

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет Физико-Математических Наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЁТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 12

дисциплина: Операционные системы

Студент: Манаева Варвара Евгеньевна

Группа: НФИбд-01-20

Преподаватель: Кулябов Дмитрий Сергеевич

МОСКВА

2021 г.

Техническое оснащение:

- Персональный компьютер с операционной системой Windows 7;
 - Планшет для записи видеосопровождения и голосовых комментариев;
 - Виртуальная коробка VirtualBox, виртуальная машина с установленной на ней операционной системой CentOS;
 - Microsoft Teams, использующийся для записи скринкаста лабораторной работы;
 - Приложение MarkPad 2 для редактирования файлов формата *md*;
 - *pandoc* для конвертации файлов отчётов и презентаций.
-

Объект и предмет исследования: Логические управляющие конструкции и циклы в языке `bash`.

Цель [1]: Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Задачи:

- 1) Изучить теоретическую справку из текста лабораторной № 11 [2];
 - 2) Изучить справку, вызываемую командой `man`;
 - 3) Выполнить задание лабораторной (ака написать коды командных файлов).
-

Теоретические вводные данные: [2]

Командные файлы и функции. Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде:
`bash командный_файл [аргументы]`

Чтобы не вводить каждый раз последовательности символов `bash`, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды `chmod +x имя_файла`

Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как-будто он является выполняемой программой. Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществит её интерпретацию.

Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключённых в фигурные скобки. Удалить функцию можно с помощью команды `unset` с флагом `-f`. Команда `typeset` имеет четыре опции для работы с функциями:

- `-f` — перечисляет определённые на текущий момент функции;
- `-ft` — при последующем вызове функции иницирует её трассировку;
- `-fx` — экспортирует все перечисленные функции в любые дочерние программы оболочек;
- `-fu` — обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную `FPATH`, отыскивая файл с одноимёнными именами функций, загружает его и вызывает эти функции.

Передача параметров в командные файлы и специальные переменные. При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. С точки зрения командного файла эти параметры являются позиционными. Символ `$` является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов `$i`, где $0 < i < 10$, вместо неё будет осуществлена подстановка значения параметра с порядковым номером i , т.е. аргумента командного файла с порядковым номером i . Использование комбинации символов `$0` приводит к подстановке вместо неё имени данного командного файла. Рассмотрим это на примере.

Пусть к командному файлу `where` имеется доступ по выполнению и этот командный файл содержит следующий конвейер: `who | grep $1`.

Если Вы введёте с терминала команду `where andy`, то в случае, если пользователь, зарегистрированный в ОС UNIX под именем `andy`, в данный момент работает в ОС UNIX, то на терминал будет выведена строка, содержащая номер терминала, используемого указанным пользователем. Если же в данный момент этот пользователь не работает в ОС UNIX, то на терминал ничего не будет выведено.

Команда `grep` производит контекстный поиск в тексте, поступающем со стандартного ввода, для нахождения в этом тексте строк, содержащих последовательности символов, переданные ей в качестве аргументов, и выводит результаты своей работы на стандартный

вывод. В примере команда `grep` используется как фильтр, обеспечивающий ввод со стандартного ввода и вывод всех строк, содержащих последовательность символов `andy`, на стандартный вывод.

В ходе интерпретации файла командным процессором вместо комбинации символов `$1` осуществляется подстановка значения первого и единственного параметра `andy`. Если предположить, что пользователь, зарегистрированный в ОС UNIX под именем `andy`, в данный момент работает в ОС UNIX, то на терминале Вы увидите примерно следующее:

```
$ where andy
andy ttyG Jan 14 09:12
$
```

Определим функцию, которая изменяет каталог и печатает список файлов:

```
$ function clist {
> cd $1
> ls
> }
```

Теперь при вызове команды `clist` будет изменён каталог и выведено его содержимое.

Команда `shift` позволяет удалять первый параметр и сдвигает все остальные на места предыдущих.

При использовании в командном файле комбинации символов `$#` вместо неё будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполнение.

Вот ещё несколько специальных переменных, используемых в командных файлах:

- `$*` — отображается вся командная строка или параметры оболочки;
- `$?` — код завершения последней выполненной команды;
- `$$` — уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
- `$!` — номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
- `$-` — значение флагов командного процессора;
- `${#}` — *возвращает целое число — количество слов, которые были результатом \$;*
- `${#name}` — возвращает целое значение длины строки в переменной `name`;
- `${name[n]}` — обращение к `n`-му элементу массива;
- `${name[*]}` — перечисляет все элементы массива, разделённые пробелом;
- `${name[@]}` — то же самое, но позволяет учитывать символы пробелы в самих переменных;
- `${name:-value}` — если значение переменной `name` не определено, то оно будет заменено на указанное `value`;
- `${name:value}` — проверяется факт существования переменной;
- `${name=value}` — если `name` не определено, то ему присваивается значение `value`;
- `${name?value}` — останавливает выполнение, если имя переменной не определено, и выводит `value` как сообщение об ошибке;
- `${name+value}` — это выражение работает противоположно `${name-value}`. Если переменная определена, то подставляется `value`;
- `${name#pattern}` — представляет значение переменной `name` с удалённым самым коротким левым образцом (`pattern`);
- `${#name[*]}` и `${#name[@]}` — эти выражения возвращают количество элементов в массиве `name`.

