# Лабораторная работа №2 «Командные файлы на BASH»

## Shell

Командный интерпретатор в среде UNIX выполняет две основные функции:

- представляет интерактивный интерфейс с пользователем, т.е. выдает приглашение, и обрабатывает вводимые пользователем команды;
- обрабатывает и исполняет текстовые файлы, содержащие команды интерпретатора (командные файлы);

В последнем случае, операционная система позволяет рассматривать командные файлы как разновидность исполняемых файлов. Соответственно различают два режима работы интерпретатора: интерактивный и командный.

Существует несколько типов оболочек в мире UNIX. Две главные - это 'Bourne shell" и "C shell'.

**Bourne shell** (или просто **shell**) использует командный синтаксис, похожий на первоначально для UNIX. В большинстве UNIX-систем Bourne shell имеет имя /bin/sh (где sh сокращение от "shell"). **C shell** использует иной синтаксис, чем-то напоминающий синтаксис языка программирования Си. В большинстве UNIX-систем он имеет имя /bin/csh.

В Linux есть несколько вариаций этих оболочек. Две наиболее часто используемые, это Новый **Bourne shell** (Bourne Again Shell) или "**Bash**" (/bin/bash) и **Tcsh** (/bin/tcsh).

**Bash** - это развитие прежнего shell с добавлением многих полезных возможностей, частично содержащихся в **C shell.**

Поскольку Bash можно рассматривать как надмножество синтаксиса прежнего shell, любая программа, написанная на sh shell должна работать и в Bash. Tcsh является расширенной версией C shell. При входе в систему пользователю загружается командный интерпретатор по умолчанию.

Информация о том, какой интерпретатор использовать для конкретного пользователя находится в файле /etc/passwd.

Возможно, вам захочется выполнить сценарий, написанный для одного из shell Linux, в то время как вы работаете в другом. Предположим, вы работаете в TCSH-shell и хотите выполнить написанный в BASH сценарий, содержащий команды этого (второго) shell. Сначала нужно с помощью команды sh перейти в BASH-shell, выполнить сценарий, а затем вернуться в TCSH. Эту процедуру можно автоматизировать, поставив первыми в сценарии символы #! и указав после них путевое имя программы нужного shell в вашей системе. Shell всегда изучает первые символы сценария и на их основании делает вывод о том, к какому типу shell этот сценарий относится - BASH, PDKSH или TCSH. Если первый символ - пробел, это сценарий BASH-shell или PDKSH-shell. Если первый символ - знак #, это сценарий TCSH-shell. Если первые символы - #!, то shell читает указанное за ними имя программы. После символов #! всегда должно следовать путевое имя программы нужного shell, по которому можно идентифицировать его тип. Если вы запускаете сценарий из shell, отличного от того, который указан в первой строке запускаемого сценария, то будет вызван shell, указанный в первой строке, и в нем выполнится ваш сценарий. В такой ситуации одного пробела или знака 41 для указания того, что это сценарий BASH или TCSH, бывает недостаточно. Такая идентификация работает только в собственных сценариях этих shell.

Чтобы обозначить сценарий другого shell, необходимо поставить символы #! и путевое имя. Например, если поставить в начало первой строки сценария hello комбинацию символов #!/bin/sh, то этот сценарий можно будет выполнять непосредственно из TCSH-shell. Сначала сценарий осуществит переход в BASH, выполнит его команды, а затем вернется в TCSH (или в тот shell, из которого он выполнялся).

В следующем примере сценарий hello содержит команду #!/bin/sh. Пользователь выполняет сценарий, находясь в TCSH-shell.

## Командные файлы

Командный файл в Unix представляет собой обычный текстовый файл, содержащий набор команд Unix и команд Shell. Для того чтобы командный интерпретатор воспринимал этот текстовый файл, как командный необходимо установить атрибут на исполнение.

```
$ echo " ps -af " > commandfile
$ chmod +x commandfile
$ ./commandfile
```

В представленном примере команда `echo “ ps -af ” > commandfile` создаст файл с одной строкой “ ps -af “, команда `chmod +x commandfile` установит атрибут на исполнение для этого файла, команда `./commandfile` осуществит запуск этого файла.

**Переменные shell**

Имя shell-переменной - это начинающаяся с буквы последовательность букв, цифр и подчеркиваний. Значение shell-переменной - строка символов.

