РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 8

дисциплина: Операционные системы

Студент: Бешкуров Михаил

Группа: НК-101

МОСКВА

2019 г.

1. Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

2. Описание процесса выполнения задания

1. Написал скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в моем домашнем каталоге. При этом файл должен архивироваться архиватором tar.

```
mihail@mihail-beshkurov ~/Загрузки $ touch script.sh
mihail@mihail-beshkurov ~/Загрузки $ mcedit script.sh

mihail@mihail-beshkurov ~/Загрузки $ chmod +x script.sh
mihail@mihail-beshkurov ~/Загрузки $ ./script.sh
mihail@mihail-beshkurov ~/Загрузки $ ls
angeom                  rsa
back.sh                 script.sh
crunch-3.6              SetupCrypTool_1_4_41_en.exe
dirsearch-master        stellar-base-0.1.5
game (fceux)            StephenKing.11.22.63.doc
lua-5.3.4               uml.png
nikto-2.1.5             xortool
PEiD-0.95-fileAnalysis  Титульный лист отчета по  лабораторной работе.doc
res.tar
mihail@mihail-beshkurov ~/Загрузки $
```

```
#!/bin/bash
backup="back.sh"
cp "$0" "$backup"
tar -czf res.tar $backup
~
```

2. Написал пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов

```
#!/bin/bash
echo "input"

head -1
~
```

```
mihail@mihail-beshkurov ~/Загрузки $  ./script.sh
input
1 2 3
1 2 3
mihail@mihail-beshkurov ~/Загрузки $  ./script.sh
input
1 2 3 4 5 6 7 8 9 10 11
1 2 3 4 5 6 7 8 9 10 11
```

3. Написал командный файл — аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.

```
#!/bin/bash
for A in *
do if test -d $A
    then echo $A: "Это каталог"
    else echo -n $A: "Этот файл"
        if test -w $A
        then echo " Доступен для записи"
            if test -r $A
            then echo " Доступен для чтения"
            else echo " Не доступен ни для чтения ни для записи"
            fi
    fi
    fi
done
~
```

```
mihail@mihail-beshkurov ~/Загрузки $  ./script.sh
angeom: Это каталог
back.sh: Этот файл Доступен для записи
 Доступен для чтения
crunch-3.6: Это каталог
dirsearch-master: Это каталог
```

4. Написал командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки (рис. 10, рис. 11).

```
Файл  Правка  Вид  Поиск  Терминал  Справка
#!/bin/bash
format=""
direct=""
echo "укажите формат"
read format;
echo "укажите директорию"
read direct;
find "$direct" -name "*.$format" -type f | wc -l
~
```

```
mihail@mihail-beshkurov ~/Загрузки $  vi script.sh
mihail@mihail-beshkurov ~/Загрузки $  ./script.sh
укажите формат
sh
укажите директорию
/home/mihail/Загрузки
1
mihail@mihail-beshkurov ~/Загрузки $ ls
angeom                  script.sh
crunch-3.6              SetupCrypTool_1_4_41_en.exe
dirsearch-master        stellar-base-0.1.5
game (fceux)            StephenKing.11.22.63.doc
lua-5.3.4               uml.png
nikto-2.1.5             xortool
PEiD-0.95-fileAnalysis  Титульный лист отчета по  лабораторной работе.doc
rsa
mihail@mihail-beshkurov ~/Загрузки $
```

3. Выводы

Изучил основы программирования в оболочке ОС UNIX/Linux. Научился писать небольшие командные файлы.

4. Контрольные вопросы

1. Командные процессоры или оболочки - это программы, позволяющие пользователю взаимодействовать с компьютером. Их можно рассматривать как настоящие интерпретируемые языки, которые воспринимают команды пользователя и обрабатывают их. Поэтому командные процессоры также называют интерпретаторами команд. На языках оболочек можно писать программы и выполнять их подобно любым другим программам.

