

отчёта по лабораторной работе 11

Операционные системы

Алхатиб Осама

Содержание

1	Отчет по лабораторной работе №11	5
2	Тема:	6
2.1	Программирование в командном процессоре ОС UNIX. Командные файлы	6
2.2	Российский Университет Дружбы Народов	6
2.2.1	Факультет Физико-Математических и Естественных Наук .	6
2.2.2	Цель работы	6
2.2.3	Последовательность выполнения работы	7
2.2.4	Введение	7
2.2.5	Ход работы:	8
2.2.6	Вывод	12
2.3	Ответы на контрольные вопросы:	12

List of Tables

List of Figures

1 Отчет по лабораторной работе №11

2 Тема:

2.1 Программирование в командном процессоре ОС UNIX. Командные файлы

2.2 Российский Университет Дружбы Народов

2.2.1 Факультет Физико-Математических и Естественных Наук

Дисциплина: Операционные системы

Студент: Алхатиб Осама

Группа: НПИбд-02-20

2021г.

2.2.2 Цель работы

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

2.2.3 Последовательность выполнения работы

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.
 2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
 3. Написать 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.
-

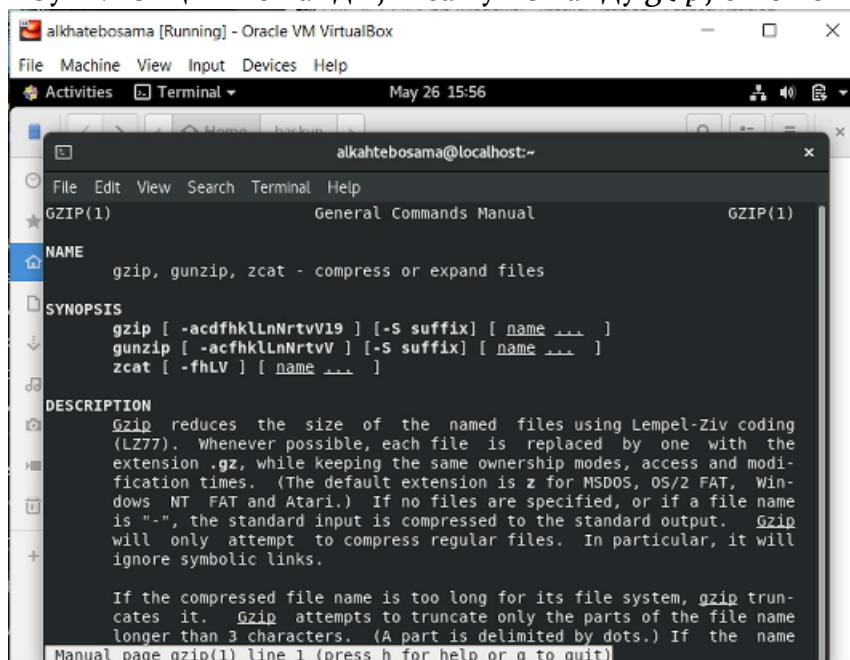
2.2.4 Введение

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).

2.2.5 Ход работы:

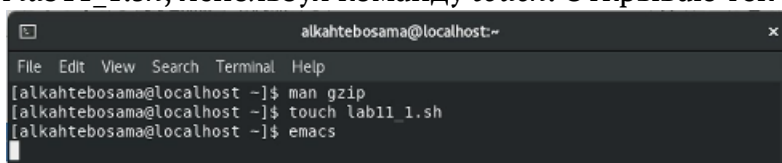
1. Изучил опции команды, и саму команду *gzip*, с помощью команды *man*.



```
alkhatebosama [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal May 26 15:56
alkhatebosama@localhost:~
File Edit View Search Terminal Help
GZIP(1) General Commands Manual GZIP(1)
NAME
gzip, gunzip, zcat - compress or expand files
SYNOPSIS
gzip [ -acdfhklLnNrtvV19 ] [-S suffix] [ name ... ]
gunzip [ -acdfhklLnNrtvV ] [-S suffix] [ name ... ]
zcat [ -fhLV ] [ name ... ]
DESCRIPTION
Gzip reduces the size of the named files using Lempel-Ziv coding
(LZ77). Whenever possible, each file is replaced by one with the
extension .gz, while keeping the same ownership modes, access and modi-
fication times. (The default extension is z for MSDOS, OS/2 FAT, Win-
dows NT FAT and Atari.) If no files are specified, or if a file name
is "-", the standard input is compressed to the standard output. Gzip
will only attempt to compress regular files. In particular, it will
ignore symbolic links.
If the compressed file name is too long for its file system, gzip trun-
cates it. Gzip attempts to truncate only the parts of the file name
longer than 3 characters. (A part is delimited by dots.) If the name
Manual page gzip(1) line 1 (press h for help or q to quit)
```

