

# **Лабораторная работа №10**

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

Панченко Денис Дмитриевич

# Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Вывод	10
4	Контрольные вопросы	11

## Список иллюстраций

2.1	Файл . . . . .	5
2.2	Архиватор . . . . .	5
2.3	Скрипт . . . . .	6
2.4	Файл . . . . .	6
2.5	Скрипт . . . . .	6
2.6	Проверка . . . . .	7
2.7	Файл . . . . .	7
2.8	Вырезка . . . . .	7
2.9	Проверка . . . . .	8
2.10	Файл . . . . .	8
2.11	Скрипт . . . . .	9
2.12	Проверка . . . . .	9

# 1 Цель работы

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

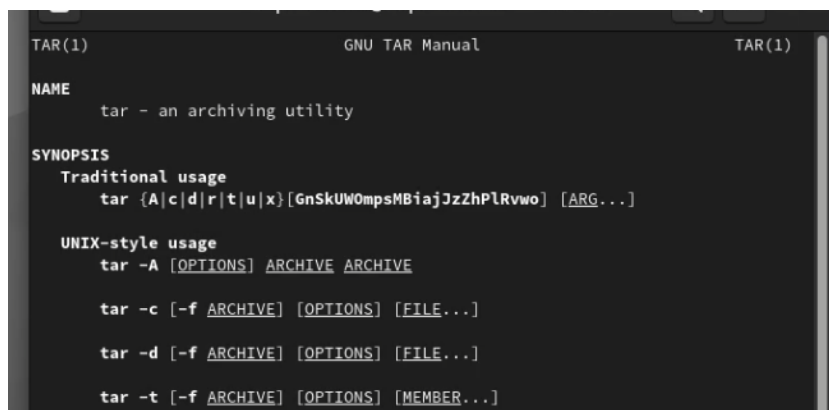
## 2 Выполнение лабораторной работы

Создаем файл для написания первого скрипта (рис. 2.1).

```
[ddpanchenko@ddpanchenko ~]$ touch script.sh  
[ddpanchenko@ddpanchenko ~]$ chmod +x script.sh
```

Рис. 2.1: Файл

Узнаем об архиваторе tar (рис. 2.2).



```
TAR(1)                                GNU TAR Manual                                TAR(1)  
  
NAME  
    tar - an archiving utility  
  
SYNOPSIS  
    Traditional usage  
        tar {A|c|d|r|t|u|x}[GnSkUW0mpsMBiajJzZhPlRvwo] [ARG...]  
  
    UNIX-style usage  
        tar -A [OPTIONS] ARCHIVE ARCHIVE  
  
        tar -c [-f ARCHIVE] [OPTIONS] [FILE...]  
  
        tar -d [-f ARCHIVE] [OPTIONS] [FILE...]  
  
        tar -t [-f ARCHIVE] [OPTIONS] [MEMBER...]
```

Рис. 2.2: Архиватор

Пишем скрипт, который при запуске будет делать резервную копию самого себя в другую директорию backup в нашем домашнем каталоге (рис. 2.3).

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

Рис. 2.3: Скрипт

Создаем файл для второго скрипта (рис. 2.4).

```
[ddpanchenko@ddpanchenko ~]$ touch script2.sh
[ddpanchenko@ddpanchenko ~]$ chmod +x script2.sh
```

Рис. 2.4: Файл

Пишем скрипт, обрабатывающий любое произвольное число аргументов командной строки (рис. 2.5).

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

Рис. 2.5: Скрипт

Проверяем скрипт (рис. 2.6).

```
[ddpanchenko@ddpanchenko ~]$ ./script2.sh
Vvedite znachenie
1 2 3 4 5
1 2 3 4 5
```

Рис. 2.6: Проверка

Создаем третий файл для скрипта (рис. 2.7).

```
[ddpanchenko@ddpanchenko ~]$ touch file.sh
[ddpanchenko@ddpanchenko ~]$ chmod +x file.sh
```

Рис. 2.7: Файл

Пишем командный файл — аналог команды ls (рис. 2.8).

```
#!/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 tes -r $A
then echo readable
else echo neither readable nor writeable
fi
fi
done|
```