UNIX обладает большим количеством оболочек. Наиболее популярными являются следующие четыре оболочки: –оболочка Борна (Bourne) - первоначальная командная оболочка UNIX: базовый, но полный набор функций; –C-оболочка - добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя C-подобный синтаксис команд, и сохраняет историю выполненных команд; –оболочка Корна - напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; –BASH - сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

2. POSIX (Portable Operating System Interface for Computer Environments)- интерфейс переносимой операционной системы для компьютерных сред. Представляет собой набор стандартов, подготовленных институтом инженеров по электронике и радиотехники (IEEE), который определяет различные аспекты построения операционной системы. POSIX включает такие темы, как программный интерфейс, безопасность, работа с сетями и графический интерфейс. POSIX-совместимые оболочки являются будущим поколением оболочек UNIX и других ОС. Windows NT рекламируется как система, удовлетворяющая POSIX-стандартам. POSIX-совместимые оболочки разработаны на базе оболочки Корна; фонд бесплатного программного обеспечения (Free Software Foundation) работает над тем, чтобы и оболочку BASH сделать POSIX-совместимой.

3. Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда mark=/usr/andy/bin присваивает значение строки символов /usr/andy/bin переменной mark типа строка символов. Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол $. Например, команда mv afile $mark переместит файл afile из текущего каталога в каталог с абсолютным полным именем /usr/andy/bin. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того, чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: ${имя переменной} например, использование команд b=/tmp/andy-ls -l myfile > ${b}ls приведет к переназначению стандартного

вывода команды ls с терминала на файл /tmp/andy-ls , а использование команды ls -l>$bls приведет к подстановке в командную строку значения переменной bls. Если переменной bls не было предварительно присвоено никакого значения, то ее значением является символ пробел. Оболочка bash позволяет создание массивов. Для создания массива используется команда set с флагом -A. За флагом следует имя переменной, а затем список значений, разделенных пробелом. Например,set -A states Delaware Michigan "New Jersey" Далее можно сделать добавление в массив, например, states[49]=Alaska. Индексация массивов начинается с нулевого элемента.

4. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение - это единичный терм (term), обычно целочисленный. Целые числа можно записывать как последовательность цифр или в любом базовом формате. Этот формат — radix#number, где radix (основание системы счисления) - любое число не более 26. Для большинства команд основания систем счисления это - 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток (%). Команда let берет два операнда и присваивает их переменной.

5. Арифметические операции

Оператор	Синтаксис	Результат
!	!exp	Если exp равно 0, то возвращает 1; иначе 0
!=	exp1 !=exp2	Если exp1 не равно exp2, то возвращает 1; иначе 0
%	exp1%exp2	Возвращает остаток от деления exp1 на exp2
%=	var=%exp	Присваивает остаток от деления var на exp переменной var
&	exp1&exp2	Возвращает побитовое AND выражений exp1 и exp2
&&	exp1&&exp2	Если и exp1 и exp2 не равны нулю, то возвращает 1; иначе 0
&=	var &= exp	Присваивает переменной var побитовое AND var и exp
*	exp1 * exp2	Умножает exp1 на exp2
*=	var *= exp	Умножает exp на значение переменной var и присваивает результат переменной var
+	exp1 + exp2	Складывает exp1 и exp2
+=	var += exp	Складывает exp со значением переменной var и результат присваивает переменной var
-	-exp	Операция отрицания exp (унарный минус)
-	exp1 - exp2	Вычитает exp2 из exp1
-=	var -= exp	Вычитает exp из значения переменной var и присваивает результат переменной var
/	exp / exp2	Делит exp1 на exp2
/=	var /= exp	Делит значение переменной var на exp и присваивает результат переменной var
<	exp1 < exp2	Если exp1 меньше, чем exp2, то возвращает 1, иначе возвращает 0
<<	exp1 << exp2	Сдвигает exp1 влево на exp2 бит
<<=	var <<= exp	Побитовый сдвиг влево значения переменной var на exp
<=	exp1 <= exp2	Если exp1 меньше или равно exp2, то возвращает 1; иначе возвращает 0
=	var = exp	Присваивает значение exp переменной var
==	exp1==exp2	Если exp1 равно exp2, то возвращает 1; иначе возвращает 0
>	exp1 > exp2	1, если exp1 больше, чем exp2; иначе 0
>=	exp1 >= exp2	1, если exp1 больше или равно exp2; иначе 0
>>	exp >> exp2	Сдвигает exp1 вправо на exp2 бит
>>=	var >>=exp	Побитовый сдвиг вправо значения переменной var на exp
^	exp1 ^ exp2	Исключающее OR выражений exp1 и exp2
^=	var ^= exp	Присваивает переменной var побитовое XOR var и exp
|	exp1 | exp2	Побитовое OR выражений exp1 и exp2
|=	var |= exp	Присваивает переменной var результат операции XOR var и exp
||	exp1 || exp2	1, если или exp1 или exp2 являются ненулевыми значениями, иначе 0

6. Для облегчения программирования можно записывать условия оболочки bash в двойные скобки — (( )).

