

# **Лабораторная работа № 11.**

Кальсин Захар Алексеевич

# Содержание

1	Цель работы	5
---	-------------	---

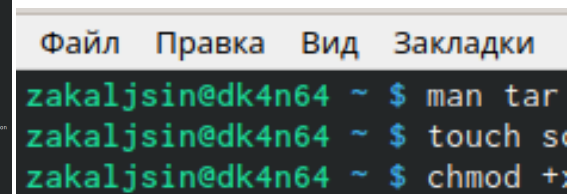
## List of Tables

# List of Figures

## 1 Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux, научиться писать небольшие командные файлы. # Выполнение лабораторной работы Ход работы:

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 домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации узнал, изучив справку.

[illegible]



misk

```
#!/bin/bash
backupname="ScriptBack.sh"
cp "$0" "$backup_name"
tar -cf laba.tar $backup_name
```



script.sh

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

```
#!/bin/bash
echo "Vvedite znachenie"
head -1
```

```
zaka1jsin@dk4n64
Vvedite znachenie
9 8 7 6 5
9 8 7 6 5
zaka1jsin@dk4n64
```

```
~:--- script2.sh All L1 (
```

аргументов.

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

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
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
```

--- script3.sh All L1 (Shell-script[bash])

ЭТОГО каталога.

```
zakaljsin@dk4n64 ~ $ ./script3.sh
abc1: is a file andwriteable
abcl: is a directory
australia: is a directory
conf.txt: is a file andwriteable
etc: is a directory
feathers: is a file andwriteable
file.txt: is a file andwriteable
GNUstep: is a directory
hello: is a file andwriteable
hello.asm: is a file andwriteable
lab02.asm: is a file andwriteable
lab03a: is a directory
lab03b: is a directory
lab05: is a directory
lab06: is a directory
lab06.o: is a file andwriteable
lab07: is a directory
lab10.sh: is a file andwriteable
lab10.sh~: is a file andwriteable
Lab0S: is a directory
letters: is a directory
logfile: is a file andwriteable
may: is a file andwriteable
memos: is a directory
misk: is a directory
monthly: is a directory
my_os: is a directory
```

4.Написал команд-



ный 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

```
file Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
format=""
direct=""
echo "point out format"
read format
echo "point out direct"
read direct
find "$direct" -name ".$format" -type f | wc -l
ls
```

--- script4.sh All L1 (Shell-script[bash])

виде аргумента командной строки.

```
zakaljsin@dk4n64 ~ $ ./script4.sh
point out format
.sh
point out direct
/home
0 0 0 -
wc: l: Нет такого файла или каталога
0 0 0 итого
abc1 conf.txt file.txt hello.asm lab03b
abc1 etc GNUstep lab02.asm lab05
australia feathers hello lab03a lab06
zakaljsin@dk4n64 ~ $
```

Вывод: изучил основы

программирования в оболочке ОС UNIX/Linux, научился писать небольшие командные файлы.

Ответы на 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;
- C-оболочка (или csh) — надстройка над оболочкой Борна, использующая подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — 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) — 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/Linux-подобных 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`.

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/andyls -l myfile > blssudoapt — getinstalltexlive — luatexls/tmp/andy — ls,ls — l >bls` приведёт к подстановке в командную строку значения переменной `bls`. Если переменной `bls` не было предварительно присвоено никакого значения, то её значением будет символ пробела.

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, 5, 6. 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. Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа `let sum=x+7`, и `let` будет искать переменную `x` и добавлять к ней 7.

Команда `let` также расширяет другие выражения `let`, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения.

Команда `let` не ограничена простыми арифметическими выражениями.

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, введённые с клавиатуры, а переменная `trash` нужна для того, чтобы отобразить всю избыточно введённую информацию и игнорировать её.

7. – `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, 9. Такие символы, как `' < > * ? | " &`, являются метасимволами и имеют для командного процессора специальный смысл. 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 \$, ' , , ”.

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, 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 является файл каталогом

13. Для создания массива используется команда set с флагом -A. За флагом следует

имя переменной, а затем список значений, разделённых пробелами. 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 инициирует 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. 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 командном 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. – \$\* — 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 опре-  
делено, 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 элемен-  
тов в массиве `name`.