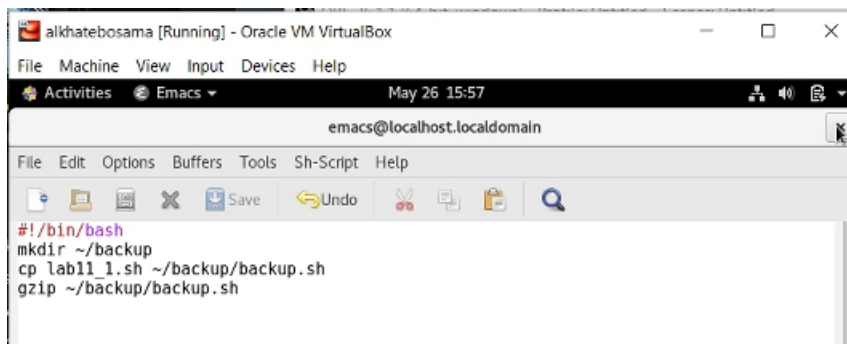
- Создал текстовый файл *lab11_1.sh*, используя команду *touch*. Открываю тек-

стовый редактор *emacs*.



```
alkhatebosama@localhost:~
File Edit View Search Terminal Help
[alkhatebosama@localhost ~]$ man gzip
[alkhatebosama@localhost ~]$ touch lab11_1.sh
[alkhatebosama@localhost ~]$ emacs
```

- Написал код, в котором я создаю папку *backup*, копировал текстовый файл *lab11_1.sh*, указав путь для сохранения файла. После архивировал данный файл через команду *gzip*.

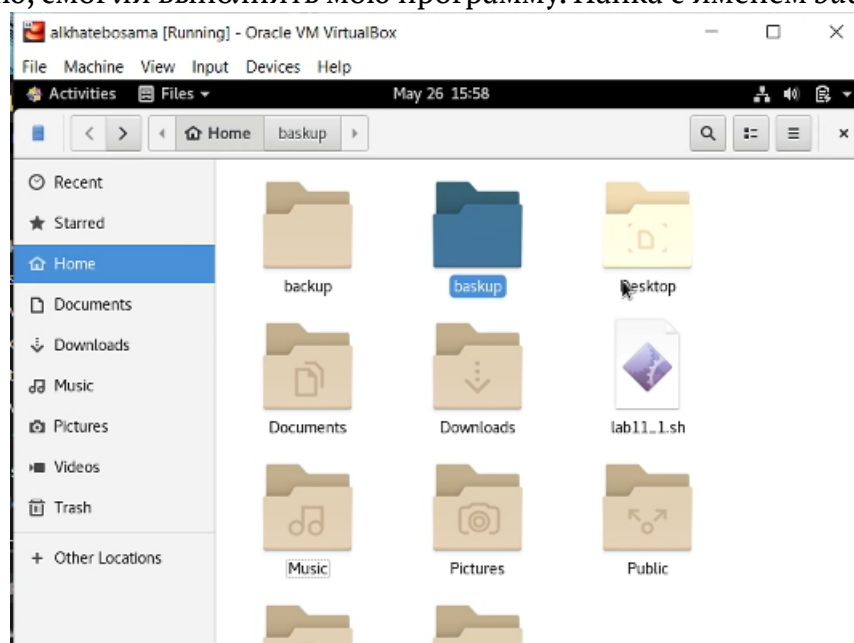


- Командой *chmod* я сделал так, чтобы файл был исполняемым в **Linux**.

