

# **Отчёт по лабораторной работе №11**

**Программирование в командном процессоре ОС UNIX. Командные  
файлы**

Ломакина Анна Ивановна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание лабораторной работы</b>	<b>6</b>
2.1	Задание №1 . . . . .	6
2.2	Задание №2 . . . . .	6
2.3	Задание 3 . . . . .	6
2.4	Задание 4 . . . . .	7
<b>3</b>	<b>Выполнение работы</b>	<b>8</b>
3.1	Задание №1 . . . . .	8
3.2	Задание №2 . . . . .	9
3.3	Задание 3 . . . . .	10
3.4	Задание 4 . . . . .	11
3.5	Контрольные вопросы: . . . . .	12
<b>4</b>	<b>Вывод</b>	<b>20</b>

## **Список таблиц**

# Список иллюстраций

3.1	Задание №4 . . . . .	12
-----	----------------------	----

# 1 Цель работы

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

## **2 Задание лабораторной работы**

### **2.1 Задание №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.

### **2.2 Задание №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.

### **2.3 Задание 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.

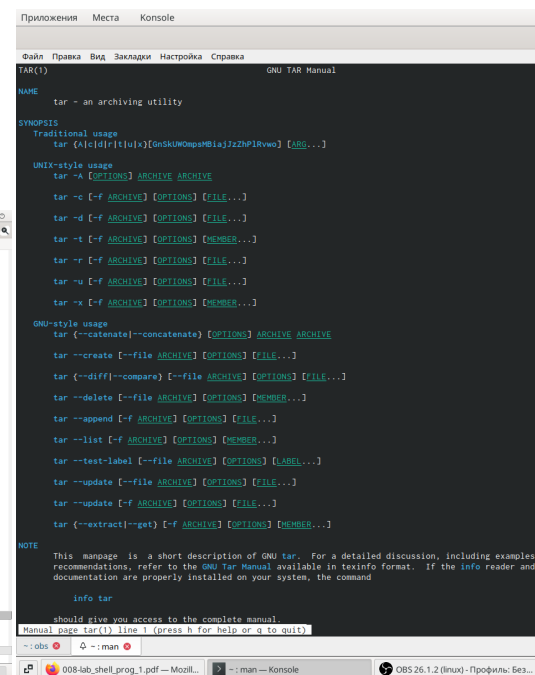
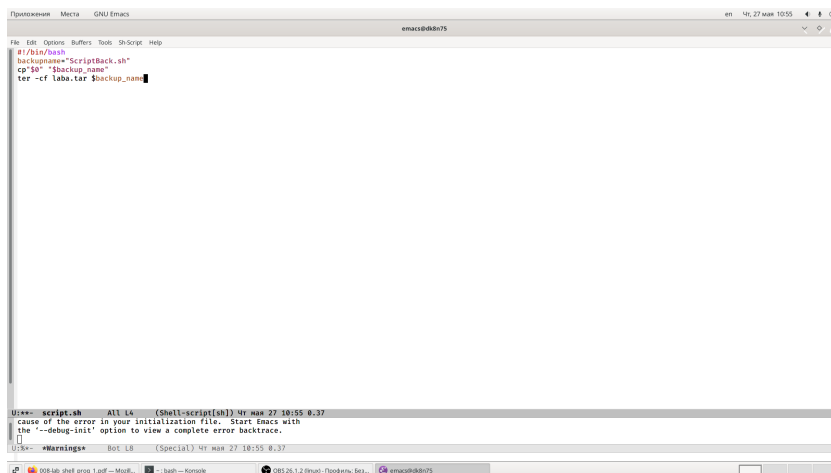
## 2.4 Задание 4

Написать 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.