Использование команды `getopts`. Весьма необходимой при программировании является команда `getopts`, которая осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий:
`getopts option-string variable [arg ...]`

Флаги — это опции командной строки, обычно помеченные знаком минус; Например, для команды `ls` флагом может являться `-F`. Иногда флаги имеют аргументы, связанные с ними. Программы интерпретируют флаги, соответствующим образом изменяя своё поведение.

Строка опций `option-string` — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за символом, обозначающим этот флаг, должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, то она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введенные данные с помощью оператора `case`.

Предположим, необходимо распознать командную строку следующего формата:

```
testprog -infilein.txt -outfileout.doc -L -t -r
```

Вот как выглядит использование оператора `getopts` в этом случае:

```
while getopts o:i:Ltr optletter
```

```
do case $optletter in
o)      oflag=1;          oval=$OPTARG;;
i)      iflag=1;          ival=$OPTARG;;
L)      lflag=1;          ;
t)      tflag=1;          ;
r)      rflag=1;          ;
*)      echo Illegal option $optletter
esac
done
```

Функция `getopts` включает две специальные переменные среды — `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента (будет равна `filein.txt` для опции `i` и `fileout.doc` для опции `o`. `OPTIND` является числовым индексом на упомянутый аргумент.

Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать её в функции не только для синтаксического анализа аргументов функций, но и для анализа введённых пользователем данных.

Управление последовательностью действий в командных файлах. Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости от результатов проверки некоторого условия. Для решения подобных задач язык программирования `bash` предоставляет возможность использовать такие управляющие конструкции, как `for`, `case`, `if` и `while`. С точки зрения командного процессора эти управляющие конструкции являются обычными командами и могут использоваться как при создании командных файлов, так и при работе в интерактивном режиме. Команды, реализующие подобные конструкции, по сути, являются операторами языка программирования `bash`. Поэтому при описании языка программирования `bash` термин оператор будет использоваться наравне с термином команда.

Команды ОС UNIX возвращают код завершения, значение которого может быть использовано для принятия решения о дальнейших действиях. Команда `test`, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения. Так например, команда

```
test -f file60
```

возвращает нулевой код завершения (истина), если файл `file` существует, и ненулевой код завершения (ложь) в противном случае:

- `test s` — истина, если аргумент `s` имеет значение истина;
- `test -f file` — истина, если файл `file` существует;
- `test -i file` — истина, если файл `file` доступен по чтению;
- `test -w file` — истина, если файл `file` доступен по записи;
- `test -e file` — истина, если файл `file` — исполняемая программа;
- `test -d file` — истина, если файл `file` является каталогом.

Оператор цикла `for`. В обобщённой форме оператор цикла `for` выглядит следующим образом:

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

При каждом следующем выполнении оператора цикла `for` переменная `имя` принимает следующее значение из списка значений, задаваемых списком `список-значений`. Вообще говоря, `список-значений` является необязательным. При его отсутствии оператор цикла `for` выполняется для всех позиционных параметров или, иначе говоря, аргументов. Таким образом, оператор `for i` эквивалентен оператору `for i in $*`. Выполнение оператора цикла `for` завершается, когда `списокзначений` будет исчерпан. Последовательность команд (операторов), задаваемая списком `список-команд`, состоит из одной или более команд оболочки, отделённых друг от друга с помощью символов `newline` или `;`. Рассмотрим примеры использования оператора цикла `for`.

В результате выполнения оператора

```
for A in alpha beta gamma
do echo A
done
```

на терминал будет выведено следующее:

```
alpha
beta
```

```
gamma
```

Предположим, что Вы хотите найти во всех файлах текущего каталога, содержащих исходные тексты программ, написанных на языке программирования Си, все вхождения функции с некоторым именем. Это можно сделать с помощью такой последовательности команд:

```
for i
do
grep $i *.c
done
```

Поместив эту последовательность команд в файл findref, после возможно, используя команду

```
findref 'hash(' 'insert(' 'symbol(',
```

вывести на терминал все строки из всех файлов текущего каталога, имена которых оканчиваются символами .c, содержащие ссылки на функции hash(), insert() и symbol(). Использование символов ' в вышеприведённом примере необходимо для снятия специального смысла с символа (.