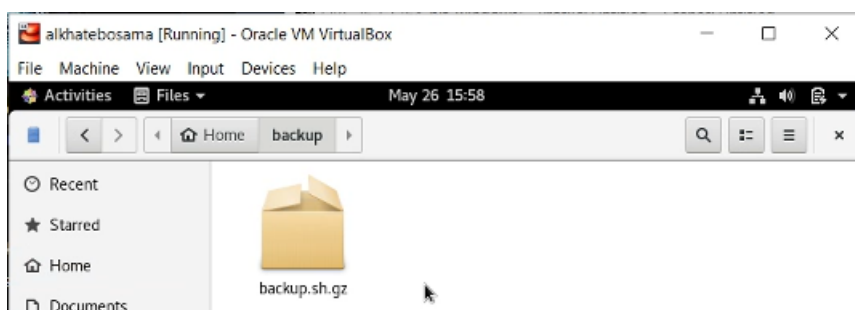
```
alkhatebosama@localhost: ~$ chmod +x lab11_1.sh
alkhatebosama@localhost: ~$ ./lab11_1.sh
```

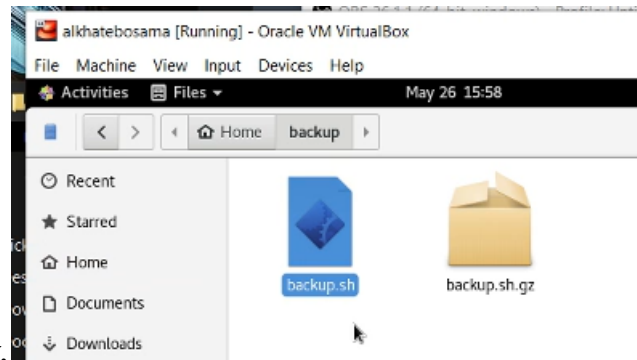
- Проверяю, смог ли выполнить мою программу. Папка с именем *backup* была

создана.



- Проверяю, смог ли выполнить мою программу.



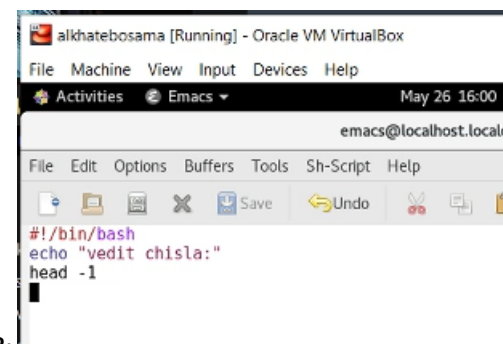


- Проверяю, смог ли выполнить мою программу.

2. Создал текстовый файл *lab11_2.sh* (для выполнения 2 пункта данной лабораторной работы), используя команду *touch*. Открываю текстовый редактор *emacs*.

```
[alkhatebosama@localhost ~]$ touch lab11_2.sh
[alkhatebosama@localhost ~]$ emacs
```

emacs.



- Написал программу, для вывода того, что я буду вводить.
- Командой *chmod* я сделал так, чтобы файл был исполняемым в **Linux**. Следующая строка для того, чтобы он выполнил нашу ранее написанную

```
[alkhatebosama@localhost ~]$ chmod +x lab11_2.sh
[alkhatebosama@localhost ~]$ ./lab11_2.sh
vedit chisla:
12 14 55 00 99 00
12 14 55 00 99 00
[alkhatebosama@localhost ~]$
```

программу.

3. Создал текстовый файл *lab11_3.sh* (для выполнения 2 пункта данной лабораторной работы), используя команду *touch*. Открываю текстовый редактор *emacs*.

```
[alkhatebosama@localhost ~]$ touch lab11_3.sh
[alkhatebosama@localhost ~]$ emacs
```

emacs.