7. Каждая переменная должна иметь уникальное имя. Программист сам решает, исходя из постановки задачи и ее алгоритма, какие имена давать переменным. Рекомендуется переменным давать смысловые, информационные имена.

Некоторые требования при задании имен переменным:

· имя должно начинаться с буквы;

· имя может содержать любые буквы и цифры;

· в имени не должно быть пробелов;

· имена нельзя называть ключевыми словами VisualBasic;

· в пределах области видимости имя должно быть неповторимым (уникальным).

Последнее требование будет ясно позже, после знакомства с областью видимости переменных. Примеры разрешенных и неразрешенных имен:

8. Такие символы, как ’ < > * ? | \ " & являются метасимволами и имеют для командного процессора специальный смысл.

9. Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа\, который, в свою очередь, является метасимволом. Для экранирования группы метасимволов, ее нужно заключить в одинарные кавычки. Строка, заключенная в двойные кавычки, экранирует все метасимволы, кроме $, ’ , \, ". Например,–echo *выведет на экран символ*,–echo ab’*\|*’cdвыдаст строку ab*\|*cd.

10. 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 ее интерпретацию.

11. Группу команд можно объединить в функцию. Для этого существует ключевое слово function, после которого следует имя функции и список команд, заключенных в фигурные скобки. Удалить функцию можно с помощью команды unset с флагом-f. Команда typeset имеет четыре опции для работы с функциями: -f — перечисляет определенные на текущий момент функции; –-ft— при последующем вызове функции инициирует ее 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 одноименными именами функций, загружает его и вызывает эти функции.

12. ls -lrt; Если есть d, то файл является каталогом

13. Используется команда set с флагом -A. За флагом следует имя переменной, а затем список значений, разделенных пробелом. Например, set -A states Delaware Michigan "New Jersey" Далее можно сделать добавление в массив, например, states[49]=Alaska . Индексация

массивов начинается с нулевого элемента. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set. Наиболее распространенным является сокращение, избавляющееся от слова let в программах оболочек. Если объявить переменные целыми значениями, любое присвоение автоматически трактуется как арифметическое. Используйте typeset -i для объявления и присвоения переменной, и при последующем использовании она становится целой. Или можете использовать ключевое слово integer (псевдоним для typeset -l) и объявлять переменные целыми. Таким образом, выражения типа x=y+z воспринимаются как арифметические. Группу команд можно объединить в функцию. Для этого существует ключевое слово function , после которого следует имя функции и список команд, заключенных в фигурные скобки. Удалить функцию можно с помощью команды unset с флагом -f . Команда typeset имеет четыре опции для работы с функциями: – -f —

перечисляет определенные на текущий момент функции; – -ft — при последующем вызове функции инициирует ее 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 одноименными именами функций, загружает его и вызывает эти функции. В переменные mon и day будут считаны соответствующие значения, введенные с клавиатуры, а переменная trash нужна для того, чтобы отобрать всю избыточно введенную информацию и игнорировать ее. Изъять переменную из программы можно с помощью команды unset.

14. 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 команд- ном файле комбинации символов $i, где $0 < i < 10$, вместо нее 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 нее 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 введете с терминала команду: where andy, то в случае, если пользователь, зарегистрированный в ОС UNIX под именем andy, в данный момент работает в ОС UNIX, 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 не будет выведено ничего. 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 этом 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 этого 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 каталог будет изменен каталог и выведено его содержимое.

15. Специальные переменные языке bash и их назначения:

– $* — 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-ному элементу массива;

– ${name[*]} — перечисляет все элементы массива, разделенные про- белом;

– ${name[@]} — то же самое, но позволяет учитывать символы про- белы 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 удаленным самым коротким левым образцом (pattern);

– ${#name[*]} и ${#name[@]} — эти выражения возвращают количество элементов в массиве name.

− $# вместо нее будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполнение.