Лабораторная работа № 2

Поиск текста в файлах по образцу.
Сценарии в командном интерпретаторе BASH.
Команда test и условные операторы в сценариях

***Цель работы*:** применение сложного синтаксиса команд, сортировка и поиск текста в файлах по образцу, изучение структуры сценариев командного интерпретатора **bash**, использования переменных командного интерпретатора; создание **shell**-сценариев с использованием переменных сценария, использование команды тестирования; создание shell-сценариев с применением сложного синтаксиса, команд условия и условных операторов.

Продолжительность работы - 4 ч.

Теоретические сведения

Поиск текста в файлах по образцу

При составлении сценариев в ряде случаев полезно использовать функции поиска текста в файлах по образцу. Их выполняют три команды-фильтра **grep, fgrep, egrep**, имеющие отличия в функциональном назначении. В последних версиях командных интерпретаторов увеличилось количество общих свойств этих функций.

Общая структура этих команд следующая:

$ команда шаблон опции каталоги_файлы

Шаблоны - фрагменты текста, по которым ведется поиск в файлах. Команды имеют разный синтаксис шаблонов. Имя каталога можно не указывать, если файлы находятся в текущем каталоге. Команда **fgrep** работает для большего количества образцов и файлов, причем быстрее, чем **grep**, но она не работает с ***регулярными выражениями***, как **grep**. Команда **egrep** объединяет возможности **grep** и **fgrep**, а также может применять в образцах расширенный набор специальных символов. В последних версиях КИ Bash все эти возможности присутствуют в команде **grep**.

В шаблонах можно использовать ***регулярные выражения*** - образцы со специальными символами, формирующие шаблон и использующие следующие символьные обозначения:

^ - циркумфлекс означает начало строки в описании шаблона;

$ - знак доллара означает конец строки в описании шаблона;

. - знак точка применяется для сопоставления в образце с любым одиночным символом;

[] - квадратные скобки, используются для сравнения семантически схожих классов символов;

***** - звездочка, используется для сравнения произвольного количества повторяющихся символов.

Структура команды **grep** следующая:

$ grep образец имя_файла

$ grep 'образец из нескольких слов' имя_файла

Опции команды **grep** (опции не являются обязательными):

i - выполнить поиск без учета регистра,

n - вывести номера строк, в которых найден образец,

c - вывести на стандартный вывод (обычно на терминал) количество строк, соответствующее образцу,

v - вывести строки, не содержащие образца,

l - вывести имена файлов, которые содержат строки, совпадающие с образцом.

Пример 1. Определить, есть ли в файле **letter** строки, начинающиеся на **Hello**

$ grep Hello letter

Hello, my dear frend - пример результата вывода.

Команду можно использовать для поиска в нескольких файлах, тогда результат поиска выводится с указанием имени файла. Если в образце несколько слов, то в шаблоне необходимо использовать одинарные кавычки:

$ grep 'Hello, my dear frend' fletter letter letter2

letter:Hello, my dear frend - пример результата вывода.

fletter: Hello, my dear frend

Пример 2. Поиск слова **while** во всех файлах с расширением **cpp**, расположенных в текущем каталоге:

$ grep while *.cpp

Пример 3. Для команд **grep** и **egrep** применяются следующие правила формирования шаблона, использующие регулярные выражения:

$ grep '^первое_слово_строки_файла' имя_файла;

$ grep 'последнее слово строки файла$' имя_файла;
$ grep 'a*' - ищутся строки с повторяющимися символами a;
$ grep 'a.l' - ищутся строки, где есть словосочетания с неизвестной любой буквой, расположенной между **a** и **l**;

Пример 3. Найти строки со словами, начинающимися с a:

$ grep 'a.'

Структура команды **egrep**:

$ egrep образцы список_файлов

Команда **egrep** объединяет возможности **grep** и **fgrep,** а также может применять в образцах расширенный набор специальных символов:

(образцы) - круглые скобки для группирования образцов;

| - логический оператор **И**;

символ? -.поиск одного или нуля предыдущих символов;

символ+ - поиск одного или нескольких повторений предыдущего символа.

Пример 4. Если нужно в тексте найти образец, включающий специальные символы, то знак обратного слеша перед символом позволяет считать его символом текста:

$ egep 'Hello\!' fileletter

Hello! - пример результата вывода

Пример 5. Найти строки с шаблоном **a.x**, вместо знака точка подразумевается произвольный символ:

$ egrep 'a.x' file110
$ grep 'a.x' file110

Пример 6. Найти файлы со строками, в которых есть слова, начинающиеся с символа **a,** или символов **aa** или **aaa** и т.д.

