывать с помощью клавиши <Up> предыдущие команды до тех пор, пока в командной строке не появится искомая команда, или же нажать клавишу с восклицательным знаком <!> и ввести номер нужной команды. Например, чтобы повторно выполнить команду *adduser* из представленного выше списка, введите команду:

\$!2

Максимальное число команд в хронологическом списке задается в пользовательском конфигурационном файле *.profile*.

1.2.2. Основные команды Linux

Команда справки *man*

Для получения справки по той или иной команде Linux используется команда *man*, и далее можно пролистать предложенные разделы. Если точное имя нужной команды забыто, следует ввести команду *man* с параметром *-k*, затем ключевое слово для поиска нужной команды. Система выполнит поиск в файлах справки, содержащей это ключевое слово. Для этой команды имеется также псевдоним *apropos*. Например, если ввести команду *man ls*, на экран будет выведена справка о команде *ls*, в том числе обо всех ее параметрах. По команде *man -k cls* выводится список всех команд, в справке о которых есть слово *cls*. Команда *apropos cls* аналогична команде *man -k cls*.

Команды перезагрузки и останова системы

Перезагрузить компьютер можно с помощью команды *reboot*. Для прекращения работы Linux также используются команды *halt*, *fasthalt* *fastboot*. Все названные команды представляют собой короткий вариант команды *shutdown* с определенными параметрами:

```
halt      - shutdown -h now
fasthalt  - shutdown -fh now
fastboot  - shutdown -fr now
reboot    - shutdown -r now
```

Параметры команды *shutdown* означают следующее:

-f - создать файл */fastboot* и при следующей загрузке компьютера пропустить тестирование файловой системы;

-h - остановить систему;

-r - перезапустить систему.

Команды для работы с каталогами

В Linux есть множество команд для работы с каталогами. Как и в других операционных системах, каталоги в Linux можно удалять, создавать, перемещать, а также выводить информацию об их состоянии. В Linux, как и в DOS, файлы хранятся в каталогах, организованных в древовидные структуры. Файл можно указывать в виде пути из корневого каталога, обозначаемого символом */*, до файла. Таким образом, полное имя файла *new*, принадлежащего пользователю *sanja*, может иметь вид */home/sanja/new*. В Linux есть понятие рабочего каталога пользователя. Рабочий каталог обычно обозначается символом *~* (тильда). Например, команда копирования файла из текущего каталога в рабочий может иметь вид *cp new ~*

Смена текущего каталога (*cd*)

Для перемещения по дереву каталогов Linux применяется команда *cd*. Для перехода в рабочий каталог эта команда вводится без параметров. Для перехода из одного каталога в другой формат команды тот же, что и DOS: *cd*

new-directory, где *new-directory* – имя каталога, в который следует перейти. Кроме того, в Linux текущий каталог представляется одной точкой (.), каталог-родитель – двумя (. .) – и, конечно же, в этом DOS наследует UNIX и Linux, а не наоборот.

Следует обратить внимание на символ разделителя каталогов. В DOS для этого применяется обратная косая черта (\), которая в Linux служит указателем продолжения команды с новой строки. В Linux каталоги разделяются прямой косой чертой (/). Кроме того, в DOS не имеет значения, отделены ли параметры (.) и (. .) пробелами от имени команды, в то время как в Linux это важно. Например, команда *cd..* не будет распознана командным интерпретатором, правильный формат которой – *cd ..* В Linux между командой и параметром обязательно должен быть пробел.

Вывод информации о файлах (ls)

Команда *ls* – сокращение от *list* (список) – выводит на экран список файлов. Это аналог команды *dir* из DOS (которую можно применять и в Linux) для вывода списка файлов в каталоге. Синтаксис команды: *ls [opt] [file1 file2 ...]*, где в качестве параметров можно задать имена каталогов, содержимое которых нужно вывести, или имена файлов, информацию о которых нужно получить. Опции команды позволяют получить список дополнительной информации. Чаще всего применяется параметр *-l*, по которому выводится полная информация о каждом файле. Параметры можно объединять, так по команде *ls -al* выводится подробная информация о скрытых файлах текущего каталога. По команде *ls new* выводится только имя этого файла, по команде же *ls -l new* – полная информация о нем.

Создание каталога (mkdir)

Поскольку структура каталогов составляет основу файловой системы, в Linux имеется также команда создания каталога *mkdir*. В отличие от DOS, где можно воспользоваться псевдонимом данной команды *md*, в Linux надо вводить ее полное имя. Синтаксис команды: *mkdir [параметр]*, где в качестве параметра указывается имя создаваемого каталога, например: *mkdir /home/new*.

Удаление каталогов (rmdir)

Удаление файлов в ОС Linux осуществляется при помощи команды *rm*, имеющей следующий синтаксис: *rm [опции] файлы*, где *опции* – разнообразные параметры, позволяющие управлять процессом удаления; *файлы* – объекты файловой системы, которые необходимо удалить. Чтобы удалить файл при помощи команды *rm*, пользователь должен иметь разрешение на каталог, содержащий данный файл, однако ему не обязательно иметь разрешение для самого файла, который он хочет удалить. Наиболее часто используемыми при работе с командой *rm* являются следующие опции:

-f (force) – используется для принудительного удаления файлов без вывода запросов на подтверждение пользователя;

-i (interactive) – используется для подачи запроса подтверждения пользователю перед удалением файлов;

-r (recursive) - используется для рекурсивного удаления содержимого каталога и самого каталога.

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

Команды работы с файлами

В Linux каталоги являются одним из типов файлов, поэтому для работы с теми и другими применяются одни и те же команды.

Копирование файлов с помощью команды (cp)

Команда *cp* аналогична команде *COPY* DOS. Она применяется для копирования одного или нескольких файлов из одного каталога в другой. Синтаксис команды: *cp from-filename to-filename*, где *from-filename* - исходный файл; *to-filename* - файл, в который происходит копирование. Чтобы скопировать файл с тем же именем в качестве второго параметра, ставится точка (.).

Конкатенация файлов (cat)

Утилита *cat* выполняет слияние и вывод файлов: по очереди читает указанные файлы и выдает их содержимое на стандартный вывод. Так, например, *cat file* распечатывает содержимое файла *file*, а *cat file1 file2 > file3* объединяет первые два файла и помещает результат в третий. Чтобы добавить файл *file1* к файлу *file2*, надо выполнить команду *cat file1 >> file2*.

Синтаксис команды: *cat [-опция] файл* - где в качестве опции указываются различные ключи, например, использование ключа *-v* целесообразно при просмотре нетекстового файла. В этом случае вывод “непечатных” символов, которые могут нарушить настройки терминала, будет подавлен. Если имя файла в командной строке не указано, то ожидается вывод - ввод данных из стандартного потока ввода.

Перемещение файлов (mv)

По команде *mv*, аналогичной команде *MOVE* из DOS, файлы перемещаются из одного каталога в другой. Действие этой команды аналогично действию команды копирования с последующим удалением исходных файлов. Команда *mv* не создает копий файлов. Синтаксис команды: *mv from-filename to-filename*, где *from-filename* - исходный файл; *to-filename* - новый файл.

Вывод содержимого файла (more)

По команде *more* на экран выводится содержимое текстового файла, при этом нет необходимости запускать текстовый редактор, распечатывать файл или нажимать клавишу паузы во время вывода текста на экран. Например, для вывода на экран содержимого конфигурационного файла *naw* вводится команда: *more etacs*. Недостаток этой команды в том, что невозможно пролистать информацию в обратном направлении.

Команда less

По команде *less* информация выводится в окно терминала. Имя этой

команде дано в противоположность команде *more*, поскольку в команде *less* пролистывание текстового файла возможно в обоих направлениях (игра слов: *more* - больше, *less* - меньше.). Синтаксис команды: *less имя_файла*.

1.3. Универсальный инструмент поиска *find*

Часто в процессе работы возникает необходимость осуществить поиск файлов с определенными характеристиками, такими как права доступа, размер, тип и т.д. Команда *find* представляет собой универсальный инструмент поиска: она позволяет искать файлы и каталоги, просматривать все каталоги в системе или только текущий каталог, проводить поиск даже на дисках NFS (Network File System), конечно, при наличии соответствующих разрешений. В подобных случаях команда обычно выполняется в фоновом режиме, поскольку просмотр дерева каталогов требует значительных затрат времени.

Общий формат команды: *find имя_каталога [-ключ]*, где *имя_каталога* - это каталог, с которого нужно начинать поиск файла в файловой системе, используя различные критерии. Символ '.' служит для обозначения текущего каталога, символ '/' - корневого каталога, а символ '~' - записанного в переменной \$HOME начального каталога текущего пользователя. Основные опции команды *find* представлены в табл. 1.1.

Таблица 1.1. Основные опции команды *find*

ключ	назначение
1	2
name	поиск файлов, имена которых соответствуют заданному шаблону
print	запись полных имен найденных файлов в стандартный поток вывода
perm	поиск файлов, для которых установлен указанный режим доступа
prune	применяется для того, чтобы команда <i>find</i> не выполняла рекурсивный поиск по уже найденному путевому имени; если указана опция <i>-depth</i> , опция <i>-prune</i> игнорируется
user	поиск файлов, принадлежащих указанному пользователю
group	поиск файлов, которые принадлежат данной группе
mtime <i>-m+n</i>	поиск файлов, содержимое которых модифицировалось менее чем (-) <i>m</i> или более чем (+) <i>n</i> дней назад; имеются также опции <i>-atime</i> и <i>-ctime</i> , которые позволяют осуществлять поиск файлов соответственно по дате последнего чтения и дате последнего изменения атрибутов файла
nogroup	поиск файлов, принадлежащих несуществующей группе, для которой, иначе говоря, отсутствует запись в файле <i>/etc/groups</i>

Продолжение табл.1.1

1	2
nouser	поиск файлов, принадлежащих несуществующему пользователю, для которого, другими словами, отсутствует запись в файле /etc/passwd
newer <i>файл</i> –type	поиск файлов, которые созданы позднее, чем указанный файл
<i>b</i> <i>d</i> <i>c</i> <i>p</i> <i>l</i> <i>s</i> <i>f</i>	поиск файлов определенного типа, а именно: специальный блочный файл; каталог; специальный символьный файл; именованный канал; символическая ссылка; сокет; обычный файл
size <i>n</i> <i>b</i> <i>c</i> <i>k</i> <i>w</i>	поиск файлов, размер которых составляет <i>n</i> единиц; возможны следующие единицы измерения: блок размером 512 байтов (установка по умолчанию); байт; килобайт (1024 байта); двухбайтовое слово
depth	при поиске файлов сначала просматривается содержимое текущего каталога и лишь затем проверяется запись, соответствующая самому каталогу
fstype	поиск файлов, которые находятся в файловой системе определенного типа; обычно соответствующие сведения хранятся в файле /etc/fstab
mount	поиск файлов только в текущей файловой системе; аналогом этой опции является опция –xdev
exec	выполнение команды интерпретатора shell для всех обнаруженных файлов; выполняемые команды имеют формат команда {} \; (следует обратить внимание на наличие пробела между символами {} и \;)
ok	аналогично exec, но перед выполнением команды отображается запрос

Примеры использования опций команды find

1) опция -name

При работе с командой *find* чаще всего используется опция -name. После нее в кавычках должен быть указан шаблон имени файла. Если необходимо найти все файлы с расширением txt в Вашем начальном каталоге, укажите

символ '-' в качестве путевого имени. Имя начального каталога будет извлечено из переменной \$HOME:

```
find ~ -name "*.txt" -print
```

Чтобы найти *все* файлы с расширением txt, находящиеся в текущем каталоге, следует воспользоваться такой командой:

```
find . -name "*.txt" -print
```

Для нахождения в текущем каталоге всех файлов, в именах которых встречается хотя бы один символ в верхнем регистре, введите следующую команду:

```
find . -name "[A-Z]*" -print
```

Команда *find . -print* аналогична команде *ls -Rfl*, но в последнем случае выводимый список будет длиннее, т.к. в процессе обхода команда *ls* отмечает каждый новый каталог, а команда *find* не обращает внимание на каталог ..

Найти в каталоге */etc* файлы, имена которых начинаются с символов "host", позволяет команда:

```
find /etc -name "host*" -print
```

Опция *-o* является обозначением операции логического ИЛИ. В случае ее применения помимо файлов с обычными именами будут найдены файлы, имена которых начинаются с точки.

Получить список всех файлов в системе, не имеющих расширения, можно, набрав представленную ниже команду (предупреждение: данная команда может существенно замедлить работу системы):

```
find / -name "*" -print
```

Найти все файлы, в именах которых сначала следуют символы нижнего регистра, а за ними две цифры и расширение .txt (например, ax37 .txt), можно, используя команду:

```
find . -name "[a-z][a-z][0-9][0-9].txt" -print
```

2) опция *-perm*

Опция *-perm* позволяет находить файлы с заданным режимом доступа. Например, для поиска файлов с режимом доступа 755 (их может просматривать и выполнять любой пользователь, но только владелец имеет право осуществлять запись) следует воспользоваться такой командой:

```
find . -perm 755 -print
```

Если перед значением режима вставить дефис, будет произведен поиск файлов, для которых установлены все указанные биты разрешений, остальные биты при этом игнорируются. Например, следующая команда осуществляет поиск файлов, к которым другие пользователи имеют полный доступ:

```
find . -perm -007 -print
```

Если же перед значением режима введен знак "+", осуществляется поиск файлов, для которых установлен хотя бы один из указанных битов разрешений, при этом остальные биты игнорируются.

3) опции *-user*, *-nouser*

Для поиска файлов, принадлежащих определенному пользователю, необходимо в команду *find* включить опцию *-user*, указав через пробел имя пользователя. Например, поиск в начальном каталоге файлов, принадлежащих пользователю *stud*, осуществляется посредством такой команды:

```
find ~ -user stud -print
```

Поиск в каталоге */etc* файлов, принадлежащих пользователю *uusr*, выполняет следующая команда:

```
find /etc -user uusr -print
```

Благодаря опции *-nouser* возможен поиск файлов, принадлежащих несуществующим пользователям. При ее использовании производится поиск файлов, для владельцев которых нет записи в файле */etc/passwd*. Конкретное имя пользователя указывать не нужно. Например, требуется найти все файлы, которые принадлежат несуществующим пользователям и находятся в каталоге */home*:

```
find /home -nouser -print
```

4) опции *-group* и *-nogroup*

Опции *-group* и *-nogroup* аналогичны опциям *-user* и *-nouser* и позволяют искать файлы, принадлежащие заданной группе или несуществующим группам. Ниже приведена команда для нахождения в каталоге */vmkss* всех файлов, которыми владеют пользователи группы *vmkss*:

```
find /vmkss -group vmkss -print
```

Нижеприведенная команда ищет во всей системе файлы, принадлежащие несуществующим группам:

```
find / -nogroup -print
```

5) опция *-type*

Операционные системы UNIX и Linux поддерживают различные типы файлов. Поиск файлов определенного типа осуществляется посредством команды *find* с опцией *-type*. Например, для нахождения всех подкаталогов в каталоге */etc* воспользуемся командой:

```
find /etc -type d -print
```

Чтобы получить список всех файлов, но не каталогов, необходимо ввести следующую команду:

```
find . ! -type d -print
```

Ниже приведена команда, которая предназначена для поиска всех символических ссылок в каталоге */etc*:

```
find /etc -type l -print
```

6) опция *-size*

В процессе поиска размер файла указывается с помощью опции *-size N*, где *N* - размер файла в блоках по 512 байтов. Возможные аргументы имеют следующие значения; *+N* - поиск файлов, размер которых больше заданного, *-N* - меньше заданного, *N* - равен заданному. Если в аргументе дополнительно указан символ *c*, то размер считается заданным в байтах, а не в блоках, а если символ *k* - в

килобайтах. Например, команда для поиска файлов, размер которых превышает 1 Мб: *find . size +1000k -print*. Поиск в каталоге */home/apache* файлов, размер которых в точности равен 100 байтам:

```
$ find /home/apache -size 100c -print
```

7) опция *-mount*

Поиск файлов только в текущей файловой системе, исключая другие смонтированные файловые системы, обеспечивает опция *-mount* команды *find*. Например, поиск всех файлов с расширением *tp* в текущем разделе диска:

```
$ find / -name "*.tp" -mount -print
```

Команда *umask*

Когда пользователь регистрируется в системе, команда *umask* устанавливает стандартный режим доступа к создаваемым файлам и каталогам. Как правило, значение *umask* устанавливается в файле */etc/profile*, доступ к которому имеют все пользователи. Поэтому, если вы хотите установить общесистемное значение *umask*, отредактируйте данный файл (для этого нужно иметь права администратора). Свое собственное значение *umask* можно задать в файле *.profile* или *.bash_profile*, находящемся в каталоге */home*.

Команда *umask* задает восьмеричное число, которое при создании каждого файла и каталога вычитается из стандартного значения режима доступа. Полученное значение режима присваивается файлу или каталогу. Стандартному режиму доступа к каталогам соответствует число 777, а режиму доступа к файлам — 666 (система не позволяет создавать текстовые файлы с установленными битами выполнения, эти биты следует добавлять отдельно с помощью команды *chmod*). Значение *umask* также состоит из трех трехбитовых наборов: для владельца, группы и других пользователей.

Общий формат команды *umask*:

```
umask nnn
```

где *nnn* - маска режима в диапазоне от 000 до 777.

Для просмотра текущего значения *umask* следует ввести команду *umask* без параметров:

```
umask
```

На основании значения *umask* можно определить режим доступа к файлу или каталогу:

```
$ umask
```

```
022
```

Цифра в значении <i>umask</i>	Результат для файла	Результат для каталога
0	6	7
1	6	6
2	4	5
3	4	4
4	2	3
5	2	2
6	0	1
7	0	0

Например, значению `umask`, равному 002, соответствует режим 664 для файлов и 775 для каталогов. Ряду пользователей удобнее работать со строками режима, руководствуясь описанной ниже последовательностью действий. Предположим, значение `umask` равно 002:

- с начала записать полную строку режима, эквивалентную числу 777;
- под ней записать строку режима, соответствующую значению `umask` (002);
- вычеркнуть из первой строки те символы, которые дублируются в тех же позициях во второй строке, получится строка режима для каталогов;
- вычеркнуть из полученной строки все символы `x`, получится строка режима для файлов.

Примеры установки значений `umask`:

Значение <code>umask</code>	Режим доступа к каталогам	Режим доступа к файлам
022	755	644
027	750	640
002	775	664
006	771	660
007	770	660

Для просмотра текущего значения `umask` введите команду `umask` без параметров:

```
$ umask
022
$ touch file1
$ ls -l file1
-rw-r--r-- 1 nan admin 0 Apr 07 22:05 file1
```

Для изменения существующей установки необходимо указать новый аргумент команды `umask`:

```
$ umask 002
```

Для проверки того, что система приняла изменения, вводим команды:

```
$ umask
002
$ touch file2
$ ls -l file2
-rw-rw-r-- 1 nan admin 0 Apr 07 22:07 file2
```

1.4. Редактор `vi` – основные команды и их взаимосвязи

В любой операционной системе самой нужной программой является простой текстовый редактор, с помощью которого можно откорректировать, например, конфигурационные файлы. В Windows - это редактор Блокнот или встроенный редактор файлового менеджера. В Linux наиболее употребляемыми для таких нужд программами являются текстовый редактор `vi` и его потомки.

На ранних этапах развития Unix-подобных систем было много аналогичных vi программ, которые являлись гибридом файлового менеджера и редактора, но они все отличались не слишком дружелюбным интерфейсом. В Linux стандартной и обязательной программой для редактирования конфигурационных файлов традиционно считается текстовый редактор vi, в последнее время чаще всего используется редактор Vim. Для вызова установленного в дистрибутиве текстового редактора используется одна и та же команда: vi.