Например: Var = “ String ” или Var = String

Команда `echo $Var` выведет на экран содержимое переменной Var т.е. строку ‘String’ , на то что мы выводим содержимое переменной указывает символ `"$"`. Так команда `echo Var` выведет на экран просто строку ‘Var’ .

Еще один вариант присвоения значения переменной Var = 'набор команд Unix'. Обратные кавычки говорят о том, что сначала должна быть выполнена заключенная в них команда), а результат ее выполнения, вместо выдачи на стандартный выход, приписывается в качестве значения переменной.

```
CurrentDate = 'date'
```

Переменной CurrentDate будет присвоен результат выполнения команды date. Можно присвоить значение переменной и с помощью команды "read", которая обеспечивает прием значения переменной с (клавиатуры) дисплея в диалоговом режиме.

```
echo "Введите число"
read X1
echo "вы ввели -" $X1
```

Несмотря на то, что shell-переменные в общем случае воспринимаются как строки, т. е. "35" - это не число, а строка из двух символов "3" и "5", в ряде случаев они могут интерпретироваться иначе, например, как целые числа.

Разнообразные возможности имеет команда "expr".

```
x=7
y=2
rez=expr $x + $y'
echo результат=$rez    ---  выдаст на экран результат=9
```

**Параметры командного файла**

В командный файл могут быть переданы параметры. В shell используются позиционные параметры (т.е. существенна очередность их следования). В командном файле соответствующие параметрам переменные (аналогично shell-переменным) начинаются с символа "$", а далее следует одна из цифр от 0 до 9: При обращении к параметрам перед цифрой ставится символ доллара "$" (как и при обращении к переменным):

$0 соответствует имени данного командного файла;

$1 первый по порядку параметр;

$2 второй параметр и т.д.

Поскольку число переменных, в которые могут передаваться параметры, ограничено одной цифрой, т.е. 9-ю ("0", как уже отмечалось имеет особый смысл), то для передачи большего числа параметров используется специальная команда "shift".

Команда "set" устанавливает значения параметров. Это бывает очень удобно. Например, команда "date" выдает на экран текущую дату, скажем, "Mon May 01 12:15:10 2002", состоящую из пяти слов, тогда set date echo $1 $3 $5 выдаст на экран Mon 01 2002

**Программные структуры**

Как во всяком процедурном языке программирования в языке shell есть операторы. Ряд операторов позволяет управлять последовательностью выполнения команд. В таких операторах

часто необходима проверка условия, которая и определяет направление продолжения вычислений.

**Команда test**

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

```
test условие
```

или

```
[ условие ]
```

В shell используются условия различных "типов".

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:

`strl = str2`
строки "strl" и "str2" совпадают;

`strl != str2`
строки "strl" и "str2" не совпадают;

`-n strl`
строка "strl" существует (непустая);

`-z strl`
строка "strl" не существует (пустая).

Условия сравнения целых чисел:

`x -eq y`
"x" равно "y",

`x -ne y`
"x" неравно "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".

То есть в данном случае команда "test" воспринимает строки символов как целые (!) числа. Поэтому во всех остальных случаях "нулевому" значению соответствует пустая строка. В данном же случае, если надо обнулить переменную, скажем, "x", то это достигается присваиванием "x=0".

Сложные условия реализуются с помощью типовых логических операций:

! (not) инвертирует значение кода завершения.

-o (or) соответствует логическому "ИЛИ".

-a (and) соответствует логическому "И"

**Управляющие структуры**

С помощью управляющих структур пользователь может осуществлять контроль над выполнением Linux-команд в программе. Управляющие структуры позволяют повторять команды и выбирать для выполнения команды, необходимые в конкретной ситуации. Управляющая структура состоит из двух компонентов: операции проверки и команд. Если проверка считается успешной, то выполняются команды. Таким образом, с помощью управляющих структур можно принимать решения о том, какие команды следует выполнять.

Существует два вида управляющих структур: циклы и условия. Цикл используется для повторения команд, тогда как условие обеспечивает выполнение команды при соблюдении некоторых критериев. В BASH-shell используются три вида циклических управляющих структур (while, for и for-in) и две условные управляющие структуры (if и case).