$ egrep 'a*' file
$ grep 'a*' file

Пример 7. Поиск слов **big** или **pig**:

$ egrep '[bp]ig' file
$ grep '[bp]ig' file

Пример 8. Поиск одного или нескольких символов **t**:

$ egrep 't+' file8

Пример 9. Поиск одного или 0 символов **b**:

$ egrep 'b?' file9

Пример 10. Поиск описаний блок-ориентированных устройств (при выполнении **$ ls -l** строки каталогов начинаются на символ **d**, строки с именами файлов начинаются на символ "короткое тире" (**-**), строки с именами блок-ориентированных устройств - на **b**):

$ ls –l | grep '^b'

Пример 11. Поиск слов **my1**, **my2**, **my3**, и т.д. до **my8**

$ grep 'my[1-8]' file

Пример 12. Поиск слов в файле из диапазона, включающего цифры от 1 до 8 (**my1**, **my2**, **my3** до **my8)**, и диапазона, включающего символы a, b, c **mya**, **myb**, **myc**

$ grep 'my[1-8a-c]' file

Пример 13. Поиск по двум шаблонам:

$ egrep 'while | for /usr/include

Пример 14. Если в образец включены специальные символы, они добавляются в кавычках, так как в качестве специальных символов регулярного выражения и групповых символов командного интерпретатора используются одни и те же символы *, [].

$ ls –l | grep '^-'

выйдут на печать или будут отображены на экране файлы для чтения:

-rwxr-x--- user1 512 Feb 10 8 34 file1.

$ ls – l | grep '^d'

На экран выводится список содержимого текущего каталога:

drwxr-x--- user user2 512 Feb 12 8 32 file11

Структура команды **fgrep**:

$ fgrep образец большой_список_файлов

Команда **fgrep** не использует символы ^, $, *, •, [], \ и регулярные выражения, включающие эти символы. Команда **fgrep** для регулярных и полных регулярных выражений не работает, но **fgrep** - очень быстрая команда для известных ей выражений. ОС Linux внешне не реагирует на эту команду, если встречаются непонятные ей выражения в образцах.

Процесс

Любая выполняемая команда в ОС Linux и ОС UNIX порождает **процесс** - задачу, выполняемую системой. Чтобы посмотреть, какие процессы идут в системе, следует набрать команду: **$ ps** без опций.

В первой колонке появившегося списка, представляющего собой таблицу, указан хозяин процесса UID, во второй - системный номер процесса PID, или идентификатор процесса, в третьей - время выполнения процесса,

в четвёртой колонке – на какое внешнее устройство предполагается вывод, в последнем столбце списка - имя процесса, соответствующее названию команды. Порядок и количество описываемых характеристик зависит от КИ и его версии.

Задание 1. Выполните несколько упражнений по управлению процессами. Возможно, Вы сможете войти в две консоли (терминала) без регистрации, это зависит от установок в системе. Откройте их. Запустите редактор Vim в одной из них и введите небольшой текст. Не записывая текст в файл, переключитесь в другую консоль и просмотрите список всех запущенных процессов с помощью команды: **$ ps** (с опцией **-a**). Выясните, какая программа сейчас активна и занимает больше памяти.

Найдите **PID** для запущенного редактора **Vim**, для этого, если список процессов большой, используйте команды **grep** и **ps**: **$ ps | grep vim**.

Закройте **Vim** с помощью команды **kill** или **killall**. Запустите **Vim** ещё раз, вернувшись по истории команд на шаг обратно.

*Программные каналы, перенаправление входных и выходных **потоков данных***

Многие команды **ОС Linux** принимают данные со стандартного ввода или передают на стандартный вывод. По умолчанию в качестве устройства для стандартного ввода используется клавиатура, для стандартного вывода - терминал. Перенаправление стандартного вывода в файл или на устройство обозначается знаком "больше" (**>**). Стандартный ввод может быть получен не с клавиатуры, а из файла. Оператором перенаправления стандартного ввода является знак "меньше" (<).

Пример 1. Вывести на экран содержимое файла **file** и записать в новый файл **file1** можно, используя перенаправление стандартного ввода-вывода, представляющего байтовый поток. Чтение данных из файла **file** и запись в **fi1le1** в командной строке осуществляются следующим образом:

$ cat < file > file1

То же самое, но только с опцией **-i** - проверки наличия файла **filef**:

$ cat < file > -i file1

Поскольку файл уже создан, появится сообщение о наличии файла **file1**, нужно ответить **n**, чтобы не изменять уже существующий файл, или **y**, чтобы записать в него.