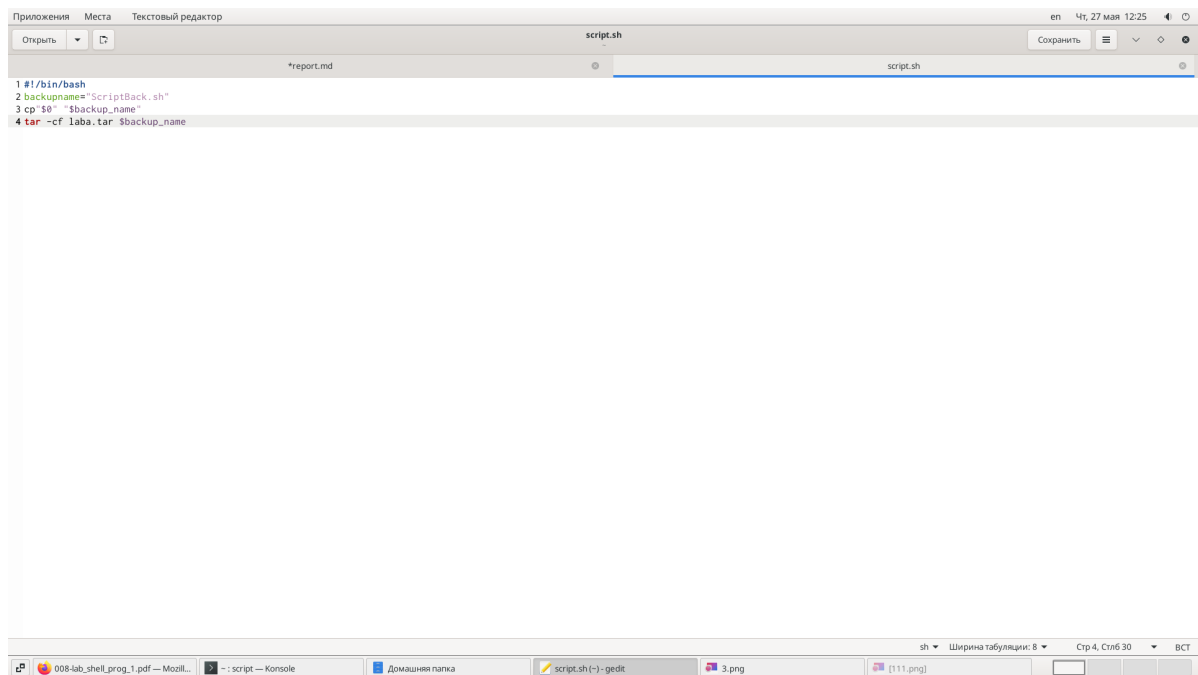
# 3 Выполнение работы

## 3.1 Задание №1

Я написала 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 узнала, изучив справку.