Рис. 2.8: Вырезка

Проверяем скрипт (рис. 2.9).

```
[ddpanchenko@ddpanchenko ~]$ ./file.sh
australia: is a directory
conf.txt: is a file andwriteable
feathers: is a file andwriteable
file.sh: is a file andwriteable
file.txt: is a file andwriteable
my_os: is a file and./file.sh: строка 8: tes: команда не найдена
neither readable nor writeable
play: is a directory
script2.sh: is a file andwriteable
script.sh: is a file andwriteable
ski.places: is a directory
text.txt: is a file andwriteable
work: is a directory
Видео: is a directory
Документы: is a directory
Загрузки: is a directory
Изображения: is a directory
Музыка: is a directory
Общедоступные: is a directory
```

Рис. 2.9: Проверка

Создаем файл для четвертого скрипта (рис. 2.10).

```
[ddpanchenko@ddpanchenko ~]$ touch file2.sh
[ddpanchenko@ddpanchenko ~]$ chmod +x file2.sh
```

Рис. 2.10: Файл

Пишем командный файл, который получает в качестве аргумента командной строки формат файла и вычисляет количество таких файлов в указанной директории. (рис. 2.11).



```
#!/bin/bash
direct=''
form=''
echo 'write format'
read form
echo 'write directory'
read direct
find "$direct" -name "*.form" -type f | wc -l
ls|
```

Рис. 2.11: Скрипт

Проверяем скрипт (рис. 2.12).

```
[ddpanchenko@ddpanchenko ~]$ ./file2.sh
write format
txt
write directory
work
0
australia  file2.sh  my_os      script.sh  Видео      Изображения  'Рабочий стол'
conf.txt   file.sh   play       ski.places Документы    Музыка        Шаблоны
```

Рис. 2.12: Проверка

## **3 Вывод**

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

## 4 Контрольные вопросы

1. Командная оболочка (shell) - это интерпретатор, который принимает команды пользователя и выполняет их на уровне операционной системы. Примеры командных оболочек: `bash`, `sh`, `csh`, `ksh`, `zsh`. Они отличаются синтаксисом и набором возможностей.
2. POSIX (Portable Operating System Interface) - это набор стандартов, которые определяют поведение операционной системы и интерфейс прикладных программ. Это позволяет написать переносимый код, который будет работать на разных операционных системах (Linux, macOS, Unix, Windows и др.).
3. Переменные задаются в виде `name=value`, например `x=10`. Доступ к переменной осуществляется через знак `$`, например `$x`. Массивы задаются в виде `array=(value1 value2 value3)`, а доступ к элементу массива осуществляется через индекс в квадратных скобках, например `${array[0]}`.
4. Оператор `let` используется для выполнения арифметических операций и присваивания результата переменной, например `let x=1+2`. Оператор `read` используется для чтения ввода пользователя в переменную, например `read x`.
5. В `bash` можно использовать арифметические операции `+`, `-`, `*`, `/`, `%`, `**`.
6. Операция `(( ))` используется для выполнения арифметических операций и вычисления выражений в скобках, например `((x=1+2))`.

7. Некоторые стандартные имена переменных: `$HOME` (домашняя директория), `$PATH` (список директорий для поиска исполняемых файлов), `$USER` (имя текущего пользователя), `$PWD` (текущий рабочий каталог).
8. Метасимволы - это специальные символы, которые имеют особое значение в командной строке. Примеры: `*` (соответствует нулю или более символов), `?` (соответствует одному символу), `[ ]` (соответствует символам из заданного набора).
9. Метасимволы можно экранировать с помощью обратной косой черты `\`. Например, `\$`.
10. Командный файл создается в текстовом редакторе и сохраняется с расширением `.sh`. Запустить его можно с помощью команды `./script.sh`. Необходимо предварительно дать файлу право на выполнение с помощью команды `chmod +x script.sh`.
11. Функция задается в виде `function name { commands }`. Вызвать функцию можно, например, `name`.
12. Для проверки типа файла можно использовать команду `test -d filename` для проверки, является ли файл каталогом, и `test -f filename` для проверки, является ли файл обычным файлом.
13. `set` используется для задания опций командной строки и установки значений специальных переменных. `typeset` используется для определения типа переменной (например, числовая или строковая). `unset` используется для удаления переменной.
14. Параметры передаются в виде аргументов командной строки. В командном файле к ним можно обратиться с помощью переменных `$1`, `$2` и т.д.
15. Специальные переменные: `$0` (имя командного файла), `$#` (количество аргументов командной строки), `$@` (список всех аргументов командной строки).

строки), \$? (код возврата последней выполненной команды).