Если требуется добавить информацию в файл, т.е. осуществлять запись в него не с начала файла, а в конец уже имеющейся записи в файле, то используется оператор добавления. Допишем в **fi1e1** информацию из файла **file2**:

$ cat file2 >> file1

ОС Linux поддерживает стандартный вывод сообщений об ошибках (**2>**), отличающийся от стандартного вывода, результат выводится в файл и в зависимости от настроек на экран.

Пример 2. Чтобы сообщения об ошибках выполнения команды **cp** записать в файл **error**, следует выполнить команду:

$ cp fileprog /k1 2> error

Чтобы добавить сообщения об ошибках в файл **error,** вместо знака (>) предыдущей команды нужно использовать знак добавления (>>):

$ cat fileinfo 2>> error

Ошибка могла возникнуть, если указанные файлы не существуют.

Для передачи результат выполнения команды в другую команду используется символ вертикальной черты **|** - так называемый программный канал. Другими словами, он передаёт выходные данные команды на вход следующей команды.

Пример 3. Передать в файл **filelist** список имен файлов текущего каталога, выданный командой **ls**. Передача данных в данном примере осуществляется по программному каналу: **$ ls | cat > filelist**.

Сжатие и архивирование файлов

Архивирование выполняется для объединения файлов в один пакет, что часто необходимо для пересылки большого количества небольших файлов по сети, а также для архивирования. Группа файлов имеет меньший объем, чем сумма объемов файлов, что особенно заметно, когда файлы однотипные и их достаточно много. При сетевой пересылке быстрее обработать группу файлов, чем каждый по-отдельности. Сжатие выполняется для уменьшения размеров файлов, предназначено для хранения и пересылки больших файлов, его алгоритмы аналогичны алгоритмам сжатия, используемым в приложениях **ОС Windows**.

Задание 2. Для изучения возможностей команд архивирования и сжатия проделайте следующие упражнения. Имена файлов и каталогов можете использовать свои.

1. Создайте в каталоге **k1** подкаталог **k2,** включите в него файлы **file1**, **file2**, **file3**. Заархивируйте (без сжатия) командой **tar** эти 3 файла, архив поместить в том же каталоге, что и файлы:

$ tar cvf имя_каталога/имя_архива.tar имена_файлов

Обратите внимание, что в этой команде опции используются без знака минус: **v** - отображение имени каждого архивируемого в данный момент файла, **c** - создание нового архива, **f** - запись в архив по указанному

имени, а не на стандартное устройство по умолчанию, эта опция пишется последней из опций.

2. Добавьте в архив еще один файл (из ранее созданных или доступных в системе):

$ tar rvf имя_каталога/имя_архива.tar имя_файла.

5. Извлеките файл **file3** из архива и поместите в каталог **k1**:

$ tar xvf имя_архива.tar file3 -C /home/user5/k1/

6. Просмотрите файлы, хранящиеся в архиве:

$ tar tvf имя_каталога/имя_архива.tar

7. Сжатие файлов наиболее часто выполняется командой-утилитой **gzip,** распаковывание - утилитой **guzip** Сжатые файлы имеют имя такое же, как у файла, расширение - **.gz**.

Опции команды **gzip: v** сообщает степень сжатия каждого файла в процентах, где опция **-число***,* которое устанавливает степень сжатия, число от 1 до 9, чем больше число, тем больше степень сжатия, тем дольше выполняется сжатие; по умолчанию степень сжатия равна 6. Команда **gzip** является утилитой, поскольку имеет большие функциональные возможности. Опция **-h** выдаёт перечень справочной информации по ним.

Попробуйте сжать Ваш файл, как указано в примере использования утилиты: **$ gzip -7 file3**

Для распаковывания вместо **guzip** можно использовать **gzip** с опцией **–d**. Файлы сжатия и архивирования имеют расширения. Известная уже команда **$ ls** выводит список имён файлов с их расширениями.

8. Выведите на экран содержимое сжатого файла, для чего воспользуйтесь командой **zcat** и направьте вывод в команду отображения:

$ zcat file3.gz | more

9. Архивированные файлы можно сжимать, в результате получаются файлы, имеющие **расширение .tar.gz.**

$ gzip имя_каталога/имя_архива.tar

$ ls - посмотреть расширение вновь созданного файла.

10. Сначала выполните сжатие, затем архивирование:

$ tar czf archfZ.tar file2

По опции **z** сначала выполняется сжатие, затем архивирование.

Просмотрите результат: **$ ls**

11. Сжатие файлов по другому алгоритму сжатия выполняется командой **bzip2**, восстановление - командой **bunzip2,** расширения файлов архивов при этом **bz2**.

Перенесите файл **filel** в каталог **myx/katal,** выполните сжатие, если команда есть в системе, просмотрите файлы, выполните восстановление файла: **$ bzip2 myx/katal/filel**

$ ls -l

$ bunzip2 myx/katal/filel.bz2

12. Посмотрите в справочнике man информацию о командах сжатия **zip и compress** и на своих примерах проверьте их работу. Форматы их файлов **.zip** и **.Z** соответственно. Новые файлы добавить в архив. Опция **r** команды **tar** добавляет файлы в архив. Просмотреть созданные структуры, используя команду **ls** с соответствующими опциями.

Группировка команд

Команды можно группировать, объединяя их логическими действиями и устанавливая определенные режимы. Для этого используются следующие средства:

знак **;** - определение последовательности выполнения команд;

знак **&** - асинхронное (фоновое) выполнение предшествующей команды;

знак **&&** - выполнение последующей команды при условии нормального завершения предыдущей, иначе игнорировать;

знак || - выполнение последующей команды при ненормальном завершении предыдущей, иначе игнорировать последующую команду.

Чтобы выполнить команды в фоновом режиме (в асинхронном режиме) после команды ставится знак "амперсанд", при выполнении команды на экран выводится номер процесса, соответствующий выполняемой команде, и система, запустив этот фоновый процесс, вновь выходит на диалог с пользователем.

Пример 1. Найти в системе файл с именем **file2.** Команду поиска **find** использовать в фоновом режиме, поиск вести, начиная от корневого каталога /; затем выполнить команду определения путевого каталога **pwd**, определяющего, где находится пользователь, в обычном режиме.

$ find / -name file2 -print &	ввод команды find
818	на терминал выведен номер (PID) фонового процесса
$ pwd	ввод команды pwd
/mnt/floppy/laba2	результат работы pwd
$	возвращение в shell
/home/student12/lab/file2	результат работы find

Для группировки команд также могут использоваться фигурные, обозначаемые знаком {}, и круглые, обозначаемые знаком (), скобки. Рассмотрим примеры, сочетающие различные способы группировки. Пусть k1, k2 и k3 - некоторые команды.

Пример 2. Пусть введена групповая команда:

$ k1 && k2; k3

тогда команда **k2** будет выполнена только при успешном завершении команды **k1**; после любого из результатов выполнения команды **k2** будет выполнена команда **k3**.

Пример 3. Здесь обе команды **k2** и **k3** будут выполнены только при успешном завершении команды **k1**.

$ k1 && {k2; k3}

Пример 4. В фоновом режиме (**&**)будет выполняться последовательность команд **k1** и **k2**: **$ {k1; k2} &**

Если в сети выполняются команды нескольких пользователей, работающих с одними и теми же ресурсами, например, обрабатывается или создается файл с одним именем, то в системе продолжит существование тот вариант файла, который записан в систему последним. Это типичная ошибка пользователей компьютеров, которые редактируют один файл параллельно с нескольких экранов. Круглые скобки, кроме выполнения функции группировки, выполняют функцию вызова нового экземпляра КИ.

Пример 5. Пусть пользователь находится в начальном каталоге **/mnt/floppy/laba2**. Последовательность команд: **$ cd ..; ls; ls**

выведет на терминал два экземпляра содержимого каталога **/mnt/floppy**, а последовательность тех же команд, но с группировкой круглыми скобками: **$ (cd ..; ls) ls**, тогда вывод на терминал будет следующий:

/mnt/floppy

/mnt/floppy/laba2

При входе в скобки вызывается новый экземпляр КИ, осуществляющий переход. При выходе из круглых скобок происходит возврат в старый КИ и в старый каталог.

Переменные командного интерпретатора

В командном интерпретаторе можно определять переменные. Они определяются на сеанс работы. Переменная описывается одним словом, которое может состоять из букв, цифр и символов, не зарезервированных командным интерпретатором для внутреннего использования.

Задание 3. Выполните следующие упражнения, назначив свои имена переменных, каталогов и файлов.