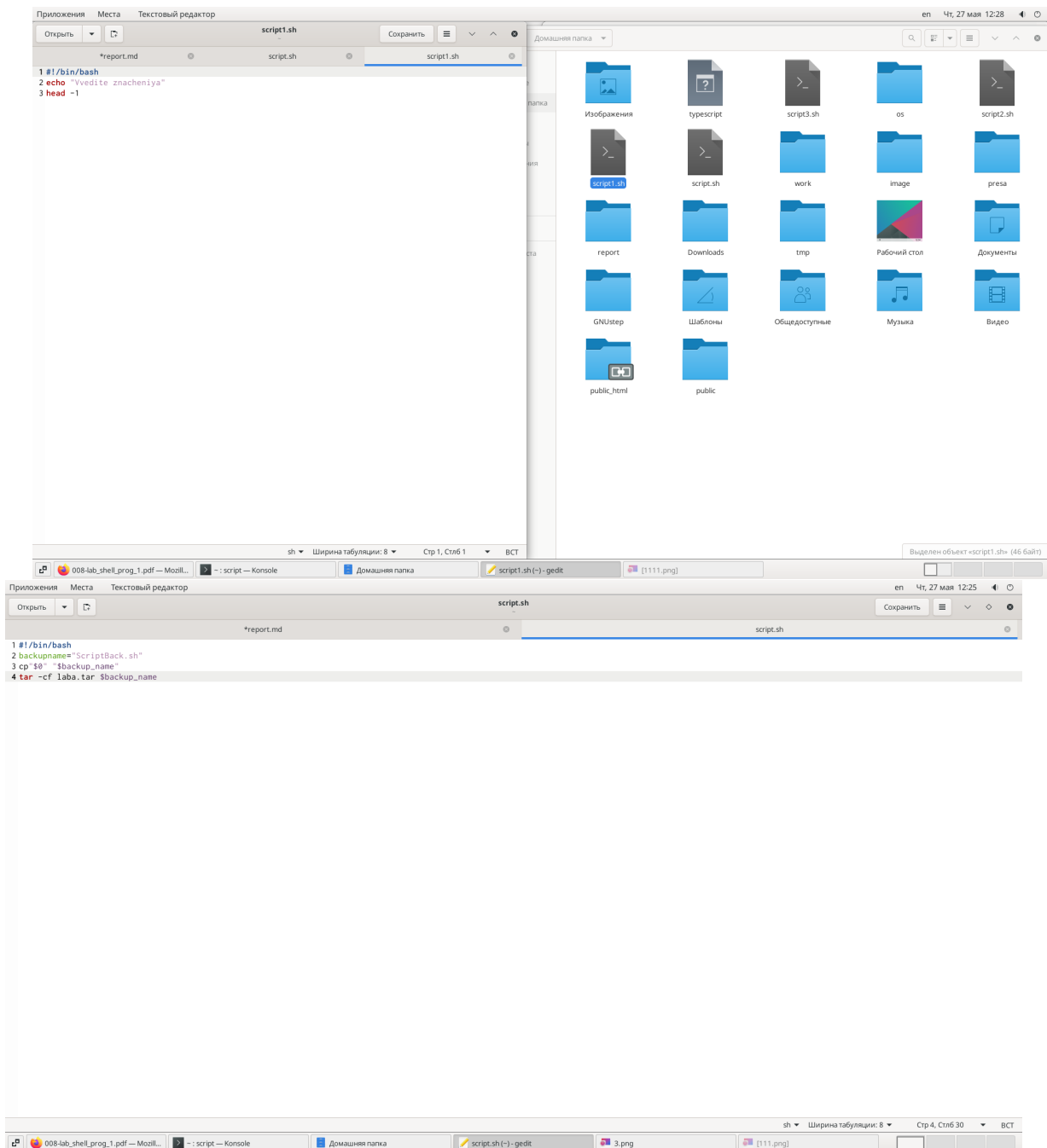






## 3.2 Задание №2

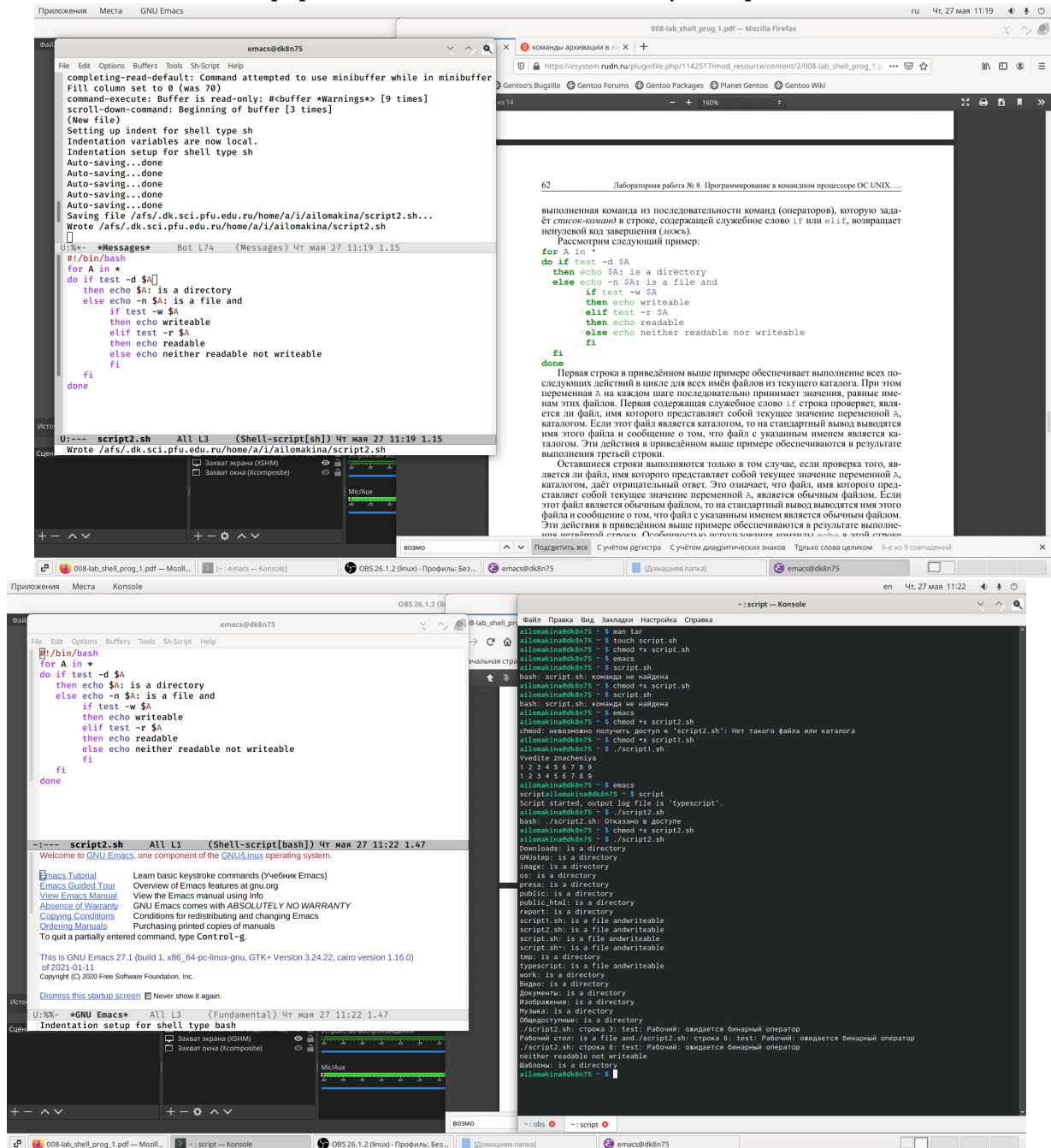
Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.