Управляющие структуры while и if - это структуры общего назначения. Они используются, например, для выполнения итераций и принятия решений на основании различных проверок. Управляющие структуры case и for - более специализированные. Структура case представляет собой модифицированную форму условия if и часто используется для построения меню. Структура for - это цикл ограниченного типа. Здесь обрабатывается список значений, и на каждой итерации цикла переменной присваивается новое значение.

В управляющих структурах if и while проверка построена на выполнении Linux- команды. Все команды ОС Linux после выполнения выдают код завершения. Если команда выполнена нормально, выдается код 0. Если по какой-либо причине команда не выполняется, то выдается положительное число, обозначающее тип отказа. Управляющие структуры if и while проверяют код завершения Linux-команды. Если код - 0, выполняются действия одного типа, если нет - другого.

**Дополнительная информация**

Во-первых, обязательно обратитесь к **man bash**.

Во-вторых, краткая памятка по командам, которые вам могут понадобиться:

`pwd`
: Вывести текущую директорию.

`hostname`
: Вывести или изменить сетевое имя машины.

`whoami`
: Ввести имя под которым я зарегистрирован.

`date`
: Вывести или изменить дату и время. Например, чтобы установить дату и время равную 2000-12-31 23:57, следует выполнить команду: date 123123572000

`time`
: Получить информацию о времени, нужного для выполнения процесса + еще кое-какую информацию. Не путайте эту команду с date. Например: Я могу определить как много времени требуется для вывода списка файлов в директории, набрав последовательность: time ls

`who`
: Определить кто из пользователей работает на машине.

`rwho -a`
: Определение всех пользователей, подключившихся к вашей сети. Для выполнения этой команды требуется, чтобы был запущен процесс rwho. Если такого нет - запустите setup. "setup" под суперпользователем.

`finger [имя_пользователя]`
: Системная информация о зарегистрированном пользователе. Попробуйте: finger root

`uptime`
: Количество времени прошедшего с последней перезагрузки.

`ps -a`
: Список текущих процессов.

`top`
: Интерактивный список текущих процессов отсортированных по использованию cpu.

`uname`
: Вывести системную информацию.

`free`
: Вывести информацию по памяти.

`df -h`
: (=место на диске) Вывести информацию о свободном и используемом месте на дисках (в читабельном виде).

`du / -bh | more`
: (=кто сколько занял) Вывод детальной информации о размере файлов по директориям начиная с корневой (в читабельном виде).

`cat /proc/cpuinfo`
: Системная информация о процессоре. Заметьте, что файла в /proc директории - не настоящие файлы. Они используются для получения информации, известной системе.

`cat /proc/interrupts`
: Используемые прерывания.

`cat /proc/version`
: Версия ядра Linux и другая информация

`cat /proc/filesystems`
: Вывести используемые в данный момент типы файловых систем.

`cat /etc/printcap`
: Вывести настройки принтера.

`lsmod`
: (как root) Вывести информацию о загруженных в данный момент модулях ядра.

`set | more`
: Вывести текущие значения переменных окружения.

`echo $PATH`
: Вывести значение переменной окружения "PATH" Эта команда может использоваться для вывода значений других переменных окружения. Воспользуйтесь командой set, для получения полного списка.

`grep ...`
: Поиск вхождения регулярного выражения в строки заданного файла (потока).

**Задания для выполнения**

Написать скрипт согласно индивидуальному заданию. Варианты индивидуальных заданий

