

# Лабораторная работа №11

---

**По дисциплине Операционные системы**

Выполнил Гамаюнов Н.Е., студент ФФМиЕН РУДН, НПМбд-01-20, 1032201717

Преподаватель Кулябов Дмитрий Сергеевич

Москва, 2021 г.

# Цель работы

---

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

## Задания

---

Написать скрипт, который при запуске будет делать резервную копию самого себя, пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять, командный файл — аналог команды ls (без использования самой этой команды и команды dir), 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.

## Выполнение лабораторной работы

---

1. Создал каталог **backup**, файл **original** и изучил тап по архиватору **zip**, чтобы выполнить первое задание (рисунки 1-2)

```
[negamayunov@negamayunov ~]$ touch original.sh  
[negamayunov@negamayunov ~]$ vi original.sh  
[negamayunov@negamayunov ~]$ man zip
```

Рисунок 1.

```
[negamayunov@negamayunov ~]$ mkdir /backup
```

Рисунок 2.

2. Написал скрипт, который при запуске делает резервную копию самого себя в архив **bckp.zip** в каталоге **backup** (рисунки 3), проверил его работу (рисунки 4)

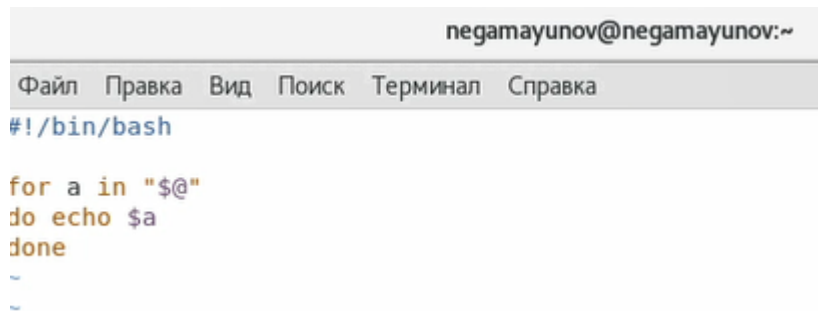
```
negamayunov@negamayunov:~  
Файл  Правка  Вид  Поиск  Терминал  Справка  
#!/bin/bash  
  
zip bckp.zip original.sh  
mv bckp.zip ~/backup  
  
~  
~  
~  
~  
~
```

Рисунок 3.

```
[negamayunov@negamayunov ~]$ ./original.sh
  adding: original.sh (deflated 18%)
[negamayunov@negamayunov ~]$ cd backup
[negamayunov@negamayunov backup]$ ls
bckp.zip
```

Рисунок 4.

3. Написал пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять (*рисунок 5*), проверил его работу (*рисунок 6*)



```
negamayunov@negamayunov:~
Файл  Правка  Вид  Поиск  Терминал  Справка
#!/bin/bash

for a in "$@"
do echo $a
done

~
~
```

Рисунок 5.

```
[negamayunov@negamayunov ~]$ ./2nd.sh 1 3 5
1
3
5
[negamayunov@negamayunov ~]$ ./2nd.sh 1 2 3 4 5 6 7 8 9 0 11 23
1
2
3
4
5
6
7
8
9
0
11
23
```

Рисунок 6.

4. Написал командный файл — аналог команды `ls` (без использования самой этой команды и команды `dir`) (*рисунок 7*), проверил его работу (*рисунок 8*)

```

negamayunov@negamayunov:~
Файл Правка Вид Поиск Терминал Справка
#!/bin/bash

for a in $1/*
do
echo $a

if test -f $a
then echo "Файл, который "
if test -w $a
then echo "доступен к записи и "
else echo "недоступен к записи и "
fi

if test -i $a
then echo "доступен к прочтению, и"
else echo "недоступен к прочтению, и"
fi

if test -x $a
then echo "доступен к записи"
else echo "недоступен к записи"
fi

```

Рисунок 7.

```

/home/negamayunov/australlia
Каталог
/home/negamayunov/backup
Каталог
/home/negamayunov/cat1
Каталог
/home/negamayunov/conf.txt
Файл, который
доступен к записи и
доступен к прочтению, и

```

Рисунок 8.