### 3.3 Задание 3

Написала командный файл — аналог команды `ls` (без использования самой этой команды и команды `dir`). Требуется, чтобы он выдавал информацию о нужномка-

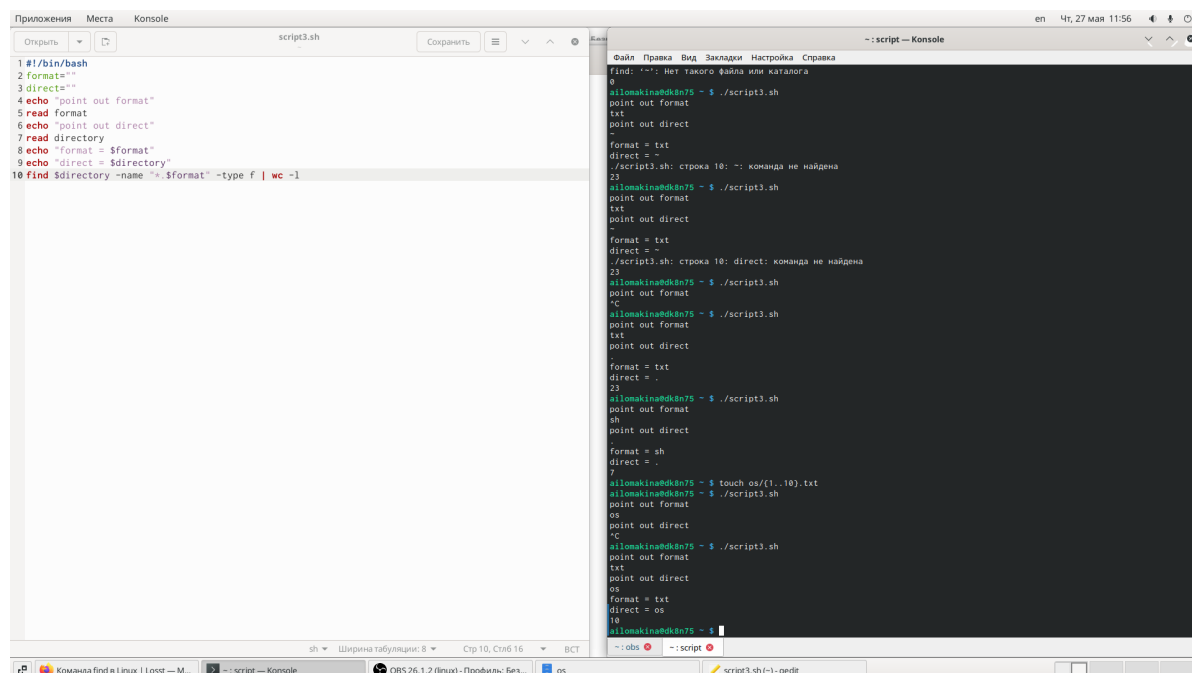
талог и выводил информацию о возможностях доступа к файлам этого каталога.



## 3.4 Задание 4

Написала 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.



```
1 #!/bin/bash
2 format=""
3 direct=""
4 echo "point out format"
5 read format
6 echo "point out direct"
7 read direct
8 echo "format = $format"
9 echo "direct = $direct"
10 find $direct -name "$format" -type f | wc -l
```

```
find: -: Нет такого файла или каталога
0
allomakina@dkn75 ~$ ./script3.sh
point out format
txt
point out direct
format = txt
direct = .
./script3.sh: строка 10: команда не найдена
23
allomakina@dkn75 ~$ ./script3.sh
point out format
txt
point out direct
format = txt
direct = .
./script3.sh: строка 10: direct: команда не найдена
23
allomakina@dkn75 ~$ ./script3.sh
point out format
txt
point out direct
format = txt
direct = .
23
allomakina@dkn75 ~$ ./script3.sh
point out format
sh
point out direct
format = sh
direct = .
7
allomakina@dkn75 ~$ touch os/(1..10).txt
allomakina@dkn75 ~$ ./script3.sh
point out format
os
point out direct
C
allomakina@dkn75 ~$ ./script3.sh
point out format
txt
point out direct
os
format = txt
direct = os
10
allomakina@dkn75 ~$
```

Рис. 3.1: Задание №4

## 3.5 Контрольные вопросы:

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;

– С-оболочка (или `ssh`) — надстройка над оболочкой Борна, использующая подобный 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`. 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 0.

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 отобрать 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 промотора 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 ее трассировку; `-fx` — экспортирует все перечисленные функции в любые дочерние программы оболочек; `-fu` — означает 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`;
- `${name?value}` — останавливает выполнение, если имя переменной не определено, и выводит `value` как сообщение об ошибке;
- `${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`.

## 4 Вывод

Я изучить основы программирования в оболочке ОС UNIX/Linux. Научилась писать небольшие командные файлы.