1. Написать командный файл, реализующий меню из трех пунктов: 1-ый пункт - ввести пользователя и вывести на экран все процессы, запущенные данным пользователем; 2-ой пункт - показать всех пользователей, в настоящий момент, находящихся в системе; 3-ий пункт - завершение.
2. Написать командный файл, реализующий меню из трех пунктов: 1-ый пункт - вывести всех пользователей, в настоящее время, работающих в системе; 2-ой пункт - послать сообщение пользователю, имя пользователя, терминал и сообщение вводятся с клавиатуры; 3-ий пункт - завершение.
3. Написать командный файл, реализующий меню из трех пунктов: 1-ый пункт - показать все процессы пользователя, запустившего данный командный файл; 2-ой пункт - послать сигнал завершения процессу текущего пользователя(ввести PID процесса); 3-ий пункт - завершение.
4. Написать командный файл, посылающий сигнал завершения процессам текущего пользователя. Символьная маска имени процесса вводится с клавиатуры.
5. Написать командный файл подсчитывающий количество определенных процессов пользователя (Ввести имя пользователя и название процесса)
6. Реализовать Меню из двух пунктов: 1-ый пункт - определить количество запущенных данным пользователем процессов bash (предусмотреть ввод имени пользователя); 2-ой пункт - завершить все процессы bash данного пользователя.
7. Реализовать Меню из трех пунктов: 1-ый пункт поиск файла в каталоге <Имя файла> и <Имя каталога> вводятся пользователем; 2-ой пункт - копирование одного файла в другой каталог - <Имя файла> и <Имя каталога> вводятся; 3-ий пункт - завершение командного файла.
8. Написать командный файл, который в цикле по нажатию клавиши выводит информацию о системе, активных пользователях в системе, а для введенного имени пользователя выводит список активных процессов данного пользователя.
9. Реализовать командный файл, который при старте выводит информацию о системе, информацию о пользователе, запустившем данный командный файл, далее в цикле выводит список активных пользователей в системе - запрашивает имя пользователя и выводят список всех процессов bash запущенных данным пользователем.
10. Реализовать командный файл, позволяющий в цикле посылать всем активным пользователям сообщение - сообщение вводится с клавиатуры. Командный файл при старте выводит имя компьютера, имя запустившего командный файл пользователя, тип операционной системы, IP-адрес машины.
11. Реализовать командный файл, позволяющий в цикле посылать всем активным пользователям (исключая пользователя, запустившего данный командный файл) сообщение - сообщение вводится с клавиатуры. Командный файл при старте выводит имя компьютера, имя запустившего командный файл пользователя, тип операционной системы, список загруженных модулей.
12. Реализовать командный файл который при старте выводит информацию о системе, информацию о пользователе, запустившем данный командный файл, далее в цикле выводит список активных пользователей в системе - запрашивает имя пользователя и выводят список всех терминалов, на которых зарегистрирован этот пользователь.
13. Реализовать командный файл, который выводит: дату, информацию о системе, текущий каталог, текущего пользователя, настройки домашнего каталога текущего пользователя, далее в цикле выводит список активных пользователей - запрашивает имя пользователя и выводит информацию об активности данного пользователя.
14. Реализовать командный файл, который выводит: дату в формате день - месяц - год - время, информацию о системе в формате: имя компьютера : версия ОС : IP адрес : имя текущего пользователя : текущий каталог, Выводит настройки домашнего каталога текущего пользователя и основные переменные окружения. Далее в цикле выводит список активных пользователей - запрашивает имя пользователя и выводит информацию об активности введенного пользователя.

15. Реализовать командный файл, реализующий символьное меню( в цикле):
    1. Пункт: Вывод полной информации о файлах каталога: Ввести имя каталога для отображения
    2. Пункт: Изменить атрибуты файла: файл вводится с клавиатуры по запросу, атрибуты, которые требуются установить тоже вводятся. После изменения атрибутов вывести на экран расширенный список файлов для проверки установленных атрибутов
    ЗВыход: При старте командный файл выводит информацию об имени компьютера, IP - адреса, и список всех пользователей зарегистрированных в данный момент на компьютере.
16. Написать скрипт с использованием цикла for, выводящий на консоль размеры и права доступа для всех файлов в заданном каталоге и всех его подкаталогах (имя каталога задается пользователем в качестве первого аргумента командной строки).
17. Написать скрипт, находящий в заданном каталоге и всех его подкаталогах все файлы, владельцем которых является заданный пользователь. Имя владельца и каталог задаются пользователем в качестве первого и второго аргумента командной строки. Скрипт выводит результаты в файл (третий аргумент командной строки) в виде полный путь, имя файла, его размер. На консоль выводится общее число просмотренных файлов.
18. Написать скрипт поиска одинаковых по их содержимому файлов в двух каталогах, например, Dirl и Dir2. Пользователь задаёт имена Dirl и Dir2 в качестве первого и второго аргумента командной строки. В результате работы файлы, имеющиеся в Dirl, сравниваются с файлами в Dir2 по их содержимому. На экран выводятся число просмотренных файлов и результаты сравнения.