5. 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 (рисунок 9), проверил его работу (рисунок 10)

```

[negamayunov@negamayunov ~]$ ./4th.sh .sh ~
5

```

Рисунок 9.

```
[negamayunov@negamayunov ~]$ cat 4th.sh
#!/bin/bash

i=0
for a in $2
do
    for b in $2/*$1
    do
        if test -f "$b"
        then let i=i+1
        fi
    done
done

echo $i
```

Рисунок 10.

## Контрольные вопросы

---

1. Командный процессор (командная оболочка, интерпретатор команд shell) – это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:
  - оболочка Борна (Bourne shell или sh) – стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
  - С-оболочка (или csh) – надстройка на оболочкой Борна, использующая Сподобный синтаксис команд с возможностью сохранения истории выполнения команд;
  - оболочка Корна (или ksh) – напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна;
  - BASH – сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании Free Software Foundation).
2. POSIX (Portable Operating System Interface for Computer Environments) – набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linuxподобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.
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. Оболочка bash позволяет работать с массивами. Для создания массива используется команда set с флагом -A. За флагом следует имя переменной, а затем список

значений, разделённых пробелами. 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. Оболочка bash поддерживает встроенные арифметические функции. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение – это единичный терм (term), обычно целочисленный. Команда let берет два операнда и присваивает их переменной. Команда read позволяет читать значения переменных со стандартного ввода: «echo "Please enter Month and Day of Birth ?"» «read mon day trash» 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 её.
5. В языке программирования bash можно применять такие арифметические операции как сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток от деления (%).
6. В (( )) можно записывать условия оболочки bash, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.
7. Стандартные переменные:
  - PATH: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /. Если имя команды содержит хотя бы один символ /, то последовательность поиска, предписываемая значением переменной PATH, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневого или текущего каталога.
  - PS1 и PS2: эти переменные предназначены для отображения промптера командного процессора. PS1 – это промптер командного процессора, по умолчанию его значение равно символу \$ или #. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, то используется промптер PS2. Он по умолчанию имеет значение символа >.
  - HOME: имя домашнего каталога пользователя. Если команда cd вводится без аргументов, то происходит переход в каталог, указанный в этой переменной.
  - IFS: последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (new line).
  - MAIL: командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение You have mail (у Вас есть почта).
  - TERM: тип используемого терминала.
  - LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.
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

нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме \$, ' , , ". Например,

- `echo *` выведет на экран символ `*`;
- `echo ab`|`cd` выведет на экран 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 её интерпретацию.

11. 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`.

12. Чтобы выяснить, является ли файл каталогом или обычным файлом, необходимо воспользоваться командами «`test -f [путь до файла]`» (для проверки, является ли обычным файлом) и «`test -d [путь до файла]`» (для проверки, является ли каталогом).

13. Команду «`set`» можно использовать для вывода списка переменных окружения. В системах Ubuntu и Debian команда «`set`» также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду «`set | more`». Команда «`typeset`» предназначена для наложения ограничений на переменные. Команду «`unset`» следует использовать для удаления переменной из окружения командной оболочки.

14. 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.

15. Специальные переменные:

- `$*` – отображается вся командная строка или параметры оболочки;
- `$?` – 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`

*Источник всей информации, которой я пользовался для ответа на вопросы, - **Методические рекомендации к лабораторной работе №11***

## Выводы

---

В ходе выполнения лабораторной работы я изучил основы программирования в оболочке ОС UNIX/Linux и научился писать небольшие командные файлы.

## Библиография

---

- Кулябов Д. С. и др. Операционные системы. Методические рекомендации к лабораторной работе №11