1. Переменной **var1** присваивается значение **english** так:

$ var1=english

Обращение к переменной осуществляется через $ перед переменной, например, чтобы вывести значение этой переменной на экран, следует выполнить: **$ echo $var1**

2. Присвойте переменной значение одного из Ваших каталогов, например: **$ var2=home/user/student/myx** и скопируйте из текущего каталога файл **spisok2** в описанный в последней переменной: **$ cp spisok2 $var2**

3. Посмотрите, какие переменные описаны в этом сеансе работы:

$ set

4. Уберите установку второй переменной и проверьте, какие переменные остались**: $ unset var2**, **$ set**.

Выведется сообщение о наличии одной переменной **var1**.

5. Посмотрите содержимое Вашего рабочего каталога: **$ ls -alf**

Если переменная является частью какого-либо имени, то используются фигурные скобки.

Пример. Имя файла, используемого для загрузки модулей ядра: **/etc/rc.d/rc.modules.** Запишите в переменную имя каталога этого файла: **a=/etc/rc.d/**, тогда следующие команды равноценны:

$ cat /etc/rc.d/rc.modules,

$ cat ${a}rc.modules,

cat выдаст на экран содержимое одного и того же файла.

Командный язык командных интерпретаторов

Командный язык командных интерпретаторов часто называют **shell**, что в переводе означает: "раковина", "скорлупа". **Shell** - язык программирования очень высокого уровня. На этом языке пользователь осуществляет **управление** компьютером. Язык удобен для выполнения задач системного администрирования и сетевых задач.

После входа в режим терминала пользователь начинает взаимодействие с командной оболочкой. Признаком того, что оболочка готова к приему команд, служит выдаваемое на терминал приглашение к работе. Обычно это **$** - знак "доллар" для обычного пользователя или # - знак "диез" для привилегированного пользователя. Основным элементом языка **shell** является команда. **Shell** стандартизован в стандарте мобильных систем **POSIX**. Его разновидности: язык **cshell**, **kshell**, **bashell** и другие. Каждый пользователь может создать свой командный язык.

Пользователь может одновременно работать на одном экземпляре операционной системы с разными командными языками. Первый **shell** вызывается автоматически при входе в систему и выдает на экран приглашение к работе. После этого можно вызывать на выполнение любые команды, в том числе и снова сам **shell** (команда **sh** - вызов интерпретатора **shell**), который создаст новую оболочку внутри прежней.

Сценарии командного интерпретатора и создание собственных команд

Несколько команд командного интерпретатора можно выполнить заданием одной команды, являющейся сценарием. Для этого следует записать несколько команд в один текстовый файл последовательно, таким образом, что в каждой строке будет расположена очередная команда, запомнить файл. Получен сценарий командного интерпретатора.

Чтобы выполнить сценарий, следует задать команду:

$ sh имя_файла_сценария

Пример 1. Создать файл, включающий следующие команды:

```
mkdir katalsh
cp spisok1 ~/katalsh
cd  katalsh
ls ..
more spisok1
```

Выполнить сценарий: **$ sh szenf1**, запись может быть и такой:

$ sh < szenf1

Команды файла выполнятся последовательно, в порядке их записи в файл. На экране появится список файлов текущего каталога, включая **katalsh**, потом первая страница содержимого файла **spisok1**.

Для файла сценария можно установить право на выполнение, указав полномочия в относительной форме:
$ chmod u+x szenf1, или указав полномочия в абсолютной форме:
$ chmod 755 szenf1
тогда файл сценария можно выполнить так: **$ szenf1**.

При наборе команды **sh** вызывается **shell**. Файл можно выполнить и в текущем экземпляре **shell**. Для этого существует специфическая команда **.** (знак "точка"): **$. szf2**

Задание 4. Составьте свой файл сценария из известных Вам команд и выполните его описанными способами (без установки прав на выполнение и с установкой полномочий).

Переменные сценария

В сценариях можно устанавливать переменные, как это выполнялось это выше в командном интерпретаторе, можно использовать встроенные переменные, можно передавать параметры в сценарии.

Встроенные переменные описываются следующим образом: **$1, $2, $3, $4, …, $9**. Они являются позиционными: значения встроенных переменных передаются в сценарий в порядке их нумерации. Если используются значения переменных, которые назначили сами, то они передаются в сценарий в том порядке, в котором были назначены. Во избежание неоднозначности оба типа переменных в одном сценарии не рекомендуется использовать.

Пример. Создать файл сценария **szf2** для просмотра на терминале файлов с указанным расширением, которое может быть различным при каждом новом запуске сценария. Текст файла следующий: **ls *.$1**

Найти файлы с раширением **txt**. Установить значение переменной **$1** в данном случае **txt**. Выполнить файл сценария с передачей в него параметра **txt**: **$ sh szf2 txt**
здесь значение переменной указано после имени файла через пробел.

Создать файл сценария с текстом, включающим две переменные, для вывода списка файлов с указанным расширением и из любого указанного каталога: **ls $1/*.$2**

Теперь **$1** - имя каталога, **$2** - расширение. Выполнить файл:

$ sh szf2 usr doc

Задание 5. Создать сценарий поиска файлов с заданным расширением, задаваемым в командной строке.

Предложить хотя бы один способ ввода переменных сценария, если требуется найти файлы без расширения.

Найти в корневом каталоге точечные файлы (их имя начинается со знака "точка"), если известно, что второй символ имен таких файлов – строчный английский символ.

Комментарии и командные оболочки

Комментарии в файлах сценариев начинаются с символа "диез" -знак "решетка" (#). Поэтому нельзя начинать командные файлы с символа #., если в системе установлена оболочка **C-Shell** (**csh**), символ # будет интерпретирован как выполняемый в **csh**, в результате будет активизирован интерпретатор **csh**. Можно начать командный **sh**-файл с пустой строки или пустого оператора "двоеточие" (**:**), что часто можно встретить в сценариях серверов.

Поскольку **ОС UNIX** и **ОС Linux** - системы многопользовательские, то в них можно работать параллельно на нескольких терминалах, даже если они находятся на одном компьютере, (переход с терминал на терминал осуществляется нажатием ALT/функциональная клавиша), имея на каждом экране нового или одного и того же пользователя со своей командной оболочкой. Можно в графическом режиме открыть большое число окон, а в каждом окне может быть свой пользователь со своей командной оболочкой.

Иногда необходимо, чтобы все фоновые процессы завершились, прежде чем будет выполняться какой-то сценарий. Для этого служит специальная команда **wait [PID]**. Эта команда ждет завершения указанного идентификатором фонового процесса. Если команда не имеет параметра, то она ждет завершения всех фоновых процессов, дочерних для данного **sh**. Фоновые процессы сложно уничтожить. Для уничтожения фонового процесса нужно знать его номер. При запуске фонового процесса на экран выдается число, соответствующее номеру (идентификатору **PID**) этого процесса. Номер также можно узнать с помощью команды **ps** с опциями **aux**, позволяющими получить перечень всех процессов: **$ ps -aux**

В выведенной на экран таблице будут находиться: перечень идентификаторов процессов (PID), имена пользователей, текущее время, затраченное процессами, и т.д. В выведенной таблице можно найти номера процессов, подлежащих уничтожению, например, это 649 и 844.

Тогда командой, прекращающей выполнение процессов:

$ kill -9 649 844

можно уничтожить эти процессы. Важно, что при уничтожении процессов пользователь должен иметь то же имя пользователя, под которым была задана команда создания процессов, или иметь имя привилегированного пользователя.

Команда test ([ ])

Команда **test** проверяет выполнение некоторого условия и часто используется при построении сценариев в языке **shell**. С использованием этой встроенной в КИ команды, позволяющей сравнивать две или несколько величин, формируются операторы выбора и цикла. Существует два возможных формата команды:

test условие

или другая форма записи

[условие]

Если использовать второй вариант записи, то вместо того, чтобы писать перед условием слово **test**, следует заключать условие в скобки. Shell будет распознавать эту команду по открывающей скобке **[**, как слову, соответствующему команде **test**. Между скобками и содержащимся в них условием обязательно должны быть пробелы. Пробелы также должны быть между значениями и символом сравнения или операции, но при присвоении значений переменным пробелы не ставятся.

Условия

Команда **test** использует условия различных типов.

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".

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.

Условия проверки строк:

str1 = str2 - строки str1 и str2 совпадают;
str1 != str2 - строки str1 и str2 не совпадают;
-n str1 - строка str1 существует (непустая);
-z str1 строка str1 не существует (пустая).

Логические операции:

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

Пример 1. Условия проверки файлов. Вводить с клавиатуры командные строки, в первом случае будет получено подтверждение (код завершения 0), а во втором - опровержение (код завершения 1). **file1** - имя существующего файла. Проверить работу команды **test**, используя файлы рабочего каталога.

[-f file1] ; echo $?

будет выведено на терминал:

0

[-d file1] ; echo $?

будет выведено на терминал:

1

Пример 2. Сравнение переменных КИ. Далее следует набранная в командной строке команда и результат вывода на экран:

```
x="text"; export x; [ "text" = "$x" ]; echo $?
0
x=abc ; export x ; [ abc = "$x" ] ; echo $?
0
```

При выполнении команда **echo** выдает на терминал все написанное правее ее. Команда **export** делает переменную **x** доступной для других сценариев.

Пример 3. Команда **test** дает значение истина (код завершения 0) в случае, если в скобках стоит непустое слово.

```
[ true ] ; echo $?
0
[ ] ; echo $?
1
```

Пример 4. Существуют два стандартных значения условия, которые могут использоваться вместо условия. Это логические значения "истина - "**true** и "ложь" - **false**. Скобки для этого не нужны.

```
true ; echo $?
0
false ; echo $?
1
```

Пример 5. Условия сравнения целых чисел. В данном случае команда **test** воспринимает строки символов как целые числа, поскольку в условии стоит -eq. Команда и результат ее выполнения:

```
x=abc ; export x ; [ abc -eq "$x" ] ; echo $?
"[": integer expression expected before -eq
```

Во всех остальных случаях нулевому значению соответствует пустая строка. Если надо обнулить переменную, например **x**, это достигается присваиванием **x=0**. Команда и результат ее выполнения:

```
x=321 ; export x ; [ 321 -eq "$x" ] ; echo $?
0
x=3.21 ; export x ; [ 3.21 -eq "$x" ] ; echo $?
"[": integer expression expected before -eq
x=321 ; export x ; [ 123 -lt "$x" ] ; echo $?
0
```

Пример 6. Логические условия реализуются с помощью символов, используемых в языке C++ и других языках программирования высокого уровня. Команда и результат ее выполнения:

```
[ ! text ] ; echo $?
1
x=priem; export x; [ "$x" -f  file1 -o priem ] ; echo $?
0
 x=""; export x; [ "$x" -a -f  file1 -o ! privet ] ; echo $?
1
```

Встроенные переменные сценариев

Для передачи параметров в сценарии используются позиционные переменные (см. стр.38), которые вводятся в командной строке после имени команды со всеми ее опциями и параметрами через пробел: **$1**,**$2**,...,**$n** - позиционные параметры, переданные сценарию.

В сценариях КИ **bash** используются встроенные переменные, представляющие специальные символы-шаблоны, подстановку которых при встрече в исполняемом сценарии производит **bash**. Встроенными переменными сценариев являются следующие:

$0 - название сценария (имя файла сценария);
$# - количество позиционных параметров, переданных сценарию;
$? - код возврата последнего процесса;
$$ - номер (PID) текущего процесса;
$! - PID последнего запущенного в фоновом режиме процесса (асинхронного);
$@ - список позиционных параметров, переданных сценарию;
$* - все позиционные параметры, слитые в единую строку и записываемые через разделитель полей, по умолчанию - пробел (**$1 $2 $3** … **$n**).

Условный оператор if

В общем случае оператор **if** имеет следующую структуру:

if условие **then** список
[elif условие **then** список **]**

[else список **]**
fi

Здесь **elif** сокращенный вариант от **else if**, он может быть использован наряду с полным, допускается вложение произвольного числа операторов **if**, а также и других операторов).

Конструкции

[elif условие **then** список команд **]**
[else список команд **]**

не являются обязательными (здесь для указания на необязательности конструкций использованы квадратные скобки - не путать с квадратными скобками команды **test**). Минимальная структура этого оператора следующая:

if условие
then список команд
fi

Если выполнено условие (получен код завершения 0), то выполняется список команд, иначе он пропускается. Структура обязательно завершается служебным словом **fi**. Количество **fi** всегда должно соответствовать количеству **if**.

Пример 1. Пусть написан файл-сценарий **fs1**:

```
 if [ $1 -gt $2 ]
then pwd
else echo $0 : Hello IVAN!
 fi
```

Тогда выполнение сценария и результат:

$ fs1 12 11 даст
/home/student/IVAN

а выполнение **$ fs1 12 13**

даст результат:

fs1: Hello IVAN!