Редактор vi имеет ряд функций, предназначенных для работы с файлами, и работа в нем сильно отличается от любых текстовых редакторов ОС Windows и MS DOS. Знание команд редактора vi может быть полезно при чтении справочных материалов в Linux (help, info, man). Основная мощь редактора vi заключается в комбинации команд передвижения курсора с командами редактора, которых более 100. Когда был написан редактор vi, многие терминалы на UNIX системах не имели клавиш-стрелок. Вместо них были выбраны клавиши h, j, k, l как команды перемещения влево, вниз, вверх, вправо. Ниже рассматриваются основные команды редактора vi.

1.4.1. Команды редактора vi

Основное отличие программы vi от обычного редактора текста в том, что после запуска программа vi может находиться в трех режимах: ввода текста, командном режиме, режиме командной строки. Для более продвинутого текстового редактора Vim введен еще один режим - визуальный выбор текста. Следует обратить внимание на то, что в редакторе vi традиционные комбинации клавиш типа Ctrl+Q, Ctrl+C и Ctrl+X, которые обычно прекращают работу программы, не дают эффекта - для выхода из программы vi надо сначала перейти в командный режим, введя символ :, а потом ввести определенную команду. В этом случае курсор переместится в нижнюю строку, в которой появится приглашение. В этом режиме вводятся команды:

:q - выход из редактора без записи файла;

:q! - выход из редактора без записи файла, когда текст в редакторе изменялся;

:x - выход из редактора с записью файла;

:w - запись файла и возвращение в командный режим;

:e name - чтение файла;

:r name - добавить содержимое указанного файла к редактируемому;

: [command - выполнить команду операционной системы;

:h - вызов справки.

Выше приведены сокращенные варианты команд, например, q - это quit, а w - write. Можно употреблять любое сокращение полного имени команды.

В тех случаях, когда в результате случайного нажатия клавиш программа vi перешла в непонятный режим, попробуйте дважды нажать клавишу Esc или комбинацию клавиш Ctrl+Q. Далее надо снова ввести символ : для перехода в командный режим.

1.4.2. Редактирование файла

Для создания и редактирования файлов в редакторе vi необходимо выполнить следующие действия:

- 1) Перейти в командный режим, нажав клавишу :
- 2) Создать новый файл, например file, с помощью команды *edit file* . Так как такого файла еще нет, то в нижней строке редактора появится сообщение *"file" [New File]*. Вначале файл file пуст и на экране нет букв или цифр.
- 3) Для начала ввода текста в файл существуют две команды:
 - i - текст добавляется перед текущим символом (курсор указывает на этот символ);
 - a - текст добавляется после текущего символа.

Для перемещения по тексту в режиме ввода текста нельзя использовать мышь и клавиши управления курсором, так как вместо сдвига курсора будут вводиться служебные символы. Для передвижения курсора служат специальные комбинации клавиш, о которых можно узнать с помощью команды *:help*.

Для более удобного перемещения указателя ввода следует воспользоваться клавишей *Esc* - перейти в режим просмотра текста. В этом режиме работают клавиши управления курсором, поэтому можно выбрать нужную позицию в тексте и нажатием клавиши с символами *i* или *a* снова перейти в режим ввода текста.

4) В режиме просмотра текста допускается удалять текущий символ с помощью команды *x*. Для удаления всей строки используется команда *dd*.

5) В режиме ввода текста работает клавиша *Backspace*, но только в текущей строке, причем стираемые символы продолжают оставаться на экране!

6) Для сохранения файла следует нажать на клавишу *Esc*, а потом ввести символ : для перехода в командный режим.

7) После появления внизу экрана приглашения командного режима необходимо ввести символ *w*, т.е. дать команду для записи файла. Если процесс записи файла закончится успехом, появится информация о записанном файле, а также статистика, например:

"file" [New File] 5 lines. 65 characters written

1.4.3. Установка опций

Существует множество опций, которые влияют на работу vi. Все доступные опции можно посмотреть с помощью команды *set all*. Также команду *set* можно использовать для установки различных опций. Например, для отображения порядковых номеров строк файла нужно использовать команду: *:set number*

Для отключения отображения номеров строк необходимо ввести команду:

:set nonumber

Некоторые опции могут иметь параметры, например, *:set tabstop=4* заставит символ табуляции отображаться как четыре пробела, вместо обычных восьми.

Для сохранения опций и при следующем сеансе работы необходимо поместить их в файл *.exrc* или установить переменную окружения *EXINIT* соответствующим образом. Например, если в качестве shell используется *Borne shell*, можно разместить в вашем *.profile* следующую строку:

EXINIT='set nomagic nu tabstop=4'; export EXINIT

Если используется *C shell*, необходимо разместить в вашем *.login* файле:

setenv EXINIT='set nomagic nu tabstop=4'

Контрольные вопросы

- 1) Перечислите основные отличия Linux от других ОС.
- 2) В чем заключается экологичность ОС Linux?
- 3) Приведите классификацию и синтаксис основных команд.
- 4) Какой инструмент используется для поиска файлов с определенными характеристиками?
- 5) Что является в Linux рабочим каталогом пользователя?
- 6) Приведите примеры обозначения рабочего каталога пользователя.
- 7) Поясните синтаксис задания диапазона при поиске файлов.
- 8) Поясните синтаксис команд создания файла.
- 9) Перечислите программы, доступные на консоли по умолчанию.
- 10) Назовите средство повторного обращения к уже выполненным командам.
- 11) Перечислите основные возможности редактора *vim*.
- 12) Как запустить редактор *vim*?
- 13) Поясните применение опций в редакторе *vi*.
- 14) Как выйти из редактора файлов с сохранением результатов модифицированного файла и без сохранения?

РАЗДЕЛ 2. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ КАК СРЕДСТВА ТЕКСТОВОГО ПОИСКА

2.1. Основные понятия и возможности

При работе в ОС UNIX или ОС Linux часто используются регулярные выражения (*regular expression, regexp*) - механизм, позволяющий задать шаблон для строки и осуществить поиск данных, соответствующих этому шаблону в заданном тексте. Сутью механизма регулярных выражений является то, что они позволяют задать шаблон для нечеткого поиска по тексту. Регулярные выражения и методы работы с ними используются в shell-программировании, что позволяет повысить качество создаваемых сценариев - во многих случаях три-четыре команды фильтрации текста можно заменить одной командой с регулярным выражением.

Регулярные выражения можно применять для фильтрации текстовых файлов или выходных данных команды либо сценария. Они представляют собой набор шаблонов, состоящих как из специальных, так и обычных символов, т.е. строка, которая всегда начинается с символа разделителя, за которым следует непосредственно регулярное выражение, затем еще один символ разделителя и затем необязательный список модификаторов. В качестве символа разделителя обычно используется слэш ('/'). Например, в регулярном выражении: `\d{4}-\d{2}/m`, символ '/' является разделителем, строка `\d{4}-\d{2}'` - непосредственно регулярным выражением, а символ 'm', расположенный после второго разделителя - это модификатор.

Регулярные выражения в той или иной форме используются всеми основными текстовыми редакторами и утилитами, выполняющими фильтрацию текста. К сожалению, наборы поддерживаемых выражений несколько различаются от программы к программе, но существуют так называемые базовые регулярные выражения, которые во всех программах обрабатываются одинаково. Единственное исключение — оператор `\{ \}`, который поддерживается в программах `sed` и `grep`, но не поддерживается в `awk`.

2.2. Создание шаблонов для поиска выражений, стоящих в начале или в конце строки

2.2.1. Метасимволы и операторы базовых регулярных выражений

Основой синтаксиса регулярных выражений является то, что некоторые символы, встречающиеся в строке, рассматриваются не как обычные символы, а как имеющие специальное значение, так называемые метасимволы.

Для создания шаблонов для поиска выражений, стоящих в начале или в конце строки, используются следующие *метасимволы и операторы базовых регулярных выражений*:

- ^ - соответствует началу строки;
- \$ - соответствует концу строки;
- [] - соответствует любому символу из числа заключенных в скобки, например, чтобы задать диапазон символов от 1 до 5, следует указать

первый символ диапазона, дефис и последний символ [1-5]) вместо шаблона [12345];

[^] - соответствует любому символу, кроме тех, что указаны в скобках;

\ - отменяет специальное значение следующего за ним метасимвола;

. - соответствует любому отдельному символу;

* - указывает на то, что предыдущий шаблон встречается ноль или более раз. В программах `awk` и `egrep`, где используются расширенные регулярные выражения, существует два дополнительных оператора:

? - предыдущий шаблон встречается не более одного раза;

+ - предыдущий шаблон встречается один или более раз;

\{n\} - предыдущий шаблон встречается ровно n раз ;

\{n,\} - предыдущий шаблон встречается не менее n раз;

\{,m\} - предыдущий шаблон встречается не более m раз;

\{n,m\} - предыдущий шаблон встречается не менее n раз и не более m раз.

2.2.2. Поиск одиночных символов с помощью метасимвола '.'

Метасимвол '.' соответствует любому одиночному символу. Если, например, требуется найти слово, начинающееся с подстроки "beg", после которой стоит произвольный символ, а за ним — символ 'n', задайте шаблон `beg.n`. Будут найдены такие слова, как "begin", "began" и т.д.

Данный метасимвол удобно применять при фильтрации результатов работы команды `ls -l` для поиска файлов, имеющих требуемые права доступа: например, шаблон, соответствующий файлам, право на исполнение которых есть у всех пользователей:

`...x..x..x`

Примеры отбора строк режима по этому шаблону приведены ниже:

`-rwxrwxr-x`

`-rwxr-xr-x`

Предположим, выполняется фильтрация текстового файла. Необходимо найти строки, состоящие из десяти символов, из которых пятый и шестой символы - "XC". Данная задача решается с помощью такого шаблона:

`....XC....`

Он означает, что первые четыре символа могут быть произвольными, следующие два - "XC", а последние четыре могут быть произвольными. Вот несколько примеров сравнения:

`1234XC9088` – соответствует заданному шаблону,

`4523XX9001` - не соответствует заданному шаблону,

`0011XA9912` - не соответствует заданному шаблону,

`9931XC3445` - соответствует заданному шаблону.

2.2.3. Поиск выражений в начале строки с помощью метасимвола '^'

Метасимвол '^' позволяет искать слова или символы, стоящие в начале строки. Например, благодаря шаблону `^d` можно отобрать из списка,

выводимого командой `ls -l`, только те записи, которые соответствуют каталогам:

`drwxrwxrw-` - соответствует заданному шаблону,
`-rw-rw-rw-` - не соответствует заданному шаблону,
`drwxrwxr-x` - соответствует заданному шаблону,
`-rwxr-xr-x` - не соответствует заданному шаблону.

Для поиска строк, у которых в четвертой позиции от начала стоит символ 'l', можно воспользоваться следующим шаблоном:

`^... l`

В результате получим:

`1234XC9088` - не соответствует;
`4523XX9001` - не соответствует;
`0011XA9912` - соответствует;
`9931XC3445` - соответствует.

Чтобы найти строки, начинающиеся с символов "comp", следует указать: `^comp`

В регулярном выражении можно сочетать различные шаблоны поиска. Предположим, после символов "comp" могут идти любые две буквы, но завершать должны символы "ing":

`^comp..ing`

Этот шаблон обеспечивает поиск таких слов, как "computing", "complaining" и т.д.

2.2.4. Поиск выражений в конце строки с помощью метасимвола '\$'

Метасимвол '\$' предназначен для поиска слов или символов, находящихся в конце строки. Он указывается в конце шаблона. Предположим, требуется найти строки, заканчивающиеся словом "trouble". Эту задачу позволяет решить шаблон:

`trouble$`

Следующий шаблон соответствует пустой строке, не содержащей символов: `^$`
 А с помощью показанного ниже шаблона можно найти строки, включающие только один символ: `^.s`

2.3. Создание шаблонов для поиска символов, встречающихся неопределенное число раз

Метасимвол '*' означает, что предыдущий символ в регулярном выражении либо отсутствует, либо встречается произвольное число раз подряд (1, 2 и т.д.). Например, шаблон `compi*t` отвечает таким словам:

`computer,`
`computing,`
`comпииииите,`

шаблон `10133*` соответствует следующему:

`101333`
`10133`
`10134`

2.4. Создание шаблонов для поиска специальных символов

Ряд символов, попадая в состав регулярных выражений, приобретает специальное значение. В общем случае специальными являются следующие символы:

`$ / ' " * [] ^ | () \ + ?`

Когда требуется найти строку, содержащую один из таких символов, его необходимо "защитить" в шаблоне с помощью обратной косой черты, которая отменяет специальное значение следующего за ней метасимвола. Предположим, строка содержит символ '.', который, как известно, в регулярном выражении соответствует произвольному символу, шаблон для него: `\.`

Если необходимо найти файлы, допустим, с расширением *xls*, можно применить следующий шаблон:

`*\.xls`

2.5. Создание шаблонов для поиска символов из указанного набора или диапазона

Шаблон `[]` соответствует списку или диапазону символов, указанных в квадратных скобках. Символы в списке можно разделять запятыми, что облегчает восприятие шаблона, хотя и не является обязательным требованием.

Для задания диапазона символов используется дефис (-). Слева от него указывается первый символ диапазона, а справа - последний. Предположим, необходимо найти символ, являющийся цифрой. Можно применить такой шаблон:

`[0123456789]`.

Однако проще задать диапазон: `[0-9]`.

Следующий шаблон соответствует любой строчной букве: `[a-z]`. Чтобы найти любую букву произвольного регистра, следует воспользоваться шаблоном `[A-Za-z]`. Здесь формируется список из двух диапазонов: прописные буквы от 'A' до 'Z' и строчные буквы от 'a' до 'z'.

Представленный ниже шаблон соответствует любому алфавитно-цифровому символу:

`[A-Za-z0-9]`.

Шаблон, предназначенный для поиска трехсимвольных комбинаций следующего типа: в начале находится буква 's', за ней может следовать любая прописная или строчная буква, а завершает последовательность буква 't', записывается следующим образом:

`s[a-zA-Z]t`

Если же комбинация состоит только из букв нижнего регистра, следует воспользоваться шаблоном:

`s[a-z]t`.

Чтобы найти слово "computer" независимо от того, расположено оно в начале предложения или нет, можно применить шаблон:

`[Cc]omputer`

Следующий шаблон соответствует слову "system", которое начинается с прописной или строчной буквы и за которым следует точка: $[S,s/system]$. Запятая в квадратных скобках поставлена для того, чтобы сделать шаблон удобным для зрительного восприятия.

Метасимвол '*', размещенный после квадратных скобок, указывает на то, что символы в скобках могут повторяться неопределенное число раз. Например, следующий шаблон соответствует любому слову:

$[A-Za-z]^*$

Метасимвол '^' после открывающей квадратной скобки - признак того, что шаблон соответствует любым символам, кроме указанных в скобках. Так, шаблон $[^a-zA-Z]$ соответствует всем символам, кроме букв, а шаблон $[^0-9]$ отвечает всем символам, которые не являются числами.

2.6. Поиск символов, встречающихся заданное число раз

Метасимвол '*' позволяет находить символы, встречающиеся несколько раз подряд, но число повторений при этом не определяется. Если же необходимо в процессе поиска учитывать точное количество последовательных вхождений символа в строку, следует применить шаблон $\{n\}$.

Существует четыре варианта этого шаблона:

- 1) $\{n\}$ - символы встречаются ровно n раз подряд;
- 2) $\{n,\}$ - символы встречаются не менее n раз подряд;
- 3) $\{,m\}$ - символы встречаются не более m раз подряд;
- 4) $\{n,m\}$ - символы встречаются не менее n и не более m раз подряд, где n и m — целые числа из интервала от 0 до 255.

Представленный ниже шаблон соответствует последовательности из двух букв 'A', за которыми следует буква 'B':

$A\{2\}B$

В результате получим "AAB".

В следующем шаблоне задано, что буква 'A' встречается не менее четырех раз подряд:

$A\{4,\}B$

Возможные результаты поиска - "AAAAB" или "AAAAAAAB", но не "AAAB". Поиск последовательности, в которой буква 'A' встречается от двух до четырех раз, выполняется по такому шаблону:

$A\{2,4\}B$

Будут найдены строки "AAB", "AAAB", "AAAAB", но не "AB" или "AAAAAB".

Допустим, в ранее рассмотренном примере фильтрации текстового файла требуется найти строки, в которых первые четыре символа цифры, за ними идут символы "XX", а затем еще четыре цифры. Решить данную задачу позволит шаблон:

$[0-9]\{4\}XX[0-9]\{4\}$

Применив этот шаблон к рассмотренному фрагменту, получим:

1234XC9088 - не соответствует;

4523XX9001 - соответствует;
 0011XA9912 - не соответствует;
 9931XC3445 - не соответствует.

2.7. Инструментальное средство **grep**

Команда **grep** (global regular expression print - печать глобальных регулярных выражений) является наиболее известным инструментальным средством в UNIX и Linux. Она выполняет в текстовых файлах или стандартном входном потоке поиск выражений, соответствующих шаблону, с последующим отображением результата на экране. Команда **grep** может работать как с базовыми, так и с расширенными регулярными выражениями. Существует три разновидности этой команды:

grep - стандартный вариант, которому уделено основное внимание в данной главе;

egrep - работает с расширенными регулярными выражениями (не поддерживает только оператор `\{ \}`);

fgrep - быстрый вариант команды **grep**. Вместо поиска выражений, соответствующих шаблону, выполняет поиск фиксированных строк из указанного списка. Несмотря на название "быстрый" это наиболее медленная из команд семейства **grep**.

К сожалению, нет единого способа задания аргументов для всех трех разновидностей команды **grep**, к тому же скорость их работы заметно различается.

2.7.1. Команда **grep**

Общий формат команды **grep** следующий:

grep [параметры] базовое_регулярное_выражение [файл]

В качестве регулярного выражения может быть указана обычная строка. Если файл не указан, текст берется из стандартного входного потока.

Строку, которая задана в качестве регулярного выражения и состоит из нескольких слов, следует заключать в двойные кавычки. В противном случае первое слово строки будет воспринято как образец поиска, а все остальные слова будут считаться именами файлов. В итоге отобразится сообщение о том, что указанные файлы не найдены.

Если образец поиска состоит из какой-нибудь системной переменной, например `$PATH`, рекомендуется тоже взять ее в двойные кавычки. Это связано с тем, что, прежде чем передавать аргументы команде **grep**, интерпретатор `shell` выполняет подстановку переменных, и команда **grep** получает значение переменной, которое может содержать пробелы. В этом случае будет выдано то же сообщение об ошибке, о котором говорилось в предыдущем абзаце.

Шаблон поиска должен быть заключен в одинарные кавычки, если в состав регулярного выражения входят метасимволы.

Команда **grep** имеет следующие параметры:

-с - задает отображение только числового значения, указывающего, сколько строк соответствуют шаблону;

- i - дает указание игнорировать регистр символов;
- h - подавляет вывод имен файлов, включающих найденные строки (по умолчанию в выводе команды `grep` каждой строке предшествует имя файла, в котором она содержится);
- l - задает отображение только имен файлов, содержащих найденные строки;
- n - задает нумерацию выводимых строк;
- s - подавляет вывод сообщений о несуществующих или нетекстовых файлах;
- v - задает отображение строк, не соответствующих шаблону.

Поиск среди нескольких файлов

Если в текущем каталоге требуется найти последовательность символов "sort" во всех файлах с расширением *doc*, следует ввести команду:

```
grep sort *.doc
```

Следующая команда осуществляет поиск фразы "sort it" во всех файлах текущего каталога:

```
grep "sort it" *
```

В рассматриваемых ниже примерах будет использоваться файл с именем *data.f*, содержащий информацию о заказах товаров. Структура записи этого файла следующая:

- 1-й столбец — код города;
- 2-й столбец — код месяца, когда был сделан заказ;
- 3-й столбец — код заказа, включающий год, когда он был сделан;
- 4-й столбец — код товара;
- 5-й столбец — цена за единицу товара;
- 6-й столбец — код фирмы;
- 7-й столбец — количество заказанного товара.

```
$ cat data.f
```

48	dec	3BC1997	LPSX	68. 00	LVX2A	138
483	sept	5AP1996	USP	65. 00	LVX2C	189
47	oct	3ZL1998	LPSX	43. 00	KVM9D	532
219	dec	2CC1999	CAD	23 00	PLV2C	68
484	nov	7PU996	CAD	49. 00	PLV2C	234
483	may	5PA1998	USP	37. 00	KVM9D	644
216	sept	3ZL1998	USP	86. 00	KVM9E	234

Разделителем полей является символ табуляции

Определение числа строк, в которых найдено совпадение

Опция `-c` позволяет узнать, сколько строк соответствуют заданному шаблону. Это может оказаться полезным в том случае, когда команда `grep` находит слишком много строк, и их лучше вывести в файл, а не на экран. Рассмотрим пример:

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

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

```
$ grep "48" data.f
48 dec 3BC1997 LPSX 68.00 LVX2A 138
483 sept 5AP1996 USP 65.00 LVX2C 189
484 nov 7PL1996 CAD 49.00 PLV2C 234
483 may 5PA1998 USP 37.00 KVM9D 644
```

Вывод номеров строк

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

```
grep -n "48" data.f
```

на экран будет выведена следующая информация:

```
1:48 dec 3BC1997 LPSX 68.00 LVX2A 138
2:483 sept. 5AP1996 USP 65.00 LVX2C 189
5:484 nov 7PL1996 CAD 49.00 PLV2C 234
6:483 may 5PA1998 USP 37.00 KVM9D 644
```

Номера строк отображаются в первом столбце.

Поиск строк, не соответствующих шаблону

Благодаря опции `-v` можно отобрать те строки, которые не соответствуют шаблону. Следующая команда извлекает из файла *data.f* строки, не содержащие последовательность символов "48":

```
grep -v "48" data.f
```

Результат выполнения команды:

```
47 oct 3ZL1998 LPSX 43.00 KVM9D 512
219 dec 2CC1999 CAD 23.00 PLV2C 68
216 sept 3ZL1998 USP 86.00 KVM9E 234
```

Поиск символов на границе слов

При поиске строк, содержащих последовательность символов "48", были найдены строки заказов с кодом города не только 48, но также 483 и 484. Если необходимо найти заказ, у которого код города равен 48, необходимо добавить в шаблон поиска символ табуляции:

```
grep "48<tab>" data.f
```

Здесь запись `<tab>` означает нажатие клавиши [Tab].

Тогда результат выполнения команды будет следующий:

```
48 Dec 3BC1997 LPSX 68.00 LVX2A 138
```

Если не известно, какой пробельный символ является разделителем полей в файле, можно воспользоваться регулярным выражением `\>` - *признаком конца слова*:

```
grep '48\>' data.f
```

Результат выполнения команды:

48 Dec 3BC1997 LPSX 68.00 LVX2A 138

Игнорирование регистра символов

По умолчанию команда `grep` чувствительна к изменению регистра символов. Для выполнения поиска без учета регистра используется опция `-i`. В файле `data.f` обозначение месяца Sept встречается как в верхнем, так и в нижнем регистре. Поэтому для отбора строк обоих видов следует применить такую команду:

```
$ grep -i "sept" data.f
```

Результатами поиска будут нижеприведенные строки:

483 Sept 5AP1996 USP 65.00 LVX2C 189

216 sept 3ZL1998 USP 86.00 KVM9E 234

Команда `grep` и регулярные выражения

С помощью регулярных выражений можно задавать более сложные критерии фильтрации информации. При работе с регулярными выражениями следует заключать шаблон поиска в одинарные кавычки - это позволит защитить все встречающиеся в нем специальные символы. В противном случае интерпретатор `shell` может "перехватывать" их у команды `grep`.

Выбор символов из списка

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

```
grep '48[34]' data.f
```

Выбранные строки:

483 Sept 5AP1996 JSP 65.00 LVX2C 189

484 nov 7PL1996 CAD 49.00 PLV2C 234

483 may 5PA1998 USP 37.00 KVM9D 644

Инверсия шаблона с помощью метасимвола '^'

Выполнить в файле `data.f` поиск строк, не начинающихся с заданных символов, например, цифры 4 или 8, можно, используя команду:

```
grep '^^[48]' data.f
```

Символ '^' заключенный в квадратные скобки, говорит о том, что шаблон соответствует любому символу, кроме цифр 4 и 8. Символ '^' в начале шаблона - признак того, что поиск производится с начала каждой строки.

Результатом будут следующие строки:

219 dec 2CC1999 CAD 23.00 PLV2C 68

216 sept 3ZL1998 USP 86.00 KVM9E 234

Шаблон, соответствующий любому символу

Предположим, в файле `data.f` требуется найти коды фирм, которые начинаются на букву 'K' и заканчиваются буквой 'D'. Реализуется это следующим образом:

```
grep 'K...D' data.f
```

Данный шаблон рассчитан на то, что коды фирм в файле состоят из *пяти* символов:

47 Oct 3ZL1998 LPSX 43.00 KVM9D 512
 483 may 5PA1998 ' USP 37.00 KVM9D 644

Ниже представлена небольшая вариация вышеприведенного примера. На этот раз осуществляется поиск всех кодов со следующей структурой: первые два символа - буквы в верхнем регистре, далее следуют два произвольных символа, а завершает последовательность буква 'C':

```
grep '[A-Z] [A-Z]..C' data.f
```

Выбранные строки:

483 Sept 5AP1996 USP 65.00 LVX2G 189
 219 dec 2CC1999 CAD 23.00 PLV2C 68
 484 nov 7PL1996 CAD 49.00 PLV2C 234

Поиск по дате

Представленная ниже команда находит все заказы, которые были сделаны в 2010 или 2011 году и коды которых начинаются с цифры 5:

```
grep '5..201[01]' data.f
```

Пример результата выполнения команды:

483 Sept 5AP2010 USP 65.00 LVX2C 189
 483 may 5PA2011 USP 37.00 KVM9D 644

Структура используемого здесь шаблона такова: первым символом является цифра 5, за ней следует два произвольных символа, затем число 201, а последним символом может быть либо цифра 0, либо цифра 1.

Поиск всех заказов, сделанных в 2013 году, выполняется посредством команды:

```
grep '[0-2]\{3\}3' data.f
```

и может иметь вид:

47 Oct 3ZL2013 LPSX 43.00 KVM9D 512
 483 may 5PA2013 USP 37.00 KVM9D 644
 216 sept 3ZL2013 USP 86.00 KVM9E 234

Примененный в этом примере шаблон означает: найти любую последовательность из трех цифр, за которой идет цифра 3.

Комбинированные диапазоны

Допустим, необходимо найти строки, в которых код города имеет следующий формат: первым символом является произвольная цифра, второй символ выбирается из диапазона от 0 до 5, а третий символ принадлежит диапазону от 0 до 6. Воспользуемся следующей командой:

```
grep '[0-9][0-5][0-6]' data.f
```

Результаты выполнения команды:

47 Oct 3ZL1998 LPSX 43.00 KVM9D 512
 484 nov 7PL1996 CAD 49.00 PLV2C 234
 483 may 5PA1998 USP 37.00 KVM9D 644
 216 sept 3ZL1998 USP 86.00 KVM9E 234

Как видно, отображается больше информации, чем необходимо. Можно сделать вывод, что в шаблоне поиска недостаточно уточнен критерий отбора информации. Очевидно, следует указать, что поиск нужно начинать в начале строки. Для этого применим метасимвол '^':

```
grep ^[0-9][0-5][0-6]' data.f
```

Результаты выполнения команды:

```
216 sept 3ZL1998 USP 86.00 KVM9E 234
```

Поиск повторяющихся последовательностей

Для поиска какой-либо строки, содержащей, например, цифру 4, повторенную минимум дважды, следует ввести команду:

```
grep '4{2,}' data.f
```

Запятая указывает, что предыдущий символ встречается не менее двух раз.

Пример результата выполнения команды:

```
483 may 5PA1998 USP 37,00 KVM9D 644
```

Для поиска всех записей, содержащих, по крайней мере, три девятки, можно воспользоваться следующей командой:

```
grep '9{3,}' data.f
```

Пример результата поиска:

```
219 dec 2CC1999 CAD 23.00 PLV2C 68
```

Иногда точное число повторений символа не известно. В таких случаях окажется полезной команда со следующей структурой:

```
grep '8{2,6}3' myfile
```

Здесь задан поиск строк, в которых цифра 8 встречается от двух до шести раз подряд и предшествует цифре 3. Ниже приведены варианты:

```
83 - не соответствует заданному поиску,
888883 - соответствует заданному поиску,
8884 - не соответствует заданному поиску,
88883 - соответствует заданному поиску
```

Выбор из нескольких шаблонов

Опция *-E* позволяет использовать в команде *grep* синтаксис расширенных регулярных выражений. Предположим, необходимо найти все заказы с кодами городов 216 или 219. В этом случае можно воспользоваться метасимволом '|' задающим выбор из двух шаблонов:

```
grep -E '219/216' data.f
```

Результатом могут быть следующие строки:

```
219 dec 2CC1999 CAD 23.00 PLV2C 68
216 sept 3ZL1998 USP 86.00 KVM9E 234
```

Поиск пустых строк

Для поиска в файле пустых строк можно составить шаблон из метасимволов '^' и '\$', например:

```
grep '^$' myfile
```

Поиск имен файлов, соответствующих заданному формату

Пусть в вашей системе применяется следующий формат наименования

файлов с документами: до шести символов, расположенных в начале, являются буквами нижнего регистра, далее следует точка, а завершают последовательность два символа верхнего регистра. Если требуется найти имена файлов подобного типа, записанные в файл *fflename*, следует применить команду:

```
grep '[a-z]\{1,6\}\. [A-Z]\{1,2\}' ffilename
```

Названия файлов: *yrend.AS*, *soa.PP*, *qr.RR* - соответствуют заданным условиям, а *mothdf* - не соответствует.

Поиск IP-адресов

Администратору DNS-сервера приходится поддерживать большое количество IP-адресов, относящихся к различным сетям, например, файл *ipfile* может содержать свыше 200 адресов. Для поиска всех адресов в формате "*nnn.nnn*" (т.е. адресов, содержащих две трехзначные последовательности, оканчивающиеся точкой), можно воспользоваться следующей командой:

```
grep '[0-9]\{3\}\.[0-9]\{3\}\.'
```

Классы символов

Команда *grep* поддерживает целый ряд предопределенных диапазонов символов, называемых *классами*. Обозначение класса состоит из открывающей квадратной скобки, двоеточия, собственно имени класса, двоеточия и закрывающей квадратной скобки. Поскольку класс представляет собой диапазон, обозначение класса дополнительно должно заключаться в квадратные скобки:

Класс	Эквивалентное регулярное выражение
<code>[[:upper:]]</code>	<code>[A-Z]</code>
<code>[[:lower:]]</code>	<code>[a-z]</code>
<code>[[:digit:]]</code>	<code>[0-9]</code>
<code>[[:alnum:]]</code>	<code>[0-9a-zA-Z]</code>
<code>[[:space:]]</code>	символы пробела
<code>[[:alpha:]]</code>	<code>[a-zA-Z]</code>

Отличие классов от эквивалентных регулярных выражений состоит в том, что класс включает не только символы английского алфавита из стандартной таблицы ASCII-кодов, но также символы того национального языка, который установлен в данной конкретной системе.

Рассмотрим несколько примеров на базе нашего файла *data.f*. Предположим, требуется найти все коды заказов, которые содержат цифру 5, сопровождаемую, по крайней мере, двумя буквами в верхнем регистре:

```
grep '5[[:upper:]][[:upper:]]' data.f
```

Результаты поиска:

```
483 Sept 5AP1996 USP 65.00 LVX2C 189
```

```
483 may 5PA1998 USP 37.00 KVM9D 644
```

Для поиска всех кодов товара, которые оканчиваются буквой 'P' или 'D', введем команду:

```
grep '[[[:upper:]][[:upper:]] [PD]]' data.f
```

Ниже приведены результаты выполнения команды:

483	Sept	5AP1996	USP	65.	.00	LVX2C	189
219	dec	2CC1999	CAD	23.	,00	PLV2C	68
484	nov	7PL1996	CAD	49.	.00	PLV2C	234
483	may	5PA1998	USP	37,	.00	KVM9D	644
216	sept	3ZL1998	USP	86.	.00	KVM9E	234

2.7.2. Команда egrep

Команда `egrep` (extended `grep`) воспринимает как базовые, так и расширенные регулярные выражения. Одной из привлекательных ее особенностей является возможность сохранения шаблонов поиска в файле. Подключается этот файл с помощью опции `-f`. Рассмотрим пример:

```
cat grepstrings
484
47
```

Введем команду:

```
egrep -f grepstrings data.f
```

В этом случае в файле `data.f` осуществляется поиск записей, которые содержат последовательность символов "484" или "47".

Создание файла шаблонов удобно в том случае, когда шаблонов много и вводить их в командной строке затруднительно. Если шаблонов немного, можно воспользоваться метасимволом `|` который позволяет сделать выбор между несколькими шаблонами. Например, следующая команда ищет в файле `data.f` записи, содержащие последовательность символов "3ZL" или "2CC":

```
egrep '3ZL|2CC' data.f
```

Результат поиска:

47	Oct	3ZL1998	LPSX	43.00	KVM9D	512
219	dec	2CC1999	CAD	23.00	PLV2C	68
216	sept	3ZL1998	USP	86.00	KVM9E	234

Круглые скобки позволяют представить выражение с несколькими шаблонами как один шаблон. Так, с помощью представленной ниже команды можно найти в текущем каталоге файлы, в которых встречаются слова из ряда "shutdown", "shutdowns", "reboot" и "reboots":

```
egrep '(shutdown/reboot) s? ' *
```

Контрольные вопросы

- 1) Поясните термин «регулярные выражения».
- 2) В чем заключается механизм регулярных выражений?
- 3) Можно ли в регулярном выражении сочетать различные шаблоны поиска?
- 4) Перечислите метасимволы и базовые операторы, используемые для создания шаблонов для поиска выражений.
- 5) Назначение и возможности инструментального средства `grep`.

РАЗДЕЛ 3. ПРОСТЕЙШИЕ СРЕДСТВА SHELL

3.1. Пользовательская среда UNIX

Командный язык shell (в переводе - раковина, скорлупа) фактически есть язык программирования, на котором пользователь осуществляет управление компьютером. Shell не является необходимым и единственным командным языком (хотя он стандартизован в рамках POSIX [POSIX 1003.2] – стандарта мобильных систем). Немалой популярностью пользуются языки cshell, kshell и др. Кроме того, каждый пользователь может создать свой командный язык, может одновременно работать на одном экземпляре операционной системы с разными командными языками.

Базовой пользовательской средой UNIX является окно алфавитно-цифрового терминала. Однако сегодня, в условиях конкуренции на рынке ОС для ПК, характер работы в UNIX существенно отличается от того, каким он был, скажем, пятнадцать лет назад. Графический многооконный интерфейс, системы меню, техника drag-and-drop, — все это, казалось бы, стирает различия в работе с ОС UNIX и, например, с ОС Windows.

Несмотря на это, интерфейс командной строки - это самый непосредственный способ выполнения множества небольших задач администрирования в среде UNIX. Программа, с которой работает пользователь, - командный интерпретатор shell. Поэтому рассмотрим базовый пример работы в UNIX - использование командной строки интерпретатора shell.

3.2. Командный интерпретатор shell

Интерпретатор (shell) представляет собой интерфейс командной строки. По своей функциональности он похож на интерпретатор COMMAND.COM системы MS DOS. Одной из задач интерпретатора является обеспечение безопасного и структурированного доступа к ядру UNIX-подобных операционных систем, т.е. фактически это программный уровень, который предоставляет среду для ввода команд, обеспечивая тем самым взаимодействие между пользователем и ядром операционной системы. Кроме того, встроенный в интерпретатор мощный язык программирования используется для решения различных задач: от автоматизации повторяющихся команд до написания сложных интерактивных программ обработки данных, получения информации из небольших баз данных.

В среде UNIX существует много командных интерпретаторов, среди которых можно выделить такие как sh, tcsh, ksh, csh, bash, в Linux по умолчанию используется bash.

В табл. 3.1. приведены встроенные команды интерпретатора shell.

Таблица 3.1. Встроенные команды интерпретатора shell

<i>команда</i>	<i>Значение</i>
:	нуль, всегда возвращает истинное значение
.	считывание файлов из текущего интерпретатора shell
<i>break</i>	применяется в конструкциях for, while, until, case

<i>cd</i>	изменяет текущий каталог
<i>continue</i>	продолжает цикл, начиная следующую итерацию
<i>echo</i>	записывает вывод в стандартный поток вывода
<i>eval</i>	считывает аргумент и выполняет результирующую команду
<i>exit</i>	выход из интерпретатора shell
<i>export</i>	экспортирует переменные, они становятся доступны для текущего интерпретатора shell
<i>pwd</i>	отображает текущий каталог
<i>read</i>	просматривает строку текста из стандартного потока
<i>readonly</i>	превращает данную переменную в переменную "только для чтения"
<i>return</i>	выход из функции с отображением кода возврата
<i>set</i>	управляет отображением различных параметров для стандартного потока вводных данных
<i>shift</i>	смещает влево командную строку аргументов
<i>Test</i>	оценивает условное выражение
<i>times</i>	отображает имя пользователя и системные промежутки времени для процессов, которые выполняются с помощью интерпретатора shell
<i>Trap</i>	при получении сигнала выполняет определенную команду
<i>Type</i>	интерпретирует, каким образом интерпретатор shell применяет имя в качестве команды
<i>Ulimit</i>	отображает или устанавливает ресурсы интерпретатора shell
<i>umask</i>	отображает или устанавливает режимы создания файлов, заданные по умолчанию
<i>Unset</i>	удаляет из памяти интерпретатора shell переменную или функцию
<i>Wait</i>	ожидает окончания дочернего процесса и сообщает о его завершении

Сам интерпретатор shell автоматически присваивает значения следующим переменным (параметрам):

- ? - значение, возвращенное последней командой;
- \$ - номер процесса;
- ! - номер фонового процесса;
- # - число позиционных параметров, передаваемых в shell;
- * - перечень параметров, как одна строка;
- @ - перечень параметров, как совокупность слов;
- флаги, передаваемые в shell.

При обращении к этим переменным, т.е. при использовании в командном файле, следует впереди ставить "\$".

Важную роль при создании уникальных файлов играет специальная переменная "\$\$", значение которой соответствует номеру процесса, выполняющего данный расчет. Каждый новый расчет, выполняемый компьютером, инициирует один или несколько процессов, автоматически получающих номера по порядку. Поэтому, используя номер процесса в качестве имени файла, можно быть уверенным, что каждый новый файл будет иметь новое имя (не запишется на место уже существующего). Достоинство является и главным недостатком такого способа именования файлов. Неизвестно, какие имена будут присвоены файлам. И, если в рамках данного процесса можно найти файл "не глядя", т.е., обратившись к нему, используя \$\$, то потом такие файлы можно легко потерять. Это создает дополнительные проблемы при отладке программ.

3.3. Приемы экранирования

Рассмотрим приемы экранирования, используемые в shell. В качестве средств экранирования используются двойные кавычки (" "), одинарные кавычки (' ') и бэк-слэш (\).

Экранирующий символ (\), используемый совместно с командами *echo* и *sed*, сообщает интерпретатору, что следующий за ним символ должен восприниматься как специальный символ:

- \n - перевод строки (новая строка);
- \r - перевод каретки;
- \t - табуляция;
- \v - вертикальная табуляция;
- \b - забой (backspace);
- \a - "звонок" (сигнал);
- \0xx - ASCII-символ с кодом 0xx в восьмеричном виде.

В командном файле бэк-слэш экранирует конец строки, что позволяет объединять строки в одну, т.е. запись:

```
cat file | grep -h \
result | sort | cat -b > filerez
```

аналогична записи:

```
cat file | grep -h result | sort | cat -b > filerez
```

Ниже приведены примеры использования экранирующих символов (\):

- вывод последовательности символов:

```
echo "\s\s\s\"
```

- вывод вертикальной табуляции:

```
echo -e "\s\s\s"
```

- вывод символа " (кавычки с восьмеричным кодом ASCII 42):

```
echo -e "\042"
```

Конструкция `$\X'` делает использование ключа `-e` необязательным;

- запись ASCII-символов в переменную, например, символа “ :

```
quote=$\042'
```


3.4. Организация циклов и ветвлений

3.4.1. Условный оператор "if"

В общем случае оператор "if" имеет структуру:

```
if условие
    then список
    [elif условие
        then список]
    else список]
fi
```

Служебное слово "elif" - сокращенный вариант от "else if", может быть использован наряду с полной записью, т.е. допускается вложение произвольного числа операторов "if" (как и других операторов). Если выполнено условие, то выполняется "список", иначе он пропускается. Структура обязательно завершается служебным словом "fi". Число "fi" всегда должно соответствовать числу "if".

3.4.2. Команда test

Команда test проверяет выполнение некоторого условия. С использованием этой (встроенной) команды формируются операторы выбора и цикла языка shell. Имеется два возможных формата команды:

```
test условие
```

или

```
[ условие ]
```

Интерпретатор shell будет распознавать эту команду по открывающей скобке "[". Между скобками и содержащимся в них условием обязательно должны быть пробелы. Пробелы должны быть и между значениями и символом сравнения или операции. В shell используются условия различных "типов":

1) условия сравнения целых чисел:

```
x eq y    - "x" равно "y",
x ne y    - "x" не равно "y",
x gt y    - "x" больше "y",
x ge y    - "x" больше или равно "y",
x lt y    - "x" меньше "y",
x le y    - "x" меньше или равно "y".
```

2) сложные условия, реализуемые с помощью типовых логических операций:

```
!      (not) инвертирует значение кода завершения;
-o     (or) соответствует логическому "ИЛИ";
-a     (and) соответствует логическому "И".
```

3) условия проверки файлов:

```
-f file   - файл "file" является обычным файлом;
-d file   - файл "file" - каталог;
-c file   - файл "file" - специальный файл;
-r file   - имеется разрешение на чтение файла "file";
```

- w file** - имеется разрешение на запись в файл "file";
- s file** - файл "file" не пустой.

4) условия проверки строк:

- str1 = str2** - строки "str1" и "str2" совпадают;
- str1 != str2** - строки "str1" и "str2" не совпадают;
- n str1** - строка "str1" существует (непуста);
- z str1** - строка "str1" не существует (пуста).

3.4.3. Оператор цикла с перечислением – **for**

Оператор цикла "for" имеет структуру:

```
for имя in список-значений
do
    список команд
done
```

где *for* - служебное слово, определяющее тип цикла, *do* и *done* - служебные слова, выделяющие тело цикла. Фрагмент *in список-значений* может отсутствовать.

Рассмотрим пример:

```
for i in f1 f2 f3
do
    proc-sort $i
done
```

В этом примере имя *i* играет роль параметра цикла. Это имя можно рассматривать как shell-переменную, которой последовательно присваиваются перечисленные значения (*i=f1*, *i=f2*, *i=f3*), и выполняется в цикле команда *procsort*.

Часто используется форма *for i in **, означающая для всех файлов текущего каталога.

3.4.4. Оператор цикла с истинным условием - **while**

Структура оператора *while* предпочтительнее тогда, когда неизвестен заранее точный список значений параметров или этот список должен быть получен в результате вычислений в цикле. Оператор цикла *while* имеет структуру:

```
while
    условие
do
    список команд
done ,
```

где *while* - служебное слово, определяющее тип цикла с истинным условием. Список команд в теле цикла (между *do* и *done*) повторяется до тех пор, пока сохраняется истинность условия (т.е. код завершения последней команды в теле цикла равен 0) или цикл не будет прерван изнутри специальными командами (*break*, *continue* или *exit*). При первом входе в цикл условие должно выполняться.

3.4.5. Оператор цикла с ложным условием *until*

Отличие оператора цикла *until* от оператора *while* состоит в том, что условие цикла проверяется на ложность (на ненулевой код завершения последней команды тела цикла) после каждого (в том числе и первого) выполнения команд тела цикла. Программистов, знакомых с операторами *until* в других языках, может вначале сбивать такая семантика этого оператора.

Оператор цикла *until* имеет структуру:

```
until условие
  do
    список команд
  done
```

где *until* - служебное слово, определяющее тип цикла с ложным условием. Список команд в теле цикла (между *do* и *done*) повторяется до тех пор, пока сохраняется ложность условия или цикл не будет прерван изнутри специальными командами (*break*, *continue* или *exit*). При первом входе в цикл условие не должно выполняться.

Кроме рассмотренных операторов ветвления и циклов существует ряд других операторов, которые можно найти в приведенной в списке литературе.

3.5. Функции интерпретатора *shell*

Интерпретатор команд *shell* позволяет группировать наборы команд или конструкций, создавая повторно используемые блоки. Подобные блоки называются *shell*-функциями.

Функция состоит из двух частей:

Метка функции *Тело функции*

В качестве метки выступает имя функции, тело функции образует набор команд, составляющих саму функцию. Имя функции должно быть уникальным; это связано с тем, что в случае наличия двух различных функций с одинаковыми именами сценарий просто не сможет вызвать нужную функцию.

Формат, применяемый для определения функций, имеет следующий вид:

```
имя функции ( )
{
  команда1
  ...
}
```

Можно также использовать ключевое слово *function* перед именем функции:

```
function имя_функции ( )
{
}
```

Функцию можно представлять себе как некоторый вид сценария, находящегося внутри другого сценария, но в этом случае следует учитывать одну особенность. При вызове функции она остается в текущем интерпретаторе *shell*, а ее копия хранится в памяти. С другой стороны, если вызывается или выполняется сценарий из другого сценария, создается отдельный

интерпретатор shell. В таком случае становятся недействительными все существующие переменные, определенные в предыдущем сценарии.

Функции могут быть размещены в том же самом файле, что и сценарии, либо в отдельном файле, содержащем функции. При этом функции не всегда включают множество конструкций или команд - может просто содержаться единственная конструкция echo, которая выполняется при вызове функции.

Применение функций позволяет сэкономить массу времени, затрачиваемого на разработку сценариев. Благодаря созданию универсальных и многократно используемых функций отпадает необходимость в технической поддержке разработанных сценариев, использующих эти функции.

3.5.1. Объявление функций в сценарии

Перед использованием функций их необходимо объявить. Суть объявления заключается в том, что все функции должны быть размещены в начале кода сценария. Невозможно сослаться на функцию до тех пор, пока она не попадет в "поле зрения" интерпретатора команд. Для вызова функции требуется просто ввести ее имя, например:

```
hello ()
(
  echo "Hello there today's date is Mate*"
)
```

В этом примере функция называлась "hello", тело функции включало конструкцию echo, которая, в свою очередь, отображала текущую дату.

3.5.2. Использование функций в сценарии

Рассмотрим методы применения функции в сценарии.

```
$ pg fund
#!/bin/sh
# funcl
hello ()
{
  echo "Hello there today's date is 'date'
}
echo "now going to the function hello" hello
echo "back from the function"
```

При выполнении этого сценария получаются следующие результаты:

```
$ fund
now going to the function hello
Hello there today's date is Sun Jun 6 10:46:59 GMT 2000
back from the function
```

В предыдущем примере функция была объявлена в начале сценария. Для обращения к функции просто вводится ее имя, в данном случае "hello". После завершения выполнения функции управление возвращается следующей конструкции, которая размещена после вызова функции. В приведенном примере речь идет о конструкции echo "back from the function".

3.5.3. Передача параметров функции

Порядок передачи параметров функции аналогичен передаче параметров обычному сценарию. При этом используются специальные переменные \$1, \$2, ... \$9. При получении функцией переданных ей аргументов происходит замена аргументов, изначально переданных сценарию интерпретатора shell. В связи с этим желательно было бы повторно присвоить значения переменным, получаемым функцией. В любом случае это стоит сделать, так как при наличии ошибок в функциях их можно будет легко обнаружить, воспользовавшись именами локальных переменных. Для вызывающих аргументов (переменных), находящихся внутри функции, имя каждой переменной начинается с символа подчеркивания, например: `_FILENAME` или `_filename`.

3.5.4. Возврат значения функции

После естественного завершения выполнения функции либо в том случае, когда она завершается в результате выполнения какого-либо условия, можно выбрать один из двух возможных вариантов:

1) завершение функции и последующая передача управления той части сценария, которая вызвала данную функцию;

2) использование ключевого слова `return`, в результате чего будет осуществлена передача управления конструкции, которая расположена за оператором вызова функции. При этом может также указываться необязательный числовой параметр. Этот параметр принимает значение 0 в случае отсутствия ошибок и значение 1 - при наличии ошибок. Действие этого параметра аналогично действию кода завершения последней команды.

При использовании ключевого слова `return` применяется следующий формат:

return - возвращает результат из функции, использует код завершения последней команды для проверки состояния;

return 0 - применяется при отсутствии ошибок;

return 1 - применяется при наличии ошибок.

Для проверки значения, возвращаемого вызванной функцией, можно воспользоваться кодом завершения последней команды, размещенной непосредственно после функции, которая вызывается из сценария. Например:

```
check_it is_a_directory $FILENAME      # вызов функции и проверка
if [ $? = 0 ]                           # применение кода завершения
                                         # последней команды для тестирования
```

```
then
```

```
echo "All is OK"
```

```
else
```

```
echo "Something went wrong! "
```

```
fi
```

Лучшим методом является использование оператора `if`, с помощью которого осуществляется проверка возвращаемого значения (0 или 1).

Встраивание вызова функции в структуру оператора `if` значительно улучшает читабельность программного кода. Например:

```
if check_it_is_a_directory $FILENAME;
then
echo "All is OK"
# действия
else
echo "Something went wrong!"
# действия
fi
```

Если функцию планируется использовать для отображения результатов некоторой проверки, для фиксации результата применяется подстановка. Формат, используемый для выполнения подстановки при вызове функции, следующий:

```
имя_переменной = 'имя_функции'
```

Выводимый результат функции *имя функции* присваивается переменной *имя_переменной*.

3.5.5. Файл функций

Файл функций содержит несколько регулярно используемых функций. В начале этого файла должна находиться конструкция `#!/bin/sh`. Файлу функций можно присвоить любое имя, затем загрузить этот файл в среду интерпретатора `shell`. Как только файл будет загружен интерпретатором `shell`, появляется возможность вызова функций из командной строки либо из сценария. Для просмотра всех определенных функций можно воспользоваться командой `set`. Поток вывода будет содержать все функции, которые были загружены интерпретатором `shell`.

Для изменения любой из ранее определенных функций используется команда `unset`, после чего функция будет выгружена из интерпретатора `shell`. После внесения изменений в файл функций необходимо выполнить его перезагрузку. В результате этого данную функцию будет невозможно вызвать из пользовательских сценариев или интерпретатора команд, хотя физического удаления функции при этом не происходит. Некоторые интерпретаторы команд могут распознавать изменения, и в этом случае применение команды `unset` вовсе не обязательно. Однако все же в целях безопасности следует использовать данную команду при модификации функций интерпретатора `shell`.

3.5.6. Создание файла функций

Рассмотрим создание файла функций, включающего одну функцию. Эта функция будет загружена интерпретатором команд, протестирована, изменена, а затем повторно загружена.

Создаваемый файл функций, например `functions.main`, будет содержать следующий код:

```
$ pg functions.main
#!/bin/sh
```

```

# functions.main
# findit: интерфейс для базовой команды find
findit () (
# findit
if [ $# -lt 1 ];
then
echo "usage :findit file"
return 1
fi
find / -name $1 -print

```

В данном примере создается файл функций *functions.main*, включающий одну функцию. Эта функция будет загружена интерпретатором команд, протестирована, изменена, а затем повторно загружена. Данный код лежит в основе интерфейса для базовой команды *find*. Если команде не передаются аргументы, то возвращается значение 1 (что свидетельствует о возникновении ошибки). Обратите внимание, что ошибочная конструкция фактически является отображенным именем функции (если же была использована команда \$0, интерпретатор команд просто возвращает сообщение *sh*). Причина отображения подобного сообщения заключается в том, что файл не является файлом сценария.

3.5.7. Подключение файла функций и проверка загруженных функций

Команда подключения файла функций имеет следующий формат:

```
./путь/имя_файла
```

Например, *\$. functions.main*. Если в результате выполнения этой команды возвращается сообщение «File not found» (файл не найден), можно воспользоваться следующей командой:

```
$. /functions.main
```

В этом случае применяется нотация

```
<точка><пробел><косая черта><имя файла>.
```

Для проверки загруженности интерпретатором команд функции используется команда *set*, которая отображает все загруженные функции, доступные для сценария:

```

$ set
USER=stud
findit=()
[
if ( $# -lt 1 ]; then
echo "usage :findit file"; return 1;
find / -name $1 -print
}
...

```

3.5.8. Вызов функций интерпретатора shell

Для вызова функции необходимо ввести ее имя (в данном случае *findit*) и указать аргумент, в роли которого может выступать имя файла, размещенного в системе:

```
$ findit groups
/usr/bin/groups /usr/local/backups/groups.bak
```

3.5.9. Удаление shell-функций

Для выполнения операции удаления shell-функции применяется команда *unset*. При вызове данной команды используется следующий формат:

```
unset имя_функции
```

Например:

```
$ unset findit
```

Если ввести команду *set*, функция *findit* не будет найдена.

3.5.10. Редактирование shell-функций

Если добавить в функцию *functions.main* цикл, используя оператор *for*, то сценарий сможет считывать более одного параметра из командной строки. Функция приобретет следующий вид:

```
$ pg functions.main
#!/bin/sh
findit()
{
# findit
#if [ $# -lt 1 ]
then
echo "usage :findit file"
return 1
fi
for loop
do
find / -name $loop -print
done
}
```

Если интерпретатор shell корректно интерпретирует цикл *for* и воспринимает все требуемые параметры, на экране отобразится соответствующее сообщение:

```
$ set findit=()
{
if [ $# -lt 1 ] then
echo "usage :'basename $0' file";
return 1;
fi;
for loop in "$@";
do
```



```
find / -name $loop -print;
done
}
```

...

Далее вызывается измененная функция `findit`. При этом поддерживаются два файла с целью осуществления поиска:

```
$ findit SPO.doc passwd
/usr/local/accounts/SPO.doc
/etc/passwd
```

...

3.6. Применение shell-сценариев

Применение shell-сценариев позволяет экономно расходовать рабочее время при выполнении важных и сложных заданий. Shell-сценарии производят сортировку файлов, вставку текста в файлы, перемещение файлов, удаление строк из файлов, удаление старых файлов из системы, а также выполняют в системе некоторые административные задания.

В сценариях используются переменные, условные, арифметические и циклические конструкции, которые можно отнести к системным командам. За счет этих возможностей сценарии создаются довольно быстро. Интерпретатор команд может применять в качестве вводных данных для одной команды информацию, полученную при выполнении другой команды. Чтобы shell-сценарий применялся в различных системах UNIX и Linux, в него нужно внести лишь небольшие изменения. Это связано с тем, что интерпретатор shell обладает высокой степенью универсальности. Определенные трудности возникают лишь вследствие ориентации системных команд на определенные системы.

Сценарий не относится к компилируемым программам, поскольку он интерпретируется построчно. Первой строкой сценария всегда должна быть строка вида:

```
#!/bin/sh
```

Таким образом, система получает указание, где следует искать интерпретатор Bourne shell. Каждый сценарий может содержать комментарии; если комментарии размещаются в сценарии, первым символом в строке комментария будет символ `#`. Встретив подобный символ, интерпретатор команд игнорирует строку комментария. Сценарий просматривается интерпретатором команд в направлении сверху вниз. Перед выполнением сценария требуется воспользоваться командой `chmod`, устанавливающей право доступа на выполнение. Для выполнения сценария достаточно указать имя файла сценария.

Приведенный ниже сценарий отменяет отображение сообщений `var/adm/` путем усечения файла сообщений. В задачи этого сценария также входит удаление всех журнальных файлов в каталоге `/usr/local/apps/log`:

```
$ pg cleanup
```

```
#!/bin/sh
# имя: cleanup
# это общий сценарий/ выполняющий очистку echo "starting
cleanup...wait"
rm /usr/local/apps/log/*.log
tail -40 /var/adm/messages /tmp/messages
rm /var/adm/messages
mv /tmp/messages /var/adm/messages
echo "finished cleanup"
```

Для выполнения сценария применим команду `chmod`:

```
$ chmod u+x cleanup
```

Для запуска сценария на выполнение необходимо ввести его название:

```
$ cleanup
```

При отображении сообщения об ошибке, например:

```
$ cleanup
sh: cleanup: command not found
```

воспользуйтесь командой:

```
$ ./cleanup
```

Если перед выполнением сценария нужно указать путь доступа к нему или же сценарий сообщает, что не может обнаружить команду, достаточно в значение переменной `PATH` из файла `.profile` добавить каталог `bin`. При вводе следующей информации сначала убедитесь, что вы находитесь в каталоге `$HOME/stud`:

```
$ pwd
$ /home/stud/bin
```

Если последняя часть команды `pwd` включает название подкаталога `/bin`, его следует использовать при указании имени пути. Изменить файл `.profile`, добавив в него каталог `$HOME/bin`, можно, используя следующую команду:

```
PATH = $PATH:$HOME/bin
```

В случае, если подкаталог `/bin` отсутствует, необходимо создать его:

```
$ mkdir bin
```

Затем в файле `.profile` добавляется каталог `bin` в переменную `PATH`, и заново инициализируется файл `.profile`:

```
$ ./profile
```

Контрольные вопросы

- 1) Что такое shell-процедура?
- 2) Какого типа команды могут быть включены в тело процедуры?
- 3) Чем отличается обработка процедуры при выполнении от обработки программы на языке высокого уровня?
- 4) Поясните общую структуру скрипта.

- 5) В каком виде хранятся переменные в программах командного интерпретатора?
- 6) Что такое параметры? Для каких целей они используются?
- 7) Какое число параметров может быть передано процедуре?
- 8) Перечислите переменные (параметры), которым интерпретатор shell автоматически присваивает значения.
- 9) Приведите примеры сложных синтаксических конструкций получения значений переменной.
- 10) Перечислите внутренние переменные shell, используемые в скриптах.
- 11) Как осуществляется ветвление вычислительного процесса процедуры?
- 12) Какого типа циклы в процедурах могут быть построены средствами языка shell?
- 13) Поясните процедуру создания файла функций.
- 14) Как подключить файл функций?
- 15) Поясните процедуру выполнения проверки загруженных функций.
- 16) В каких случаях целесообразно использование shell-сценариев?

РАЗДЕЛ 4. УСТАНОВКА И ОБНОВЛЕНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ В ОС LINUX

4.1. Основные способы установки программного обеспечения

4.1.1. Дистрибутивы Linux

ОС Linux распространяется посредством дистрибутивов. *Дистрибутив* представляет собой набор пакетов программного обеспечения, в который также входит ядро ОС и необходимый для работы набор утилит.


С момента выхода первой версии ядра в мире уже существует несколько десятков различных дистрибутивов Linux. Некоторые из них разрабатываются компаниями на коммерческой основе, а другие распространяются на условиях лицензии GPL, т.е. бесплатно. Можно выделить два отличия дистрибутивов:


- программа установки и конфигурирования операционной системы (текстовые инсталляторы, графические оболочки и т.д.);
- количество программного обеспечения и утилит, которое поставляется с дистрибутивом.





Приложения, базовые средства и утилиты включены в дистрибутивы в виде так называемых откомпилированных программных групп, которые называются пакетами. Вот именно в формате этих пакетов и кроется еще одно различие между дистрибутивами.




Наибольшее распространение получили три вида пакетов: rpm (формат Red Hat Linux), deb (формат Debian) и tgz (формат Slackware). По этому признаку часто выделяют дистрибутивы, которые основаны на Red Hat Linux или Debian. В табл. 4.1 приведены основные дистрибутивы Linux.

Таблица 4.1. Основные дистрибутивы Linux

Название	Описание
 http://www.debian.org/	<p>ОС Debian - дистрибутив с большим набором свободного ПО, который наследует лучшие традиции ОС UNIX. Данный дистрибутив - полностью некоммерческий проект, в разработку и развитие которого вносят свой вклад множество добровольных разработчиков со всего мира. В процессе разработки дистрибутива параллельно существует 3 ветки - стабильная, в процессе тестирования и нестабильная. При появлении новой версии пакета Debian она помещается в нестабильную категорию. После прохождения начального процесса тестирования она переходит в категорию основного тестирования, в котором может находиться несколько месяцев. И только после тестирования множеством тестеров новая версия официально признается стабильной. В результате такого процесса разработки данный</p>

	<p>дистрибутив всегда получается очень надежным, стабильным и подходящим для использования на серверах. В качестве недостатков можно указать неудобную установку, которая требует от пользователя знаний ОС и затратность по времени, в то же время имеется удобный инсталлятор пакетов apt-get, благодаря которому обновления системы проходят более просто, чем установка системы.</p>
 redhat http://www.redhat.com	<p>Для многих пользователей Red Hat - это и есть Линукс. Одной из причин популярности Red Hat является большое разнообразие сервисов, которые предлагает компания. Программы просто обновляются через механизм Red Hat Net Network. В дистрибутив всегда входит самое актуальное ядро, библиотеки, графические оболочки KDE и GNOME, а еще и большое количество разнообразных программ. RedHat может успешно работать в качестве сервера для WWW/FTP, Proху-сервера, рабочего места администратора сети, разработчика приложений, мультимедийной платформы. В полный комплект поставки дистрибутива входит документация и исходные коды самых популярных программ. В Red Hat используется свой формат пакетов RPM - Red Hat Packet Manager. Все это позволяет использовать RedHat в качестве ОС на самых ответственных серверах Интернета. Существует сертификационная программа для пользователей этой системы RHCE (Red Hat Certified Engineer), процесс обучения и сертификации по которой сейчас есть во многих странах мира.</p>
 ASPLINUX http://www.asplinux.ru	<p>Дистрибутив ASP Linux разработан одноименной российской компанией. При этом большинство программ русифицировано, как и процесс установки системы. ASP Linux отличается простотой использования и функциональностью. Ориентирован на рядового пользователя, в его состав входит все необходимое ПО для повседневной работы - офисные пакеты, веб-инструменты, игры, мультимедиа и т.д.</p>

 http://www.altlinux.ru	<p>Выпускается командой разработчиков ALT Linux (ALT Linux Team), в которую входят преимущественно разработчики из Восточной Европы: России, Украины, Белоруссии. Сама же фирма координирует разработку этого дистрибутива, осуществляет техническую поддержку.</p>
 http://www.slackware.com/	<p>Один из самых старых дистрибутивов и до сих пор очень популярен среди опытных пользователей Linux (содержит текстовый инсталлятор и текстовые утилиты для конфигурирования системы). Имеет репутацию защищенной и стабильной ОС, так как многие пакеты находятся в своей первоначальной форме - в исходных текстах.</p>
 http://www.suse.com	<p>Дистрибутив получил наибольшее распространение в странах Германии и Восточной Европы из-за собственного инсталлятора с набором утилит конфигурирования Yast. Также с дистрибутивом всегда идет очень подробная документация. SuSE Linux 7.3 даже заработал звание "Продукт года" от Linux Journal. Работа над выходом новых версий продукта происходит внутри компании SuSE, к процессу разработки посторонние не допускаются. Рекомендуются для пользователей, которые еще не работали с unix-подобными системами, а также для тех, кто привык к интерфейсу ОС Windows.</p>
 http://www.mandriva.com	<p>Дистрибутив, основанный Red Hat Linux, разрабатывается командой из Франции. Имеет большое количество полезных программ, содержит очень простую программу установки DrakX с графическим интерфейсом и поддержкой русского, белорусского и украинского языков. Популярен благодаря простоте установки для начинающего пользователя, возможности автоматического распознавания оборудования и утилиты для управления разделами жесткого диска признаны одними из лучших. Mandriva - единственный дистрибутив, который предлагает обе технологии AIGLX и Xgl сразу.</p>

 <p>gentoo linux</p> <p>http://www.gentoo.org</p>	<p>Дистрибутив Gentoo Linux отличается возможностью автоматической оптимизации и приспособления для применения в любой области информационных технологий. Высокая производительность, широкие возможности настройки и многочисленное сообщество пользователей и разработчиков - вот лишь главные черты ОС Gentoo Linux. В данном дистрибутиве используется технология портежей, что делает его идеальным защищенным сервером, рабочей станцией для программиста, настольной офисной системой, игровым приложением, мультимедиа.</p>
 <p>http://www.knoppix.ru</p>	<p>Дистрибутив Knoppix выгодно выделяется своей простотой: он может работать с компакт-диска, даже не требуя инсталляции на жесткий диск. В Knoppix используется так называемая технология «динамическая компрессия», что позволяет разместить на диске около тысячи программных пакетов (это две тысячи программ и два гигабайта информации), несколько оконных менеджеров (KDE среди них), офисные пакеты, графические редакторы, браузеры, проигрыватели аудио- и видеофайлов.</p>
 <p>http://www.geebox.org</p>	<p>Самозагрузочный диск с дистрибутивом Линукса, предназначенный в основном для проигрывания видео (DivX, XviD, FFmpeg, MPEG 1/2, VCD, DVD, OggMedia, Windows Media, RealMedia, и т.д.) и аудио (MP3, Audio CD, Ogg/Vorbis, и т.д.). Основой всего этого служит MPlayer, которым можно еще и ТВ смотреть.</p>

4.1.2. Способы установки программных пакетов в ОС Linux

Уже прошло довольно много времени с момента выхода первой версии ядра ОС Линукс (1991), но можно смело утверждать, что развитие этой ОС продолжается и в настоящее время. Выходят как совершенно новые дистрибутивы, так и новые версии уже старых и известных пакетов.

Необходимость в установке новых программных пакетов под Linux возникает в двух основных случаях:

- когда появляется новая версия одного из уже установленных на компьютере пакетов;
- когда возникает желание или необходимость использовать пакет еще не установленный.

Во втором случае это может быть один из пакетов, имеющихся на вашем установочном диске, но не установленный в процессе инсталляции. Однако

чаще всего программное обеспечение (ПО) находится в Интернет, так как значительная часть его бесплатна.

Для дистрибутивов, основанных на Red Hat Linux, существует две основные формы распространения ПО: в исходных текстах и в виде исполняемых модулей. В первом случае пакет ПО обычно поставляется в виде tar-gz архива, во втором случае - в виде rpm-пакета (но это не обязательно, исполняемые модули также могут распространяться в виде tar-gz-архива).

Для инсталляции новых пакетов необходимо войти в систему как пользователь root. Следует заметить, что установка ПО, представленного в виде rpm-пакета, содержащего исполняемые файлы, более проста с точки зрения пользователя.

4.2. Tar-gz-архивы и rpm-пакеты: понятие, создание и распаковка

4.2.1. Программа rpm

Название этой программы (или команды) является аббревиатурой от Redhat Package Manager. Такая расшифровка дается в большинстве книг и руководств по Linux и кажется более правильной и логичной, хотя в главе 6 "The Official Red Hat Linux Reference Guide" говорится: "The RPM Package Manager (RPM), is an open packaging system available for any-one to use, and works on Red Hat Linux as well as other Linux and UNIX systems", т.е. предлагается рекурсивная расшифровка названия RPM, подобная расшифровке GNU -GNU is Not Unix.

Программа *rpm* в некотором смысле аналогична программам типа *setup wizard* для MS Windows. Преимуществом использования этой программы по сравнению с установкой tar-gz архивов является то, что она автоматически выполняет все необходимые действия по установке программного обеспечения: создает необходимые каталоги, распределяет по ним файлы, создает ссылки. Кроме того, она может быть использована не только для установки нового пакета, но и для обновления версий ПО, получения перечней установленного ПО и проверки установки, а также для деинсталляции отдельных пакетов (например, если после периода пробной работы с программой принято решение отказаться от ее дальнейшего использования). С помощью той же программы rpm можно подготовленному пользователю создать пакет формата rpm.

4.2.2. Rpm-пакеты

Rpm-пакеты - это специальным образом подготовленные архивы, предназначенные для обработки программой rpm. Название rpm-пакетов оканчивается на суффикс *.rpm*, например, *xzip-180-l.i386.rpm* или *xzip-180-l.src.rpm*.

Как видно, перед суффиксом *.rpm* стоит еще один суффикс. Если это *.i386*, *.i686* или *.i586*, то в пакете находятся исполняемые файлы, оптимизированные для соответствующего типа процессора.

Если этот суффикс *.src*, то в пакете находятся исходные тексты, которые после установки еще надо скомпилировать. Обычно как на

установочных компакт-дисках, так и в интернет-каталогах rpm-пакеты с исполняемыми файлами располагаются в каталогах с названием RPMS, а rpm-пакеты с исходными текстами - в подкаталогах SRPMS.

Часто встречаются также rpm-пакеты с суффиксом *.noarch.rpm*, содержащие файлы, которые без дополнительной обработки устанавливаются в соответствующие каталоги, например, файлы страниц интерактивного руководства man.

И, наконец, если rpm-пакет рассчитан на версию Linux, предназначенную для другой аппаратной платформы (AMD, DEC Alpha, SUN Sparc, MIPS; PowerPC), это тоже будет отображено в имени пакета: вместо *.i386* в суффиксе будет стоять, соответственно, *athlon*, *alpha*, *sparc*, *mips* или *ppc*.

В Интернет rpm-пакеты можно найти на различных серверах, например, <http://rufus.w3.org> (другое имя <http://rpmfmd.net>). На нем установлена поисковая система, которая позволяет упорядочивать список пакетов:

- по именам пакетов;
- по дистрибутивам;
- по группам приложений;
- по датам;
- по поставщикам (производителям) программного обеспечения.

Из ftp-серверов в России следует выделить два: *ftp://frp.chg.ru/pub/Linux* и *ftp://ftp.nc.ore.ru/*.

Необходимо заметить, что если для перекачки пакетов из Интернет используется компьютер, работающий под ОС Windows, то возможна замена имен пакетов. Дело в том, что Windows "не любит" имена, в которых несколько точек (например, *glib-1.0.6-3.i386.rpm*) и заменит "лишние" точки на знаки подчеркивания - *glib-1_0_6-3_i386.rpm*. Поэтому после получения пакета (при переносе его на ПК с ОС Linux) желательно эти "исправления" устранить, вернувшись к исходным именам UNIX. Правда, делать это не обязательно, поскольку внутри rpm-пакет все равно правильно идентифицирован, но для единообразия и облегчения поиска файлов все же целесообразно.

Установка нового пакета

Для установки совершенно нового пакета из скаченного rpm-архива (т.е. на компьютере не было предыдущих версий этого ПО) достаточно перейти в тот каталог, где находится архив, и выполнить команду:

```
[root]# rpm -i имя_rpm_архива
```

Установка обновленного пакета

Если была установлена предыдущая версия пакета, то в простейшем случае надо дать команду следующего формата:

```
[root]# rpm -U —force имя__rpm_архива
```

Здесь параметр *-U* указывает на то, что необходимо произвести обновление (upgrade) пакета, а опция *-force* требует безусловного (без дополнительных вопросов) обновления всех входящих в пакет файлов. В некоторых случаях, может быть, следует сохранить какие-то (например, конфигурационные) файлы

от предыдущей версии. Если установка проходит нормально и никаких дополнительных сообщений не появляется, то после завершения работы программы (после появления приглашения оболочки) можно пользоваться вновь установленным пакетом.

Программа `rpm` позволяет выяснить, какие файлы установит тот или иной пакет. Для этого надо выполнить следующую команду:

```
[root]# rpm -qpl имя_ rpm _архива
```

При выполнении этой команды текущим должен быть каталог, содержащий интересующий пользователя пакет.

Информацию о назначении ПО, содержащегося в `rpm`-пакете, можно получить, используя команду:

```
[root]# rpm -qpi имя_ rpm _архива
```

Дело в том, что файлы RPM кроме собственно архива файлов содержат информацию о пакете, включая имя, версию и краткое описание. С помощью той же программы `rpm` можно просмотреть эту дополнительную информацию. Например, для пакета `glib-1.0.6-3.i386.rpm` результат исполнения команды:

```
[root]# rpm -qpi glib-1.0.6-3.i386.rpm
```

будет примерно таким:

```
Name : glib Relocations: (not relocateable)
```

```
Version : 1.0.6 Vendor: Red Hat Software
```

```
Release : 3 Build Date: Суб 10 Окм 1998 04:49:03
```

```
Install date: (not installed)
```

```
Build Host: porky.redhat.com
```

```
Group : Libraries Source RPM: glib-1.0.6-3.i386.rpm
```

```
Size: 55305
```

```
Packager : Red Hat Software <bug@redhat.com>
```

```
Summary : Handy library of utility functions
```

```
Description : Handy library of utility functions. Development libs and headers  
are in gtk+-devel.
```

Для выдачи списка входящих в пакет файлов с указанием того, куда они будут установлены, необходимо ввести команду:

```
[root]# rpm -qpl glib-1.0.6-3.i386.rpm
```

Пример результата выполнения команды:

```
/usr/lib/libglib.so.1
```

```
/usr/lib/libglib.so.1.0.6
```

4.2.3. RPM как система запросов

RPM также предоставляет мощную систему запросов по установленным в системе пакетам. Используя команду:

```
[root]# rpm -qa
```

можно получить перечень всех установленных в системе пакетов. Так как перечень большой по объему, то необходимо направить вывод в фильтр `more` или в файл, который потом просматривать с помощью `less` или встроенной программы просмотра из оболочки Midnight Commander.

Можно выполнить поиск информации об отдельном пакете или об отдельных файлах. Например, какому пакету принадлежит файл `/etc/bashrc` и откуда появился:

```
[root]# rpm -qf /etc/bashrc
```

В результате выполнения предыдущей команды система выдаст похожее сообщение:

```
bash-1.14.7-16
```

Для выполнения проверки, не удален ли важный файл из установленного пакета, используется команда:

```
[root]# rpm -Va
```

Пользователь будет оповещен о любых аномалиях. Можно переустановить пакет, если это необходимо, любые конфигурационные файлы будут сохранены.

Из вышеприведенных примеров видно, что `rpm` - полезная утилита, у нее имеется много разных опций. Всего `rpm` имеет 16 основных режимов работы, которые можно объединить в 6 групп (после двоеточия приводится формат команды для соответствующего режима):

1) Запросы:

Запрос: `rpm [--query] [queryoptions]`

Показать метки запросов (Querytags): `rpm [--querytags]`

2) Установка и поддержка установленных пакетов:

Установка: `rpm [--install] [installoptions] [package_file]+`

Обновление: `rpm [--freshen|-F] [installoptions] [package_file]+`

Деинсталляция: `rpm [--uninstall|-e] [uninstaloptions] [package]+`

Проверка: `rpm [--verify|-V] [verifyoptions] [package]+`

3) Подписи - пакеты подписываются электронной цифровой подписью в формате PGP, с целью обеспечения неизменяемости и сохранения авторства пакетов:

Проверка подписи: `rpm [--verify|-V] [verifyoptions] [package]+`

Переподписывание: `rpm [--resign] [package_file]+`

Добавление подписи: `rpm [--addsign] [package_file]+`

4) Работа с базой

Инициализация базы: `rpm -i [--initdb]`

Обновление базы (Rebuild Database): `rpm -i [--rebuilddb]`

5) Создание rpm -пакетов

Создать пакет: `rpm [--b|t] [package_spec]+`

Перекомпилировать пакет: `rpm [--rebuild] [souce rpm]+`

Скомпилировать пакет из tar-архива: `rpm [--tarbuild] [tarredsource]+`

6) Разное

Показать конфигурацию программы `rpm`: `rpm [--showrc]`

Задать пользователей: `rpm [--setperms] [package]+`

Задать группы: `rpm [--setgids] [package]+`

Подробное описание всех возможностей команды `rpm` приведено в [5].

Как и другие программы для Linux, программа `grm` постоянно развивается и совершенствуется. При этом при замене версии этой программы могут возникнуть трудности с установкой пакетов, созданных в предыдущих версиях. Так было, например, при переходе с третьей на четвертую версию `grm`. Так что надо использовать пакеты, соответствующие установленной на данном компьютере версии.

Приведенное выше описание программы `grm` предполагает, что она запускается с консоли или в эмуляторе терминала.

Между тем в разных дистрибутивах имеются графические оболочки для управления `grm`-пакетами. В составе графической среды KDE такая оболочка называется `kpackage`. Ее можно запустить либо из командной строки, либо из основного меню KDE.

4.3. Компиляция ПО из исходных текстов

Если `grm`-пакеты с необходимым программным обеспечением нужно еще поискать (и не всегда можно найти), то `tar-gz`-архив любого программного обеспечения для Linux найдется в Интернете всегда.

В некоторых случаях такие архивы содержат исполняемые модули приложений. Тогда установка приложения лишь немного сложнее, чем в случае установки из `grm`-пакета: необходимо просто раскрыть архив с помощью программ `gunzip` и `tar`, перейти в созданный каталог и можно запускать полученное приложение.

Но чаще всего приложения поставляются в исходных текстах, т.е. в виде программы на языке Си. Установить их в этом случае немного сложнее.

Напомним, что операционная система UNIX родилась на свет одновременно с языком программирования C. Более того, язык C был создан специально для разработки этой ОС, значительная часть UNIX была написана на языке C. ОС Linux тоже написана на Си. Поэтому, а также в соответствии с принципом свободного распространения исходных кодов, многие приложения для Linux распространяются в виде текстов на C (а в последнее время - и на C++).

Естественно, что для установки и запуска такого приложения на исполнение его необходимо предварительно скомпилировать. Для выполнения процедур компиляции обычно используется программа `gcc` (хотя существуют и некоторые альтернативные разработки).

Изначально аббревиатура *GCC* имела смысл *GNU C Compiler*, но в апреле 1999 года сообщество GNU решило взять на себя более сложную миссию и начать создание компиляторов для новых языков с новыми методами оптимизации, поддержкой новых платформ, улучшенных runtime-библиотек и других изменений (<http://gcc.gnu.org/gccmission.html>).

Поэтому сегодня GCC расшифровывается как GNUCompiler Collection (коллекция компиляторов GNU) и содержит в себе компиляторы для языков C, C++, Objective C, Chill, Fortran, Ada и Java, а также библиотеки для этих языков (`libstdc++`, `libgcj`, ...).

4.4. Основные компоненты GNU- компилятора

GNU-компилятор с языка C *gcc* содержит в себе четыре основные компонента, соответствующие четырем этапам преобразования исходного кода в исполняемую программу.

Первый компонент - это препроцессор, который модифицирует исходный код программы перед компиляцией в соответствии с командами препроцессора, содержащимися в C-программе. В соответствии с этими командами выполняются простые подстановки текста.

Второй - собственно компилятор, который обрабатывает исходный код и преобразует его в код на языке ассемблера.

Третий компонент - ассемблер, который генерирует объектный код.

Четвертый компонент - компоновщик, который собирает исполняемый файл из файлов объектного кода. Как известно, большие программы обычно пишутся по частям, в виде множества отдельных файлов, содержащих исходный код соответствующей части. Компилятор обрабатывает каждый такой файл отдельно и создает отдельные объектные модули (файлы таких модулей обычно имеют расширение *.o*). Создание единой исполняемой программы из таких модулей и является задачей компоновщика. При таком подходе, если в какой-то модуль программист вносит исправление, нет необходимости заново компилировать всю программу: достаточно откомпилировать исправленный модуль и заново запустить компоновщик.

Для выполнения стандартных операций программист может использовать функции из стандартных библиотек. Самый характерный пример - это библиотека *libc*, которая содержит функции, выполняющие такие задачи, как управление памятью и операции ввода-вывода. Программисты могут создать свои собственные библиотеки и использовать их при написании новых программ.

Библиотеки бывают статическими, разделяемыми и динамическими. *Статическая библиотека* - это библиотека, код которой встраивается в программу при компиляции. Код *разделяемой библиотеки* не встраивается в программу, а загружается в память одновременно с программой и программа получает доступ к функциям этой библиотеки. *Динамические библиотеки* - разновидность разделяемых, но библиотечные функции загружаются в память только в том случае, если из программы поступит вызов соответствующей функции. В процессе выполнения программы они могут выгружаться и заменяться другими функциями из той же или другой библиотеки.

Имена статических библиотек обычно имеют суффикс *.a*, а имена разделяемых библиотек - суффикс *.so*, за которым следует старший и младший номера версии. Имя может быть любой строкой, которая однозначно характеризует библиотеку. Обычно имена библиотек начинаются с *lib*. Примеры:

libm.so.5 - общая математическая библиотека,

libXll.so.6 - библиотека для работы с системой X Window.

Библиотека `libc.so.5` компонуется автоматически, в то время как большинство других библиотек необходимо явно указывать в командной строке при вызове программы `gcc`. Это делается через опцию `-l`, за которой следует уникальная часть имени библиотеки, например, для вызова математической библиотеки достаточно указать `-lm`.

Многие системные библиотеки располагаются в системных каталогах, например, в `/usr/lib` и `/lib`, но некоторые могут располагаться и в других местах.

Список этих каталогов помещается в файл `/etc/ld.so.conf`. Каждый раз, когда разделяемая библиотека изменяется или устанавливается вновь, нужно выполнять команду `ldconfig`, чтобы обновить файл `/etc/ld.so.conf`, а также ссылки на него. Если библиотека устанавливается из PRM-пакета, это обычно делается автоматически, хотя и не всегда.

При компиляции больших программ, использующих фрагменты исходного кода, расположенные в разных файлах, бывает сложно отследить, какие файлы нужно перекомпилировать, а какие только компоновать. В таких случаях помогает утилита `make`, которая автоматически определяет, следует ли компилировать файл исходного кода, по дате его последней модификации.

Утилита `make` оперирует файлами, исходя из их зависимости друг от друга. Эти зависимости определяются файлом с именем `makefile`. Строка файла `makefile` состоит из трех частей:

- имени целевого файла,
- списка файлов, от которых он зависит,
- команды.

Если какой-либо файл из списка изменился после целевого файла, то выполняется указанная в строке команда.

В строке может быть указано несколько команд. Обычно команда - это вызов компилятора для компиляции файла исходного кода или компоновки файлов объектного кода.

Строки, определяющие зависимости, отделяются друг от друга пустой строкой.

4.5. Установка пакетов ПО из исходных текстов

Рассмотрим обращение с пакетами программ, распространяемыми в виде исходных кодов. Для установки таких пакетов необходимо иметь в своей системе утилиты `gcc` и `make`.

Непосредственно процесс установки пакета состоит из следующих шагов:

- 1) Перейти (с помощью команды `cd`) в каталог, содержащий исходные коды устанавливаемого пакета;
- 2) Выполнить команду `./configure`, которая осуществляет конфигурирование пакета в соответствии с системой пользователя. Процесс выполнения этой команды занимает довольно длительное время, причем команда выдает на экран сообщения о том, какие именно особенности системы тестируются;

3) Выполнить команду *make*, для того, чтобы скомпилировать пакет.

4) Выполнить (этот шаг не является обязательным) команду *make check*, которая вызывает запуск процедур самотестирования, которые поставляются с пакетом.

5) Выполнить команду *make install* для установки программ, а также файлов данных и документации.

6) Заключительный этап состоит в выполнении команды *make clean*, которая удаляет промежуточные объектные и двоичные файлы из каталога с исходными кодами. Для удаления временных файлов, которые создала команда *configure* (после чего пакет можно компилировать для другого типа компьютеров), надо выполнить команду *make distclean*.

В большинстве случаев выполнение этой последовательности команд достаточно для установки нового пакета.

Основная проблема, с которой приходится сталкиваться при инсталляции программ из исходных кодов, связана с конфликтами версий: для вновь устанавливаемого пакета требуются новые версии каких-то системных утилит, которые пока еще не установлены в пользовательской системе. Более того, часто возникает целая цепочка (или даже дерево): для программы нужна какая-то новая версия утилиты, для последней нужно обновить еще какие-то утилиты и т.д.

Контрольные вопросы

1) Перечислите основные способы установки программного обеспечения ОС Linux.

2) Что понимается под дистрибутивом?

3) Перечислите наиболее популярные, назовите их основные отличия.

4) Что означает GNU?

5) Как создать пользовательский RPM-пакет?

6) Как распаковать TAR.GZ архив?

7) Как обновить пакет?

8) Перечислите основные этапы процесса инсталляции пакетов.

9) Какие основные проблемы возникают при инсталляции программ из исходных кодов?

10) Какие команды используются при установке пакета из архива?

11) Какой инструмент используется для получения информации о пакете, включая имя, версию и краткое описание?

12) Перечислите основные режимы работы rpm-утилиты.

13) Назовите отличия установки rpm-пакетов и tar-gz архивов.

14) Перечислите основные компоненты GNU- компилятора.

15) В чем состоит основная задача утилиты make при компиляции больших программ, использующих код, расположенный в разных файлах?

РАЗДЕЛ 5. CRON-ДЕМОН

5.1. Понятие демона в Linux -понимании

Демоном в Unix называют программу, которая работает "на заднем фоне", не требуя управления с терминала и предоставляя пользователю возможность выполнять "на переднем фоне" другую работу. Демон может быть запущен либо другим процессом, например, одним из системных стартовых скриптов без обращения к какому-либо управляющему терминалу, либо пользователем с какого-нибудь терминала.

Демоны, ссылки на которых присутствуют /etc/init.d, призваны работать в Linux как сервисы или службы. Сервисы - это программы, которые запускаются и останавливаются через инициализационные скрипты, расположенные в каталоге /etc/init.d. Многие из этих сервисов запускаются на этапе загрузки системы.

Утилита /sbin/service обеспечивает интерфейс (взаимодействие) пользователя с инициализационными скриптами, которые в свою очередь обеспечивают интерфейс для управления сервисами, предоставляя пользователю опции для запуска, остановки, перезапуска, запроса состояния сервиса и т.п.

Просмотреть текущее состояние всех системных служб можно с помощью следующей опции утилиты *service*:

```
/sbin/service -status-all
```

Результат выполнения утилиты представлен ниже:

```
acpid (pid 2481) is running...
anacron (pid 2647) is running...
atd (pid 2657) is running...
auditd (pid 2189) is running...
```

Утилита /usr/bin/system-config-services предоставляет графический интерфейс к системным службам, позволяющий просматривать и модифицировать их текущее состояние, а также задавать их запуск на различных уровнях исполнения. Вид графического интерфейса, предоставляемого утилитой /usr/bin/system-config-services, представлен на рис. 5.1.

Сервисы и демоны отображаются в выводе команды *ps*:

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	23:36	?	00:00:00	init [5]
root	2161	1	0	23:37	?	00:00:00	auditd
root	2177	1	0	23:37	?	00:00:00	syslogd -m 0
root	2287	1	0	23:37	?	00:00:00	rpc.idmapd
root	2577	1	0	23:37	?	00:00:00	cron
root	2631	1	0	23:37	?	00:00:00	/usr/sbin/atd

Как видно, для каждого из демонов идентификатор родительского процесса (PPID) равен 1. Это показывает, что демоны были запущены процессом *init* во время загрузки системы.

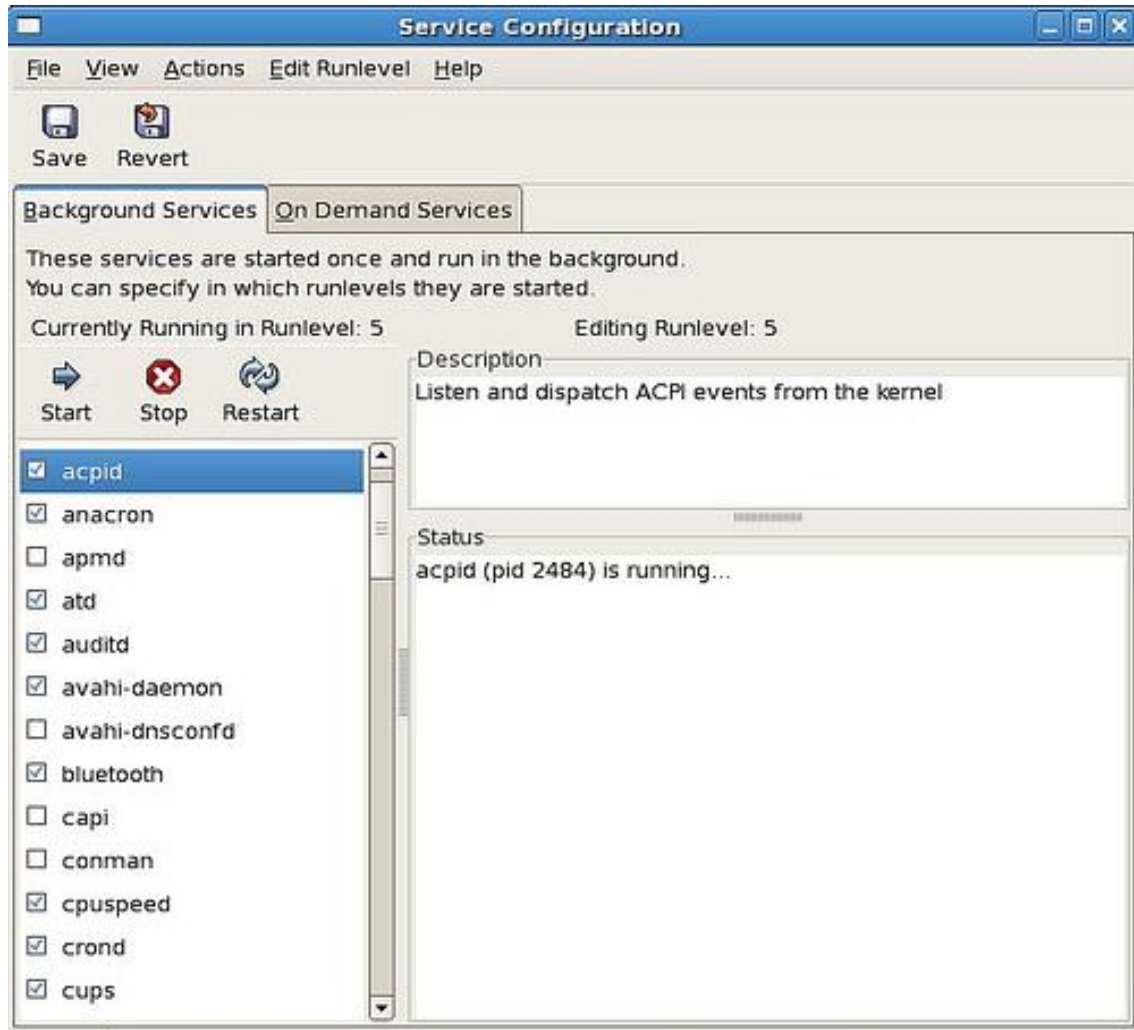


Рис. 5.1. Графический интерфейс, предоставляемый утилитой `/usr/bin/system-config-services`

5.2. Понятие cron

5.2.1. Определение cron

Cron - это демон, который используется для запуска задач по расписанию, в зависимости от времени, дня месяца, дня недели и т.д.

Периодически в системе необходимо обновлять разнообразные файлы, например, базы данных антивирусов, базу данных *whatis* или список всех доступных на чтение файлов системы, *locatedb*. Кроме того, необходимо собирать статистику по работе системы, анализировать целостность системы (этим занимаются службы *Osec*, *TripWire* или *AIDE*) и производить множество других регулярных действий. Всем этим и занимается демон *cron*. В *Red Hat Linux* также включена задача *cron*, удаляющая все неиспользуемые модули каждые десять минут. Эта задача *cron* описана в файле `/etc/cron.d/kmod`.

Чтобы использовать службу `cron`, необходимо установить RPM-пакет `vixie-cron`. Для проверки, установлен ли этот пакет, используется команда:

```
rpm -q vixie-cron
```

Чтобы запустить службу `cron`, следует выполнить следующую команду: `/sbin/service crond start`. Чтобы остановить её - команду: `/sbin/service crond stop`. Рекомендуется, чтобы эта служба запускалась при загрузке системы.

5.2.2. Конфигурационный файл `/etc/crontab`

При загрузке системы запускается демон `cron` и проверяет очередь `at` и заданий пользователей в файлах `crontab`.

Конфигурационный файл демона `cron` называется `/etc/crontab` и содержит следующие строки:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

Первые четыре строки - это переменные, настраивающие среду окружения, в котором будут работать задачи `cron`:

- значение переменной `SHELL` сообщает системе о том, какую оболочку использовать (в этом примере будет использована оболочка `bash`), если переменная не указана, значение берется из файла `/etc/passwd` для пользователя, являющегося владельцем файла,

- переменная `PATH` определяет пути, используемые для выполнения команд,

- переменная `MAILTO` определяет адрес электронной почты пользователя, по которому будет выслан результат выполнения задач `cron`. Если в качестве значения переменной `MAILTO` задана пустая строка (`MAILTO=""`), электронные письма отправляться не будут,

- переменная `HOME` задаёт домашний каталог, используемый при выполнении команд или сценариев.

Каждая строка в файле `/etc/crontab` имеет следующий формат:

```
minute hour day month dayofweek command ,
```

где *minute* - любое целое число от 0 до 59,

hour - любое целое от 0 до 23,

day - любое целое от 1 до 31 (день должен быть корректным, если указан месяц),

month - любое целое от 1 до 12 (или короткое название месяца, например: `jan`, `feb` и так далее),

dayofweek - любое целое от 0 до 7, где 0 или 7 означает воскресенье (или короткое название дня недели, например: sun, mon и так далее),

command - команда, которая должна быть выполнена. Командой может быть как простая команда, так и команда запуска написанного пользователем специального сценария.

Для любых указанных выше параметров можно использовать звездочку (*), что означает все допустимые значения. Например, если поставить звёздочку в значении месяца, команда будет выполняться каждый месяц во время, указанное другими параметрами.

Дефис (-) между целыми числами обозначает диапазон чисел.

Список значений, разделенных запятыми, обозначает перечень перечислений, например, перечисление 3, 5, 8, 9 означает четыре указанных целых числа.

Косая черта (/) используется для определения шага значений. Целочисленное значение может быть пропущено в диапазоне, если после диапазона указать /<целое>. Например, значение минут 0-59/2, определяет, что будет пропущена каждая вторая минута.

В качестве шага значений также может быть указана звёздочка. Например, значение месяца */2 определяет, что будет пропущен каждый второй месяц.

Любые строки, начинающиеся с символа решетки (#), являются комментариями и не обрабатываются.

Если ввести в командной строке пользователя *root* команду вывода файла на экран:

```
cat /etc/crontab
```

получим следующую информацию:

```
#minute (0-59),
#| hour (0-23),
#| | day of the month (1-31),
#| | | month of the year (1-12),
#| | | | day of the week (0-6 with 0=Sunday).
#| | | | | user
#| | | | | | commands
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

Первые пять полей этого файла определяют время запуска команды: минуту, час, число месяца, месяц и день недели. Символ "*" означает, что соответствующая часть даты не учитывается. Шестое поле - имя пользователя, от лица которого запускается команда, указанная в остальных полях строки. Так, в примере команда:

```
run-parts /etc/cron.weekly
```

будет запускаться в 4 часа 22 минуты каждое воскресенье (нулевой день) любого числа любого месяца. Как видно из примера, обычно */etc/crontab* невелик: чаще всего он состоит из почасового, подневного, понедельного и помесечного запуска специального сценария (в примере - *run-parts*). Этот сценарий реализует упрощенную схему ".d", он попросту запускает отсортированные в лексикографическом порядке сценарии из соответствующего каталога (например, из */etc/cron.daily*):

```
ls /etc/cron.daily
```

Результат: *000anacron logrotate makewhatis osec stmpclean updatedb*

Если задачи *cron* должны выполняться по расписанию, но не ежечасно, их можно добавить в каталог */etc/cron.d*. Все файлы в этом каталоге имеют тот же синтаксис, что и */etc/crontab*.

Демон *cron* каждую минуту ищет изменения в файле */etc/crontab* и каталогах *etc/cron.d/* и */var/spool/cron*. Если какие-либо изменения будут найдены, они загружаются в память. Таким образом, демон не нуждается в перезапуске при изменении файла *crontab*.

5.3. Создание файла *crontab* для пользователя

Пользователям системы можно разрешить иметь собственные расписания, также обрабатываемые демоном *cron*. Все созданные пользователями файлы *crontab* хранятся в каталоге */var/spool/cron* и выполняются от имени создавшего их пользователя.

Чтобы создать файл *crontab* для пользователя, войдите в систему под его именем и введите команду *crontab -e*, чтобы отредактировать *crontab* пользователя, с помощью редактора, указанного в значении переменной окружения *VISUAL* или *EDITOR*. Этот файл использует тот же формат, что и */etc/crontab*, только шестое поле ("user") отсутствует. После сохранения изменений в файле *crontab* он будет записан в соответствии с именем пользователя под названием */var/spool/cron/username*.

Рассмотрим пример создания файла *crontab* для пользователя *user*. Задача: запускать каждую минуту файл */home/user/mail*, который будет отправлять почту:

```
#содержимое файла mail (должны быть права запуска, например,755)  
#!/bin/bash  
mess="test cron"  
echo "$mess"/mutt -s "'subj'" -m application/octet-stream mst@server.ru
```

Создаем временный файл */home/user/test*, содержимое которого приводится ниже:

```
SHELL=/bin/bash  
MAILTO=user  
0-59 * * * * /home/ user/mail
```

Далее запустить в терминале команду:

```
crontab /home/user/test
```

После этого в каталоге `/var/spool/cron` будет создан файл `user` с примерным содержанием:

```
#DO NOT EDIT THIS FILE –edit the master and reinstall/
#(/home/user/test installed on Mon Apr 07 09:20:00 2011)
#(Cron version --$Id:crontab.  )
SHELL=/bin/bash
MAILTO=user
0-59 * * * * /home/ user/mail
```

Файл `/home/user/test` будет запускаться демоном `cron` каждую минуту.

Доступ в каталог `/var/spool/cron` непривилегированному пользователю закрыт, чтобы посмотреть пользователем `user`, есть ли у него файл `crontab`, достаточно набрать команду (для RedHat):

```
crontab -l
```

Если файл существует, будет показано содержимое.

Для удаления файла используется ключ `crontab -r`. Для редактирования файла используется команда: `crontab -e`.

5.4. Журнал событий cron-демона

Посмотреть информацию о всех командах, запускаемых демоном `cron`, можно в каталоге `/var/log` в соответствующих файлах `cron`, `cron1` и т.д.

В файле `/var/log/cron` записано время запуска всех заданий `cron` за предыдущий день:

```
Apr 07 09:20:00 rst CROND[4434]: (user) CMD (/home/user/mail)
Apr 07 09:20:37 rst CROND[4479]: (user) CMD (/home/user/mail)
Apr 07 09:21:00 rst CROND[4514]: (user) CMD (/home/user/mail)
Apr 07 09:22:00 rst CROND[4551]: (user) CMD (/home/user/mail)
```

В остальных файлах `cron`, `cron1` и т.д. находится подобная информация, но более старая, чем в `cron`.

Контрольные вопросы

- 1). Поясните термин «демон».
- 2). Что понимается под термином «cron»?
- 3). Чем отличается `cron` от `crontab`?
- 4). Поясните формат файла `crontab`.
- 5). Поясните использование файла `cron.d`.
- 6). Какая информация содержится в файле `crontab`?
- 7). Поясните формат и значение полей пользовательского файла `crontab`.
- 8). Какой каталог сначала проверяет демон `cron` при запуске?
- 9). Сколько файлов `crontab` может иметь пользователь?
- 10). Сколько записей может быть в пользовательском файле `crontab`?
- 11). Если не будет указано значение переменной `SHELL`, из какого файла будет взято значение?
- 12). Где можно посмотреть журнал событий `cron` демона?

РАЗДЕЛ 6. АДМИНИСТРИРОВАНИЕ И УСТАНОВКА WEB-СЕРВЕРА

6.1. Директивы, используемые для конфигурирования web-сервера

Apache является наиболее распространенным web-сервером в мире. По данным компании Netcraft (<http://www.netcraft.com/Survey/>) общее число web-узлов, работающих под его управлением, к концу прошлого века достигало 2 млн., т.е. 55% от общего числа узлов, и это число постоянно растет. Для сравнения - на долю серверов Microsoft приходится 25%, Netscape - 7 %.

Будучи бесплатной и открытой программой, предназначенной для бесплатных Unix-систем (FreeBSD, Linux и др.), Apache по функциональным возможностям и надежности не уступает коммерческим серверам, кроме того, широкие возможности конфигурирования позволяют настроить его для работы практически с любой конкретной системой. Существуют локализации сервера для различных языков, в том числе и для русского.

Каждая строка конфигурационного файла, кроме строк, начинающихся с символа #, описывает какую-либо директиву конфигурации и ее значение. Строки, начинающиеся с символа # - это комментарии.

Описание основных директив конфигурирования web-сервера приведено в табл. 6.1. Более подробную информацию получить в документации по web-серверу Apache.

Таблица 6.1. Описание основных директив конфигурирования

Наименование директивы	Описание
ServerType standalone	Показывает тип запуска программы <i>httpd</i> - может вызываться через <i>inetd</i> (<i>inetd</i>) или как отдельный демон (<i>standalone</i>). По умолчанию вызывается как отдельный демон
Port 80	Порт, по которому http-сервер будет получать запросы. По умолчанию 80
HostnameLookups off	Определяет, будут ли записываться в лог-файлы имена (on) или только IP-адреса (off) хостов, просматривающих сайт. По умолчанию off
User ser nobody Group nobody	Определяет, под каким пользователем должен запускаться <i>httpd</i> . Для доступа к 80 порту <i>httpd</i> должен запускаться под root. Если в директивах User и Group указаны другие пользователи, то <i>httpd</i> все свои действия выполняет с правами этих пользователей, а не root.

ServerAdmin root@localhost	Здесь указывается <i>e-mail</i> администратора, отвечающего за работу сервера
ServerRoot /etc/httpd	Каталог, в котором находятся все конфигурационные файлы, лог-файлы и модули. Все пути, которые используются далее, указаны относительно этого каталога
ErrorLog logs/error_log	Местонахождение log-файлов об ошибках сервера
LogLevel warn	Уровень сообщений в log-файлах. Возможные значения: <i>debug, info, notice, warn, error, crit, alert, emerg</i>
LoadModule имя_модуля modules/имя_файла_модуля.so	Команда используется для загрузки различных модулей сервера Apache
ClearModuleList	Очистка списка модулей
AddModule имя_файла_модуля.c	Добавление модулей в список модулей
LogFormat "%h %l %u %t\"%r\" %>s %b \"%{Referer}i\" \"%{User- Agent}i\" combined	Описание различных форматов записи логов. Имя формата используется в директиве CustomLog
LogFormat "%h %l %u %t\"%r\" %>s %b" common	
LogFormat "%\{Referer}i->%U" referer	
LogFormat "% {User- Agent}i" agent	
LogFormat "формат" имя	
CustomLog logs/access_log common	Задаёт формат log-файла логов запросов. С помощью <i>CustomLog</i> можно задать файл логов любого формата и имени.
PidFile /var/run/httpd.pid	Имя файла, в котором хранится PID запущенного httpd. Используется для останова сервера и перезапуска

UseCanonicalName on	Разрешает использование коротких имен в URL. Удобно при использовании в Intranet-серверах. Если включено (on), преобразует, например строку <code>http://www/stat</code> в строку <code>http://www.user_домен.ru/stat</code> . Если отключено (off), преобразования не происходит.
Timeout 300	Время в секундах, по истечении которого при отсутствии запросов сервер прерывает связь с клиентом
KeepAlive On	Разрешать (on) или нет (off) множественные запросы через одно TCP-соединение
MaxKeepAliveRequests 100	Максимальное количество запросов через одно TCP-соединение
KeepAliveTimeout 15	Время ожидания следующего запроса в секундах
MinSpareServers 8 MaxSpareServers 20	Минимальное и максимальное значение загруженных процессов <code>httpd</code> . Если набрать в консоли команду: <code>ps ax grep httpd</code> , то можно просмотреть все процессы <code>httpd</code> , загруженные на машине. Если используется Apache на локальной машине, то необходимо уменьшить значение <code>MinSpareServers</code>
StartServers 10	Количество изначально запускаемых процессов
MaxClients 150	Ограничивает количество одновременно подключенных клиентов к серверу
MaxRequestsPerChild 100	Максимальное количество запросов, которые обрабатывает процесс до завершения работы
DocumentRoot /home/httpd/html	Путь к файлам сайта. Здесь хранятся те файлы, которые пользователь получает при наборе URL <code>http://твой_сайт.ru</code> . После установки там находится документация по Apache. Эти файлы следует заменить на файлы своего сайта

UserDir public_html	Имя каталога в домашней директории пользователя, где должны находиться файлы его веб-страницы, обращение к ним происходит по адресу <i>http://мой.сайт.ru/~имя пользователя</i>
DirectoryIndex index.html index.shtml index.cgi	Файлы индекса каталога - файлы, которые выводятся при задании в браузере адреса, являющегося ссылкой на каталог. Например, <i>http://localhost/manual</i> - выведет файл <i>index.</i> , <i>html</i> , находящийся в каталоге <i>/home/httpd/html/manual</i> . Если файл индекса отсутствует в каталоге, выводится все содержимое каталога. Можно задать значение <i>DirectoryIndex default.htm</i>
FancyIndexing on	При включении этой опции список файлов каталога будет выводиться в виде таблицы, в которой над каждой колонкой есть ссылка, при нажатии на которую список сортируется по этой колонке (например, по времени модификации или размеру)
AddIconByEncoding (CMP,/icons/compressed.gif) x- compress x-gzip AddIconByType (TXT,/icons/text.gif) text/* AddIcon /icons/binary.gif .bin .exe DefaultIcon /icons/unknown.gif	Эти директивы задают имя файла иконки в зависимости от типа, расширения и имени файла, которое будет выводиться перед именем файла в списке содержимого каталога
ReadmeName README HeaderName HEADER	<i>ReadmeName</i> - имя README-файла, который сервер ищет по умолчанию. <i>HeaderName</i> - имя файла, который включается в листинг каталога в качестве заголовка
IndexIgnore .??* *~ *# HEADER* README* RCS	Файлы, не включаемые в список каталога

AccessFileName .htaccess	Имя файла с правилами доступа, содержимое которого проверяется для каждого каталога
TypesConfig /etc/mime.types	Файл с описаниями типов файлов и принадлежащих к ним расширений
DefaultType text/plain,	Тип файла по умолчанию для тех файлов, тип которых неизвестен
AddEncoding x-compress Z AddEncoding x-gzip gz	Позволяет некоторым браузерам распаковывать "сжатые" и gzip- архивы "на лету", т.е. при копировании файла типа textcounter.tar.gz получается уже распакованный файл textcounter.tar
AddLanguage en .en AddLanguage fr .fr AddLanguage de .de AddLanguage da .da AddLanguage el .el AddLanguage it .it	Позволяет определить язык документа. Если на сайте есть английская и русская версии страницы, то можно создать файлы doc.html.ru и doc.html.en в зависимости от языка, прописанного в браузере клиента, ему будет показываться одна из страниц при обращении к doc.html. Расширение не обязательно должно совпадать с аббревиатурой языка
LanguagePriority en fr de	Определяет приоритет языка документа
Alias /icons/ /home/httpd/icons/	Разрешает создавать <i>алиасы</i> (альтернативные имена) для каталогов, т.е. если html-страница лежит в /home/httpd/html, то при обращении к http://mвой.caum.ru/icons Apache будет искать каталог /home/httpd/html/icons, а после определения такого алиаса Apache будет искать каталог /home/httpd/icons
ScriptAlias /cgi-bin/ /home/httpd/cgi-bin/	То же самое, что и Alias, но для серверных скриптов (CGI)
AddType text/html.shtml AddHandler server-parsed.shtml	Добавляет поддержку SSI для файлов с расширением .shtml

AddHandler imap-file map	Добавляет поддержку
BrowserMatch "Mozilla/2" nokeepalive BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0 BrowserMatch "RealPlayer 4\.0" force-response-1.0 BrowserMatch "Java/1\.0" force-response-1.0 BrowserMatch "JDK/1\.0" force-response-1.0	Прописывает различные параметры работы <i>httpd</i> при обращении к нему описанных браузеров
<Directory /> Options None AllowOverride None </Directory>	Описывает параметры доступа к корневой директории. Options контролирует то, какие опции доступны. Как видно из третьего примера – запрещено изменять опции доступа посредством содержимого файла <i>htaccess</i> каталога
<Directory /home/httpd/html> Options Indexes Includes FollowSymLinks AllowOverride None order allow,deny allow from all </Directory>	Устанавливает параметры доступа к каталогу <i>/home/httpd/html/</i> . Опции доступа (Options) следующие:
Indexes	если эта опция включена, то при указании URL, ссылающегося на каталог, например, <i>http://www.ru/nirvana</i> или <i>http://localhost/manual</i>), при отсутствии в каталоге файла, описанного директивой <i>DirectoryIndex</i> (<i>index.html</i> , <i>index.php3</i> , <i>index.cgi</i> и т.д.); <i>httpd</i> возвращает клиенту листинг этого каталога
Includes	разрешает выполнение SSI директив в файлах, описанных директивой <i>AddHandler server-parsed</i> тип и находящихся в каталоге <i>/home/httpd/html/</i>

FollowSymLinks	позволяет использовать символические ссылки на файлы или каталоги, не находящиеся в /home/httpd/html/
AllowOverride None	аналогично вышеописанному
Order allow, deny	определяет последовательность применения права запрета и правил разрешения доступа к каталогу. Здесь сначала проверяются разрешающие правила, затем запрещающие, т.е. если разрешено, например, по IP, и запрещено, например, по имени хоста, то будет запрещен доступ к этому каталогу, т.к. запрещающие правила сработали последними
Allow from all	разрешен доступ отовсюду
<pre><Directory /home/httpd/cgi~bin> AllowOverride None Options ExecCGI </Directory> Alias /doc /usr/doc <Directory /usr/doc></pre>	прописывает параметры доступа к каталогу /home/httpd/cgi-bin (т.е. где лежат CGI-скрипты). AllowOverride None - см. выше. Options ExecCGI - разрешен запуск CGI и больше ничего

6.2. Настройка виртуальных серверов в файле httpd.conf

В большинстве случаев один http-сервер способен обрабатывать запросы, поступающие на различные, так называемые виртуальные, web-серверы. Виртуальные серверы могут иметь как один и тот же IP-адрес, но разные доменные имена, так и разные IP-адреса.

С точки зрения пользователя второй вариант чуть более предпочтителен, поскольку запрос к серверу, отличающемуся от основного только доменным именем, должен содержать его имя, а некоторые старые браузеры, не поддерживающие протокол HTTP/1.1 (например, Microsoft Internet Explorer 2.0), не включают в запрос эту информацию. Однако такие браузеры выходят из употребления (сейчас их уже менее 0,5 % от общего числа).

С другой стороны, выделение собственного IP-адреса каждому виртуальному серверу может быть неоправданной растратой адресного пространства компании.

Для описания адресов и доменных имен виртуальных серверов служат директивы ServerName, ServerAlias, NameVirtualHost и VirtualHost. Они необходимы, только если пользователю нужно установить более одного виртуального сервера.

Директива `ServerName`, находящаяся вне секций `VirtualHost`, определяет имя основного сервера, т.е. сервера, корневого каталог которого задан директивой `DocumentRoot` в файле `srm.conf`. Виртуальные серверы наследуют настройки основного; при необходимости специальной настройки соответствующие директивы помещаются в секции `VirtualHost`, относящейся к данному серверу. Допустимы любые директивы, которые могут встретиться в файлах `httpd.conf` и `srm.conf`, например `DocumentRoot`, `ErrorLog`, `CustomLog`, `Location`, `ServerAdmin`.

Для редактирования файлов конфигурации и навигации по файловой системе удобно использовать программу *mc*, для ее загрузки необходимо набрать в командной строке:

```
[root@lis] $ mc
```

Интерфейс программы интуитивно понятен и не представляет трудностей при использовании. Файлы конфигурации web-сервера Apache в данной системе находятся в каталоге:

```
/etc/httpd/conf/
```

Для запуска, остановки, перезапуска web-сервера используются следующие скрипты (программы):

- остановка: `/etc/rc.d/init.d/httpd stop;`
- запуск: `/etc/rc.d/init.d/httpd start;`
- перезапуск: `/etc/rc.d/init.d/httpd restart.`

6.3. Инструмент настройки Apache

Инструмент настройки Apache позволяет настроить файл `/etc/httpd/conf/httpd.conf` - файл настроек Web сервера Apache. Он не использует старые файлы конфигурации, такие как `srm.conf` или `access.conf`; их необходимо оставить пустыми. В графическом интерфейсе можно настраивать параметры Apache, такие как виртуальные узлы, атрибуты протоколирования и максимальное количество соединений.

Модули, поставляемые с Red Hat Linux, могут быть настроены с помощью Инструмента настройки Apache. Если установлены дополнительные модули, их нельзя настроить с помощью этой программы.

Чтобы использовать *Apache Configuration Tool* (инструмент настройки Apache), требуется запустить систему X Window под именем root. Запустить Apache Configuration Tool можно следующими способами:

1) Нажав на рабочем столе GNOME Кнопку Main Menu (Главное меню) (на панели) => Programs (Программы) => System (Система) => Apache Configuration (Настройка Apache).

2) Нажав на рабочем столе Кнопку Main Menu (Главное меню) (на панели) => Red Hat => System (Система) => Apache Configuration (Настройка Apache).

3) Выполнив в приглашении оболочки (например, в XTerm или GNOME-terminal) команду *apacheconf*.

Ниже перечислены основные этапы настройки веб-сервера Apache, используя Инструмент настройки Apache:

- 1) Определить основные параметры на вкладке Main (Основные);
- 2) Перейти на вкладку Virtual Hosts (Виртуальные узлы) и настроить параметры по умолчанию;
- 3) На вкладке Virtual Hosts настроить Default Virtual Host (Виртуальный узел по умолчанию). Если будет обслуживаться несколько URL или виртуальных узлов, определить дополнительные виртуальные узлы;
- 4) Настроить параметры сервера на вкладке Server;
- 5) Определить настройки соединений на вкладке Performance Tuning (Настройка производительности).
- 6) Скопировать все необходимые файлы в каталоги /DocumentRoot и /cgi-bin, затем сохранить настройки в Инструменте настройки Apache.

Apache может использовать модуль *mod_env*, чтобы настроить переменные окружения, передаваемые в CGI-сценарии и страницы SSI. На странице *Environment Variables* (Переменные окружения) можно определить параметры этого модуля Apache (рис. 6.1).

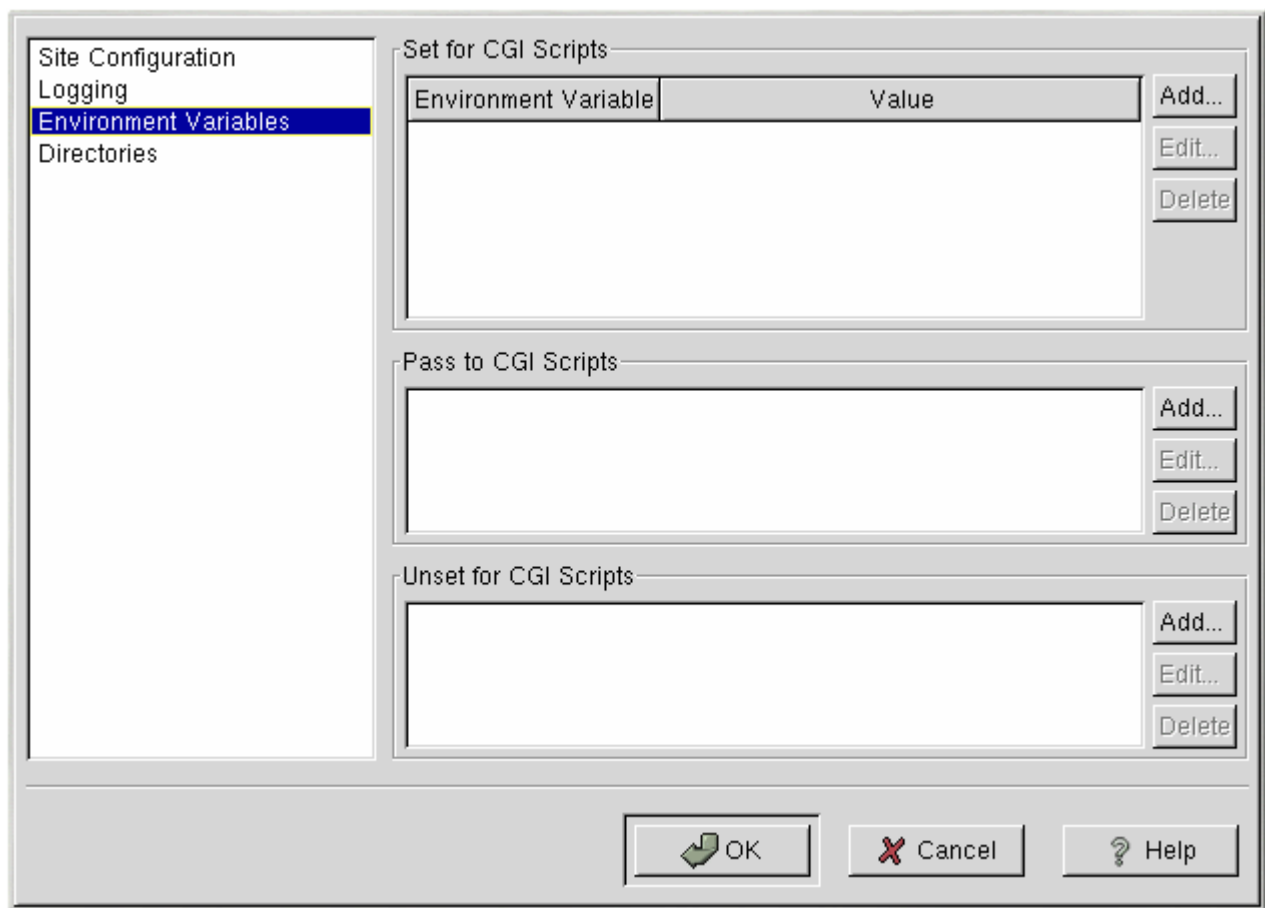


Рис.6.1. Экранная форма «Переменные окружения»

6.4. Протоколирование

По умолчанию Apache записывает протокол передачи информации в файл /var/log/httpd/access_log, а протокол ошибок в файл /var/log/httpd/error_log.

Протокол передачи информации содержит список всех попыток обращения к Web серверу. В нём фиксируется IP-адрес клиента, пытавшегося соединиться с сервером, дата и время этой попытки, а также имя файла на web-сервере, к которому клиент пытался обратиться. Следует ввести полный путь и имя файла, в котором будет сохранена эта информация. Если полный путь и имя файла начинается не с косой черты (/), путь будет определяться относительно заданного каталога `server root`. Этот параметр соответствует указанию `TransferLog` (рис.6.1).

Пользователь может определить особый формат протокола, установив флажок *Use custom logging facilities* (использовать возможности настройки протоколирования) и задав формат протокола в поле *Custom Log String* (строка формата протокола). При этом будет определен параметр `LogFormat`. В файле http://httpd.apache.org/docs/mod/mod_log_config.html#formats содержится описание формата этого параметра.

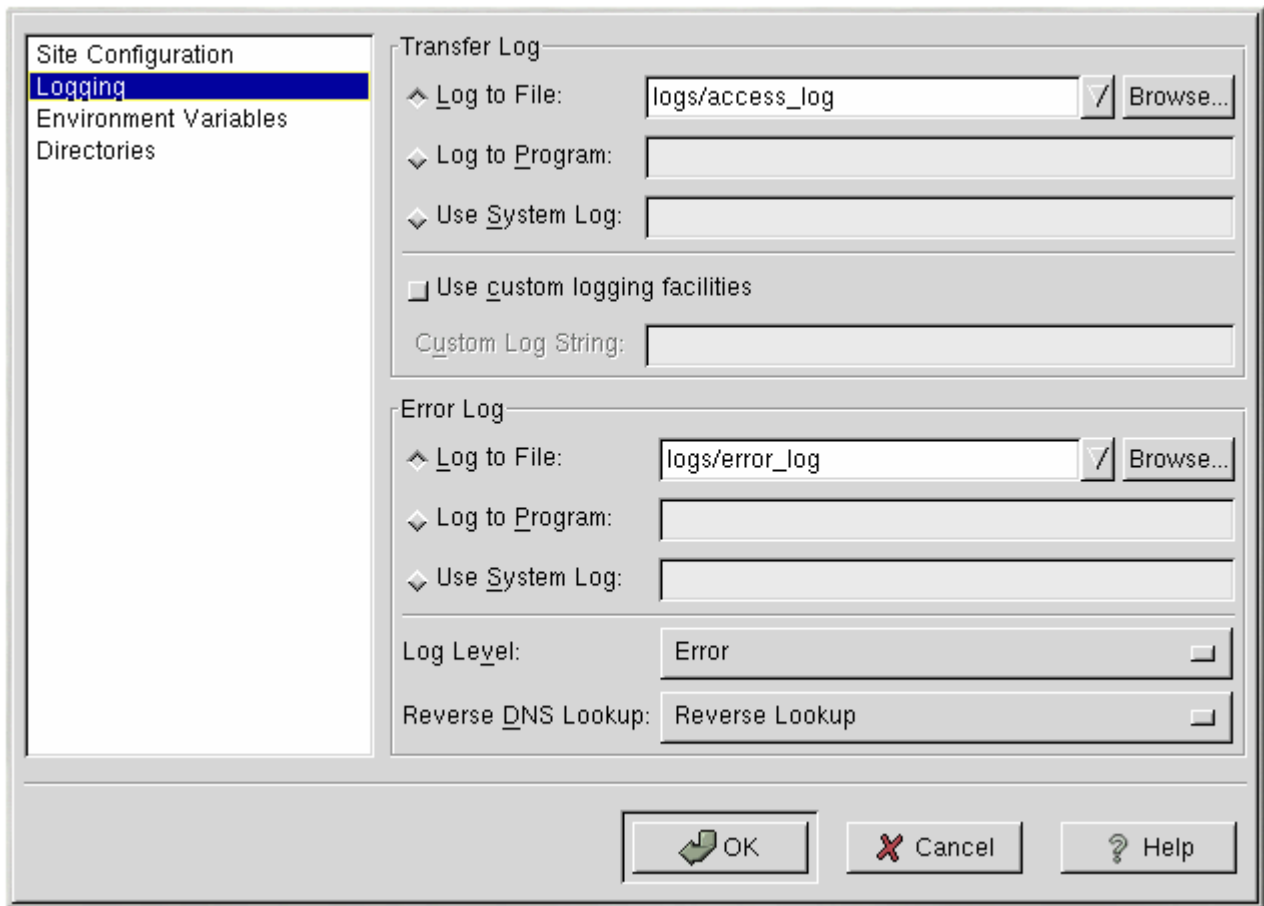


Рис.6.2. Экранная форма протоколирования

Протокол ошибок содержит список всех произошедших ошибок сервера. Также следует ввести полный путь и имя файла, в котором будет сохранена эта информация. Этот параметр соответствует указанию `ErrorLog`.

Чтобы определить, насколько подробны в протоколе будут сообщения об ошибках, следует использовать меню *Log Level* (уровень протоколирования). Этот параметр может принимать значения:

- emerg (аварийные ситуации),
- alert (тревожные ситуации),
- crit (критические),
- error (ошибки),
- warn (предупреждения),
- notice (уведомления),
- info or debug (информация или отладка).

Этот параметр соответствует указанию *LogLevel*.

Значение, выбранное в меню *Reverse DNS Lookup* (обратное преобразование в DNS), определяет указание *HostnameLookups* (разрешение имени узла). Установить его значение равным "off" можно, выбрав *No Reverse Lookup* (без обратного преобразования), выбрав *Reverse Lookup* (обратное преобразование), пользователь установит значение равным "on".

Выбрав двойное обратное преобразование *Double Reverse Lookup*, пользователь установит значение равным "double". Если выбрано обратное преобразование *Reverse Lookup*, сервер будет автоматически разрешать IP-адреса для каждого соединения при запросе документа web-сервера.

Разрешение IP адресов означает, что сервер будет производить одно или несколько соединений с DNS, чтобы выяснить имя узла, соответствующее определенному IP адресу. Если выбрано *Double Reverse Lookup* - сервер будет выполнять двойной обратный поиск в DNS. Другими словами, для полученного после выполнения обратного преобразования имени будет выполнено прямое преобразование. По крайней мере, один из IP адресов при прямом преобразовании должен будет совпасть с адресом, обратное преобразование которого производилось.

Обычно следует установить этот параметр в значение *No Reverse Lookup* без обратного преобразования, потому что запросы DNS дополнительно загружают сервер и могут снизить его производительность. Если ваш сервер загружен, влияние таких прямых и обратных преобразований может быть довольно заметным.

Обратные и двойные обратные преобразования также являются проблемой для интернета в целом. Все единичные подключения, сделанные для выполнения преобразования, объединяются. Поэтому для пользы собственного web-сервера, а также всей Глобальной сети следует установить для этого параметра значение *No Reverse Lookup*.

6.5. Проверка работоспособности web-сервера

Для демонстрации работоспособности необходимо переписать с <ftp://ftp.msctuca.ru/data/sop/> средствами Linux на локальный компьютер каталоги *testserver1* и *testserver2*, для чего использовать в программе *mc* пункт меню *left* (*right*), далее *ftp link*. В данных каталогах содержатся тестовые сайты:

- проверка исполнения файла `/cgi-bin/test1.cgi`;
- проверка исполнения файла `/cgi-bin/test2.pl`;
- проверка исполнения файла `/cgi-bin/test3`;
- проверка SSI-включения файла `/cgi-bin/test1.cgi` в файл `test4.html`.

Для создания виртуальных серверов по доменным именам используемые доменные имена необходимо прописать в файле `c:\winnt\system32\driver\etc\hosts` на том компьютере, на котором будет проводиться проверка.

6.6. Инсталляция SAMS

Ниже приведен порядок процесса инсталляции SAMS.

Распаковать архив: `tar xzf sams-xxxxxx.tar.gz` . Будет создан каталог, в котором будет размещено содержимое архива.

Перейти в созданный каталог `cd sams-xxxxxx`

Собрать SAMS. Для настройки конфигурации ввести команду `./configure` .SAMS по умолчанию находится в `/usr/local`.

Для изменения пути к расположению программ и пути к расположению библиотек и файлов заголовков MySQL, к каталогу расположения root директории http-сервера (если путь к нему отличается от `/var/www/html` или `/var/www/htdocs`), к каталогу расположения php следует использовать ключи `configure`.

Для получения списка опций настройки введите команду:

`./configure --help`

Если на этапе конфигурирования выдаются ошибки, следует проверить наличие необходимых SAMS пакетов.

По завершении работы команды `configure` необходимо откомпилировать дистрибутив и установить его:

`make`

`make install`

Если на этапе сборки выдаются ошибки, необходимо проверить наличие необходимых SAMS пакетов.

По окончании инсталляции файлы SAMS будут проинсталлированы:

`/etc:`

`sams.conf` - файл конфигурации SAMS

`/usr/local/sams:`

файлы:

`sams` - анализатор логов `squid`

`samsf` - демон, создающий `fifo` файл, в который `squid` записывает логи.

`samsdaemon` - демон, отвечающий за переконфигурацию `squid` и автоматический запуск `sams` и `samsf`

`/usr/local/share/sams:`

файлы web-интерфейса `sams`

В результате инсталляции должен быть создан симлинк из *root* каталога http сервера (apache) на каталог web-интерфейса SAMS. Если он не создан, необходимо создать симлинк из каталога вашего web-сервера на каталог /usr/local/share/sams

```
ln -s /usr/local/share/sams /our/path/www/htdocd/sams
```

Для выполнения запуска web-интерфейса SAMS и создания базы SAMS в MySQL следует запустить браузер и подключиться к web-интерфейсу SAMS:

```
http://localhost/sams
```

Следует заметить, что httpd-сервер должен поддерживать работу php скриптов. Для работы web-интерфейса SAMS необходимо установить модули php и настроить работу php в *safe_mode*.

Если базы SAMS еще не созданы, пользователю будет предложен диалог создания баз и пользователя SAMS в MySQL. Для этого необходимо заполнить поля:

MySQL Hostname: - адрес сервера, на котором установлен MySQL,

MySQL login: - имя пользователя MySQL, имеющего права на создание баз данных (обычно это root);

MySQL password: - пароль пользователя root;

Create SAMS MySQL user - установить галочку, если необходимо создать пользователя, от имени которого будет работать SAMS;

SAMS MySQL user: ввести имя пользователя, от имени которого будет работать SAMS (если MySQL расположен на этом же хосте, то это будет, например, *sams@localhost*);

SAMS MySQL user password: Пароль пользователя *sams*.

Далее следует занести введенные значения в файл конфигурации SAMS /etc/sams.conf:

MYSQLEHOSTNAME=localhost - имя хоста, где стоит MySQL ,

MYSQLEUSER=sams - имя пользователя MySQL, от имени которого будет работать SAMS (внимание, полный адрес: *sams@localhost* не указывать),

MYSQLEPASSWORD=yourpasswd - Пароль пользователя в MySQL

Далее активизировать кнопку "Create database", и, если поля формы были заполнены правильно, базы и пользователь SAMS будут созданы, затем кнопку "Starting SAMS web interface". Если данные в файл конфигурации *sams.conf* были занесены верно, то будет выведен web-интерфейс SAMS.

Следующий шаг - изменить владельца каталога /usr/local/share/sams на пользователя, от имени которого работает ваш web сервер:

```
chown -R apache:apache /usr/local/share/sams
```

Далее следует сменить права доступа на каталог /usr/local/share/sams/data на 777.

Необходимо настроить *sams* и *squid* на работу с авторизацией в OpenLDAP.

Проверить авторизатор SQUID на работу с OpenLDAP. Для этого нужно запустить:

/usr/lib/squid/ldap_auth -R -v 3 -d -b ou=Users, dc=example, dc=local -f uid=%s

Далее ввести вручную имя и пароль пользователя (через пробел). Если все работает корректно, то будет выдан ответ:

OK

Открыть на редактирование файл */etc/squid/squid.conf* и добавить строки:

auth_param ntlm program /usr/lib/squid/ldap_auth -R -v 3 -d -b ou=Users, dc=example, dc=local -f uid=%s

auth_param ntlm children 5

auth_param ntlm max_challenge_reuses 0

auth_param ntlm max_challenge_lifetime 2 minutes

auth_param basic program /usr/lib/squid/ldap_auth -R -v 3 -d -b ou=Users, dc=example, dc=local -f uid=%s

auth_param basic children 5

auth_param basic realm Squid proxy-caching web server

auth_param basic credentialsttl 2 hours

Теперь необходимо сконфигурировать SAMS на работу с OpenLDAP. Открываем на редактирование */etc/sams.conf* и добавляем следующие строки:

LDAPSERVER=127.0.0.1 - имя или ip адрес сервера OpenLDAP

LDAPBASEDN=dc=example,dc=local - домен, поддерживаемый OpenLDAP. Ввод осуществляется без кавычек или пробелов.

LDAPUSER=cn=admin,dc=example,dc=local - пользователь, входящий в группу OpenLDAP администраторов, от имени которого SAMS будет работать с OpenLDAP

LDAPUSERPASSWD=12345 - пароль

LDAPUSERSGROUP=ou=Users,dc=example,dc=local - контейнер, содержащий пользователей в OpenLDAP (обычно это Users)

LDAPUSEFILTER=(objectClass=SambaSamAccount) -фильтр для отсека «не пользователей».

Контрольные вопросы

- 1) Какие файлы содержат конфигурационную информацию web-сервера?
- 2) Какова последовательность установки web-сервера?
- 3) Как проверить работоспособность web-сервера?
- 4) Где хранятся log-файлы?
- 5) Что такое виртуальный web-сервер?
- 6) Что определяет обратное преобразование в DNS?
- 7) При каком преобразовании сервер будет автоматически разрешать IP- адреса для каждого соединения?
- 8) Какая информация содержится в протоколе ошибок?
- 9) Перечислите основные директивы конфигурирования.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

- 1) Скотт Граннеман. Linux. Необходимый код и команды. Карманный справочник. - М.: Вильямс, 2010. - 416 с.
- 2) Вахалия Ю. UNIX изнутри. – СПб.: Питер, 2003. –844 с.
- 3) Митчелл М., Джеффри Д., Самьюэл А. Программирование для Linux. Профессиональный подход/ перевод с англ. - М.: Вильямс, 2003. – 288 с.
- 4) Керниган Б.В., Пайк Р. UNIX - универсальная среда программирования. - М.: Финансы и статистика, 1992. -304 с.
- 5) Браун П. Введение в операционную систему UNIX. - М.: Мир, 1987. - 287 с.
- 6) Банахан М., Раттер Э. Введение в операционную систему UNIX. - М.: Радио и связь, 1986. -341 с.
- 7) Баурн С. Операционная система UNIX. - М.: Мир, 1986. -462 с.
- 8) Готье Р. Руководство по операционной системе UNIX. - М.: Финансы и статистика, 1985. -232 с.
- 9) Кристиан К. Введение в операционную систему UNIX. - М.: Финансы и статистика, 1985. -318 с.
- 10) Томас Р., Йейтс Дж. Операционная система UNIX. Руководство для пользователей. - М.: Радио и связь, 1986. -352 с.
- 11) Топхем Д., Чьюнг Х.В. Юникс и Ксеникс. - М.: Мир, 1988. -392 с.
- 12) Робачевский А.М. Операционная система UNIX. - СПб.:BNV-Санкт-Петербург, 2007. -528 с.
- 13) Беляков М.И., Рабовер Ю.И., Фридман А.Л. Мобильная операционная система. - М.: Радио и связь, 1991. - 208 с.
- 14) Тихомиров В.П., Давидов М.И. Операционная система UNIX: Инструментальные средства программирования. - М.: Финансы и статистика, 1988. - 206 с.

ИТОГОВЫЙ ТЕСТ

- 1) Язык, на котором написана большая часть кода Linux/UNIX:
а) C б) Pascal
в) Ada г) Basic
- 2) Какую дату можно считать официальной датой рождения UNIX?
а) 31 декабря 1974 б) 30 октября 1984
в) 1 января 1970 г) 1 сентября 1971
- 3) Какой из дистрибутивов Linux производится в России?
а) Debian б) ASPLinux
в) Red Hat г) Slackware
- 4) Какая из этих операционных систем не является POSIX-совместимой?
а) IRIX б) BeOS
в) MacOS X г) OS/2
- 5) Какие права установлены для группы?
-rwxr---w- 1 root root 93 11 apr 2011 file.txt
а) Чтение и запись б) Чтение
в) Запись г) Чтение, запись, исполнение
- 6) Файловая система Ext3 в отличие от файловой системы Ext2:
а) Журналируемая б) Виртуальная
в) Псевдо-файловая г) Сетевая
- 7) На разделы какой файловой системы Linux может записывать файлы?
а) ISO9660 б) NTFS
в) HPFS г) ReiserFS
- 8) На разделы какой файловой системы Linux не может записывать файлы?
а) Ext2 б) NTFS
в) msdos г) ReiserFS
- 9) Приведены метаданные 2-х файлов. Что можно сказать об этих файлах?
/home/sergey /home/ivan
12567 file1 12567 file2
- 10) Сколько корневых каталогов может быть в UNIX?
а) 1 б) 2
в) 26 г) неограниченно
- 11) Что содержит каталог /dev в UNIX?
а) файлы пользователей б) файлы, необходимые для загрузки
в) файлы устройств г) конфигурационные файлы
- 12) В каком файле содержатся данные о монтируемых устройствах и точках монтирования?
а) /etc/fstab б) /etc/inittab
в) /etc/crontab г) /etc/motd
- 13) Для кого установлено право на запись в файл?
-rwxr-x-r-- 1 root root 93 1 янв 2003 file.txt

- а) для группы
в) для остальных
б) для владельца
г) для всех
- 14) Команда *chmod u=4 file* установит атрибуты файла *file*:
а) Чтение для владельца
б) Запись для владельца
в) Исполнение для группы
г) Чтение и запись для владельца
- 15) Команда *chmod o=2 file* снимет атрибуты файла *file*:
а) Чтение для остальных
б) Запись для владельца
в) Запись для остальных
г) Исполнение для группы
- 16) Команда *ls* предназначена для вывода информации:
а) о текущем пользователе
б) о файлах
в) о дисках
г) о процессах
- 17) Опция команды *ls* для отображения файлов в длинном формате:
а) -a
б) -l
в) -d
г) -f
- 18) Какая команда является внутренней в интерпретаторе?
а) pwd
б) cd
в) ls
г) df
- 19) Команда для просмотра информации о смонтированных устройствах:
а) ps
б) df
в) wc
г) pwd
- 20) Какой процесс запрашивает имя пользователя при загрузке?
а) login
б) getty
в) lpd
г) cron
- 21) Какой символ используется в команде при создании неименованного канала между процессами:
а) &
б) >
в) |
г) ~
- 22) При выполнении команды *\$ cat file1 | w* происходит:
а) создание неименованного канала
б) перенаправление ввода
в) последовательный запуск программ
г) запуск в фоновом режиме
- 23) Команда, позволяющая получить список всех файлов в системе, не имеющих расширения:
а) *find / -name ".*" -print*
б) *find ~ -name "*.txt" -print*
в) *find / -name "*" -print*
г) *find ~ -name ".*" -print*
- 24) Какие директивы конфигурирования сервера Apache определяют, под каким пользователем должен запускаться *httpd*:
а) *User ser nobody*
б) *Group nobody*
в) *<Directory /home/httpd/html>*
г) *<Directory /home/httpd/html>*
Options Indexes
Options FollowSymLinks
- 25) Какая команда осуществляет поиск всех адресов, содержащих две трехзначные последовательности, оканчивающиеся точкой:
а) *grep '[0-9]{3}\.[0-9]{3}.' Ipfile*
б) *find '[0-9][0-9]. Ipfile*
в) *grep '[0-9].[0-9]{3}.' Ipfile*
г) *find '[0-9].[0-9]{3}.' Ipfile*