- Написал командный файл — аналог команды *ls* (без использования самой

```

alkhaitebosama [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Emacs May 26 16:03
emacs@localhost.localdomain
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/bin/bash
for i in *
do if test -d $i
then echo $i: is a directory
else echo -n $i: is a file
if test -w $i
then echo available for writing
elif test -r $i
then echo readable
else echo neather for readable nor writeable
fi
done

```

этой команды и команды `dir`).

- Он выдавал информацию о нужном каталоге и выводил информацию о воз-

```

alkhaitebosama@localhost ~]$ chmod +x lab11_3.sh
alkhaitebosama@localhost ~]$ ./lab11_3.sh
backup: is a directory
baskup: is a directory
Desktop: is a directory
Documents: is a directory
Downloads: is a directory
lab11 0.sh-: is a fileavailable for writing
lab11 1.sh-: is a fileavailable for writing
lab11 1.sh-: is a fileavailable for writing
lab11 2.sh-: is a fileavailable for writing
lab11 2.sh-: is a fileavailable for writing
lab11 3.sh-: is a fileavailable for writing
lab11 3.sh-: is a fileavailable for writing
lab11 4.sh-: is a fileavailable for writing
lab11.sh-: is a fileavailable for writing
Music: is a directory
Pictures: is a directory
Public: is a directory
Templates: is a directory
Videos: is a directory
alkhaitebosama@localhost ~]$

```

можностях доступа к файлам этого каталога.

4. Создал текстовый файл `lab11_2.sh`(для выполнения 2 пункта данной лабораторной работы), используя команду `touch`. Открываю текстовый редактор `emacs`.

```

alkhaitebosama@localhost ~]$ touch lab11_4.sh
alkhaitebosama@localhost ~]$ emacs

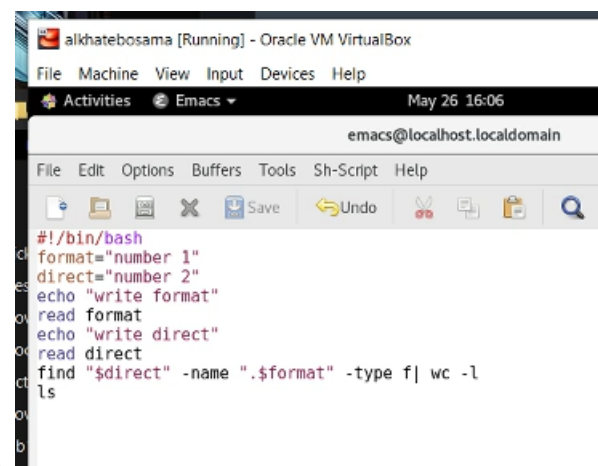
```

- Написал командный файл, который получает в качестве аргумента команд-

ной строки формат файла (.txt, .doc, .jpg, pdf и т.д.).

- Вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.

```
[alkahtebosama@localhost ~]$ emacs
[alkahtebosama@localhost ~]$ chmod +x lab11_4.sh
[alkahtebosama@localhost ~]$ ./lab11_4.sh
write format
.sh
write direct
home/alkahtebosama
find: 'home/alkahtebosama': No such file or directory
0
backup    Downloads  lab11_2.sh  lab11_4.sh  Pictures
baskup    lab11_0.sh-  lab11_2.sh-  lab11_4.sh-  Public
Desktop   lab11_1.sh  lab11_3.sh  lab11.sh-    Templates
Documents lab11_1.sh-  lab11_3.sh-  Music        Videos
[alkahtebosama@localhost ~]$
```



2.2.6 Вывод

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

2.3 Ответы на 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) — 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 ${mark}` переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: `${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 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 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 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.

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 $1 > ls > }`. Теперь при вызове команды `clist` каталог будет изменен каталог и выведено его содержимое. 15. – `$*` — 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.