Оператор выбора case. Оператор выбора case реализует возможность ветвления на произвольное число ветвей. Эта возможность обеспечивается в большинстве современных языков программирования, предполагающих использование структурного подхода.

В обобщённой форме оператор выбора case выглядит следующим образом:

```
case имя in
шаблон1) список-команд;;
шаблон2) список-команд;;
...
esac
```

Выполнение оператора выбора case сводится к тому, что выполняется последовательность команд (операторов), задаваемая списком список-команд, в строке, для которой значение переменной имя совпадает с шаблоном. Поскольку метасимвол /* соответствует произвольной, в том числе и пустой, последовательности символов, то его можно использовать в качестве шаблона в последней строке перед служебным словом esac. В этом случае реализуются все действия, которые необходимо произвести, если значение переменной имя не совпадает ни с одним из шаблонов, заданных в предшествующих строках.

Рассмотрим примеры использования оператора выбора case.

В результате выполнения оператора

```
for A in alpha beta gamma
do case $A in
alpha) B=a;;
beta) B=c;;
gamma) B=e
esac
echo $B
done
```

на терминал будет выведено следующее:

```
a c e
```

Условный оператор if. В обобщённой форме условный оператор if выглядит следующим образом:

```
if список-команд
then список-команд
{elif список-команд
then список-команд}
[else список-команд]
fi
```

Выполнение условного оператора if сводится к тому, что сначала выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово if. Затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), то будет выполнена последовательность команд

(операторов), которую задаёт список-команд в строке, содержащей служебное слово `then`. Фраза `elif` проверяется в том случае, когда предыдущая проверка была ложной. Строка, содержащая служебное слово `else`, является необязательной. Если она присутствует, то последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `else`, будет выполнена только при условии, что последняя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `if` или `elif`, возвращает ненулевой код завершения (ложь).

Рассмотрим следующий пример:

```
for A in *
do if test -d $A
then echo $A: is a directory
else echo -n $A: is a file and
if test -w $A
then echo writeable
elif test -r $A
then echo readable
else echo neither readable nor writeable
fi
fi
done
```

Первая строка в приведённом выше примере обеспечивает выполнение всех последующих действий в цикле для всех имён файлов из текущего каталога. При этом переменная `A` на каждом шаге последовательно принимает значения, равные именам этих файлов. Первая содержащая служебное слово `if` строка проверяет, является ли файл, имя которого представляет собой текущее значение переменной `A`, каталогом. Если этот файл является каталогом, то на стандартный вывод выводятся имя этого файла и сообщение о том, что файл с указанным именем является каталогом. Эти действия в приведённом выше примере обеспечиваются в результате выполнения третьей строки.

Оставшиеся строки выполняются только в том случае, если проверка того, является ли файл, имя которого представляет собой текущее значение переменной `A`, каталогом, даёт отрицательный ответ. Это означает, что файл, имя которого представляет собой текущее значение переменной `A`, является обычным файлом. Если этот файл является обычным файлом, то на стандартный вывод выводятся имя этого файла и сообщение о том, что файл с указанным именем является обычным файлом.

Эти действия в приведённом выше примере обеспечиваются в результате выполнения четвёртой строки. Особенностью использования команды `echo` в этой строке является использование флага `-n`, благодаря чему выводимая командой `echo` строка не будет дополнена символом `newline` (перевод строки), что позволяет впоследствии дополнить эту строку, как это, например, показано в приведённом выше примере.

Вторая строка, содержащая служебное слово `if`, проверяет, доступен ли по записи файл, имя которого представляет собой текущее значение переменной `A`. Если этот файл доступен по записи, то строка дополняется соответствующим сообщением. Если же этот файл недоступен по записи, то проверяется, доступен ли этот файл по чтению. Эти действия в приведённом выше примере обеспечиваются в результате выполнения седьмой строки. Если этот файл доступен по чтению, то строка дополняется соответствующим сообщением. Если же этот файл недоступен ни по записи, ни по чтению, то строка также дополняется соответствующим сообщением.

Эти действия в приведённом выше примере обеспечиваются в результате выполнения девятой строки.

Операторы цикла `while` и `until`. В обобщённой форме оператор цикла `while` выглядит следующим образом:

```
while список-команд
do список-команд
done
```

Выполнение оператора цикла `while` сводится к тому, что сначала выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `while`, а затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `do`, после чего осуществляется безусловный переход на начало оператора цикла `while`. Выход из цикла будет осуществлён тогда, когда последняя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `while`, возвратит ненулевой код завершения (ложь).

Приведённый ниже фрагмент командного файла иллюстрирует использование оператора цикла `while`. В нем реализуется ожидание события, состоящего в удалении файла с определённым именем, и только после наступления этого события производятся дальнейшие действия:

```
while test -f lockfile
do sleep 30
echo waiting for semaphore
done
:create the semaphore file
echo > lockfile
:further commands and after them delete the semaphore file
rm lockfile
```

Командный файл, продемонстрированный в приведённом примере, по сути, является простейшей реализацией механизма синхронизации взаимодействующих процессов на основе семафоров.

При замене в операторе цикла `while` служебного слова `while` на `until` условие, при выполнении которого осуществляется выход из цикла, меняется на противоположное. В остальном оператор цикла `while` и оператор цикла `until` идентичны. В обобщённой форме оператор цикла `until` выглядит следующим образом:

```
until список-команд
do список-команд
done
```

Следующие две команды ОС UNIX используются только совместно с управляющими конструкциями языка программирования `bash`: это команда `true`, которая всегда возвращает код завершения, равный нулю (т.е. истина), и команда `false`, которая всегда возвращает код завершения, не равный нулю (т.е. ложь). Ниже приведены два примера, иллюстрирующие бесконечные циклы, которые будут выполняться до тех пор, пока ЭВМ не сломается или не будет выключена (ну, по крайней мере, до тех пор, пока Вы не нажмёте клавишу, соответствующую специальному символу `INTERRUPT`):

```
while true
do echo hello andy
done64
```

И

```
until false
do echo hello mike
done
```

Прерывание циклов. Два несложных способа позволяют вам прерывать циклы в оболочке `bash`. Команда `break` завершает выполнение цикла, а команда `continue` завершает данную итерацию блока операторов.

Команда `break` полезна для завершения цикла `while` в ситуациях, когда условие перестаёт быть правильным. Пример бесконечного цикла `while` с прерыванием в момент, когда файл перестаёт существовать:

```
while true
do
if [! -f $file]
then
break
fi
sleep 10
done
```

Команда `continue` используется в ситуациях, когда больше нет необходимости выполнять блок операторов, но вы можете захотеть продолжить проверять данный блок на других условных выражениях. Пример предназначен для игнорирования файла `/dev/null` в произвольном списке:

```
while file=$filelist[$i]
(( $i < ${#filelist[*]} ))
do
if
[ "$file" == "dev/null" ]
then
continue
```

```
fi
action
done
```

Эта программа пропускает нужное значение, но продолжает тестировать остальные.

Этапы работы:

1) Используя команды `getopts` `grep`, написал командный файл, который анализирует командную строку с ключами (список ниже), а затем ищет в указанном файле нужные строки, определяемые ключом `-p`.

- `-iinputfile` — прочитать данные из указанного файла;
- `-ooutputfile` — вывести данные в указанный файл;
- `-ршаблон` — указать шаблон для поиска;
- `-C` — различать большие и малые буквы;
- `-n` — выдавать номера строк

2) Написала на языке Cpp программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку. Командный файл вызывает эту программу и, проанализировав с помощью команды `$?`, выдаёт сообщение о том, какое число было введено.

3) Написала командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до `i`. Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют).

4) Написала командный файл, который с помощью команды `tag` запаковывает в архив все файлы в указанной директории. Модифицировал его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовал команду `find`).

Выводы: я изучила больше скриптов и команд, позволяющих мне работать с командными файлами большей сложности.

Контрольные вопросы [1]:

1) Каково предназначение команды `getopts`?

Ответ: Команда `getopts` предназначена для разбора параметров сценариев. Она обрабатывает исключительно однобуквенные параметры как с аргументами, так и без них.

2) Какое отношение метасимволы имеют к генерации имён файлов?

Ответ: Метасимволы используются для генерации имен:

- `?` произвольный символ;
- `*` произвольная последовательность символов;
- `[...]` любой из символов, указанных в скобках перечислением и/или с указанием диапазона;

3) Какие операторы управления действиями вы знаете?

Ответ: Операторы: `;` ; `&&(и)` ; `||(или)`; `$.` - операторы управления действиями.

4) Какие операторы используются для прерывания цикла?

Ответ: Оператор `break` используется для прерывания цикла.

5) Для чего нужны команды `false` и `true`?

Ответ: Команда `true` всегда возвращает. Команда `false` всегда возвращает не ноль. Программа `true` – всегда завершается с кодом 0, `false` – всегда завершается с кодом 1.

6) Что означает строка `if test -f man$/$i.$s`, встреченная в командном файле?

Ответ: Эта строка означает условие существования файла `man$/$i.$s`

7) Объясните различия между конструкциями `while` и `until`.

Ответ: Цикл While выполняется до тех пор, пока указанное в нем условие истинно. Цикл Until выполняется до тех пор, пока указанное в нем условие ложно.

Библиография:

1. Текст лабораторной работы № 12;
2. Текст лабораторной работы № 11.