Пример 2. Выдача командами различных кодов завершения. Пусть сценарий **fs2** будет следующий:

```
if a=`expr "$1" -le "$2"`
then echo a=$a code=$?
else echo a=$a code=$?
fi
```

тогда выполнение **fs2** с параметрами **fd fd** даст **a=2 code=0**,
а выполнение **fs2** с параметрами **fd fb** даст **a=0 code=1**.

Пример 3. Использование нескольких вложений условного оператора **if**.

```
echo -n "Какую оценку получил на экзамене?: "
read z
if [ $z = 5 ]
        then echo Молодец !
elif [ $z = 4 ]
        then echo Все равно молодец !
elif [ $z = 3 ]
        then echo Все равно
elif [ $z = 2 ]
        then echo Плохо
else echo !????
fi
```

Оператор выбора case

Оператор выбора **case** имеет следующую структуру:

```
case строка in
    шаблон) список команд;;
    шаблон) список команд;;
    .
    .
    *) ...
esac
```

Здесь **case in** и **esac** - служебные слова. Строка (это может быть и один символ) сравнивается с шаблоном. Затем выполняется список команд выбранной строки. Служебное слово **esac** необходимо для завершения структуры.

Пример 1. Пример уже рассматривался при изучении структуры **if,** но его нагляднее можно реализовать с помощью структуры **case**.

```
echo -n "Какую оценку получил на экзамене?: "
```

```
read z
case $z in
    5) echo Молодец ! ;;
    4) echo Все равно молодец ! ;;
    3) echo Все равно;;
    2) echo Плохо ;;
    *) echo !???? ;;
esac
```

В конце строк выбора пишется "**;;**". Для каждого варианта может быть выполнено несколько команд. Если эти команды будут записаны в одну строку, то символ "**;**" будет использоваться как разделитель команд. Обычно последняя строка выбора имеет шаблон "*****", что в структуре **case** означает "любое значение". Эта строка выбирается, если не произошло совпадение значения переменной (здесь **$z**) ни с одним из ранее записанных шаблонов, ограниченных скобкой "**)**". Значения просматриваются в порядке записи.

Пример 2. Реализация меню с помощью команды **case**. В общем случае на месте отдельных команд могут быть группы команд.

```
echo Назовите файл, а затем (через пробел) наберите цифру,
echo соответствующую требуемой обработке:
echo 1 - отсортировать
echo 2 - выдать на экран
echo 3 - определить число строк
read x y # x - имя файла, y - что сделать
case $y in
    1) sort -r < $x ;;
    2) cat < $x ;;
    3) wc -l < $x ;;
    *) echo "unknow command ! " ;;
esac
```

Пример 3. Добавление информации со стандартного входа к файлу, указанному первым параметром (если параметр один), либо (если два параметра) из файла, указанного в качестве первого параметра в файл, указанный вторым параметром. Используется стандартная переменная **$#**, указывающая число параметров при вводе сценария и перенаправление с добавлением в файл, обозначаемое знаком ">>".

```
case $# in
     1) cat >> $1  ;;
     2) cat >> $2 < $1  ;;
     *) echo "Формат: case4 [откуда] куда" ;;
esac
```

Лабораторное задание и порядок выполнения работы

Изучить теоретический материал, выполнить рекомендуемые задания с использованием своих файлов и шаблонов.

Выполнить примеры по архивированию и сжатию, составлению сценариев, используя свои переменные, данные и наборы команд.

Составить и выполнить сценарии с использованием встроенных переменных команды *test*, операторов *if* и *case* изменять.

Законспектировать материал по новым командам, входящим в выполненные примеры и задания. Оформить отчет и защитить работу.

Контрольное задание

1. В каталоге **var** или любом доступном для чтения каталоге найти все файлы, создать их список и поместить в свой каталог в новый файл.
2. Найти подкаталоги выбранного каталога, создать их список и поместить его в свой каталог в новый файл. Файлы заархивировать и сжать одним из известных способов.
3. (**Дополнительное задание**) Создать текстовый файл **aaa.txt** с датами в разных форматах, составить сценарий записывающий в файл **bbb.txt** какой день недели соответствует первой дате из файла **aaa.txt** соответствующей формату DD.MM.YYYY. (Для этого можно использовать команду **date**)

Требования к отчету

Отчет должен содержать:

1. Описания нескольких разнотипных сценариев;
2. краткие сведения о работе;
3. материал по новым командам, входящим в выполненные примеры и задания;
4. описание последовательности выполнения основных команд по поиску в файлах по образцу и команд создания архивов с Вашими именами файлов и каталогов выполнить подробно.