

# Цель работы

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

## Выполнение лабораторной работы

Напишем скрипт, который при запуске будет делать резервную копию самого себя.



```
GNU nano 7.2 backup_self.sh
#!/bin/bash

echo "Starting backup"

SCRIPT_FILE=$(basename "$0")
echo "Script file name: $SCRIPT_FILE"

mkdir -p ~/backup
echo "Backup directory created: ~/backup"

zip ~/backup/"$SCRIPT_FILE".zip "$0"
echo "Script file archived: ~/backup/$SCRIPT_FILE.zip"
```

Тут происходит следующее:

1. `SCRIPT_FILE=$(basename "$0")`: эта строка получает текущее имя файла сценария с помощью команды `basename`.
2. `mkdir -p ~/backup`: эта строка создает каталог `backup` в домашнем каталоге, если он еще не существует.
3. `zip ~/backup/"$SCRIPT_FILE".zip "$0"`: эта строка архивирует файл сценария с помощью `zip`.

Запустим его.

```
irulev@irulev:~$ nano backup_self.sh
irulev@irulev:~$ chmod +x backup_self.sh
irulev@irulev:~$ ./backup_self.sh
Starting backup
Script file name: backup_self.sh
Backup directory created: ~/backup
adding: backup_self.sh (deflated 40%)
Script file archived: ~/backup/backup_self.sh.zip
irulev@irulev:~$ find ~ | grep backup
/home/irulev/.mozilla/firefox/kn4qa3gx.default-release/bookmarkbackups
/home/irulev/.mozilla/firefox/kn4qa3gx.default-release/sessionstore-backups
/home/irulev/.cache/gnome-software/icons/d174bbd9ab4735bc7aa63ae31807054cc30bc692-com.darhon.syncbackup.png
/home/irulev/backup_self.sh
/home/irulev/backup
/home/irulev/backup/backup_self.sh.zip
irulev@irulev:~$
```

Напишем скрипт обрабатывающий любое произвольное число аргументов командной строки, в том числе превышающее десять.

```
GNU nano 7.2 process_args.sh
#!/bin/bash

echo "Процесс скрипта с аргументами..."

for arg in "$@"; do
    echo "Аргумент: $arg"
done
```

В этом примере `$@` — это специальная переменная `bash`, содержащая все аргументы командной строки. Цикл `for` проходит по каждому аргументу и выводит его значение с помощью `echo`.

```
irulev@irulev:~$ nano process_args.sh
irulev@irulev:~$ ./process_args.sh arg1 arg2 arg3 arg4 arg5
bash: ./process_args.sh: Отказано в доступе
irulev@irulev:~$126$ chmod +x process_args.sh
irulev@irulev:~$ ./process_args.sh arg1 arg2 arg3 arg4 arg5
Процесс скрипта с аргументами...
Аргумент: arg1
Аргумент: arg2
Аргумент: arg3
Аргумент: arg4
Аргумент: arg5
irulev@irulev:~$
```

Напишем 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.

```
GNU nano 7.2      myls.sh      Изменён
#!/bin/bash

print_file_info() {
    local file=$1
    if [ -e "$file" ]; then
        local perms=$(stat -c "%a" "$file")
        local owner=$(stat -c "%u" "$file")
        local group=$(stat -c "%g" "$file")
        local size=$(stat -c "%s" "$file")
        local mtime=$(stat -c "%y" "$file")

        echo "${perms} ${owner}:${group} ${size} ${mtime} ${file}"
    else
        echo "Error: $file does not exist"
    fi
}

print_dir_info() {
    local dir=$1
    if [ -e "$dir" ]; then
        echo "Directory: $dir"
        for file in "$dir"/*; do
            if [ -e "$file" ]; then
                if [ -d "$file" ]; then
                    print_dir_info "$file"
                else
                    print_file_info "$file"
                fi
            fi
        done
    else
        echo "Error: $dir does not exist"
    fi
}

if [ $# -eq 0 ]; then
    print_dir_info "."
else
    for arg in "$@"; do
        if [ -d "$arg" ]; then
            print_dir_info "$arg"
        else
            print_file_info "$arg"
        fi
    done
fi
```

```
irulev@irulev:~$ nano myls.sh
irulev@irulev:~$ chmod +x myls.sh
irulev@irulev:~$ ls
archive.sh      check_number    man_command.sh  process_args.sh  semaphore.lock  Видео          Изображения    'Рабочий стол'
backup          check_number.c  myls.sh         random.sh         semaphore.sh     Документы      Музыка          Шаблоны
backup_self.sh  file_manager.sh output.txt       search_script.sh  work            Загрузки       Общедоступные
irulev@irulev:~$ ./mys.sh
Directory: .
644 irulev:irulev 178 2024-05-17 21:40:41.300176831 +0300 ./archive.sh
Directory: ./backup
644 irulev:irulev 334 2024-06-09 11:00:26.387500384 +0300 ./backup/backup_self.sh.zip
755 irulev:irulev 258 2024-06-09 10:59:52.413007310 +0300 ./backup_self.sh
755 irulev:irulev 23568 2024-05-17 21:25:10.406067757 +0300 ./check_number
644 irulev:irulev 450 2024-05-17 21:18:22.637403620 +0300 ./check_number.c
755 irulev:irulev 653 2024-05-17 21:41:50.157137098 +0300 ./file_manager.sh
755 irulev:irulev 655 2024-05-26 18:56:30.828003069 +0300 ./man_command.sh
755 irulev:irulev 1057 2024-06-09 11:12:52.720207892 +0300 ./mys.sh
644 irulev:irulev 0 2024-05-17 21:15:21.102928154 +0300 ./output.txt
755 irulev:irulev 140 2024-06-09 11:07:03.593298822 +0300 ./process_args.sh
755 irulev:irulev 921 2024-05-26 19:00:47.201532292 +0300 ./random.sh
755 irulev:irulev 1400 2024-05-17 21:08:53.520789988 +0300 ./search_script.sh
644 irulev:irulev 0 2024-05-26 18:54:59.124505908 +0300 ./semaphore.lock
755 irulev:irulev 957 2024-05-26 18:53:07.602784460 +0300 ./semaphore.sh
Directory: ./work
Directory: ./work/os
Directory: ./work/os/lab06
755 irulev:irulev 94 2024-04-25 21:01:54.101375201 +0300 ./work/os/lab06/hello.sh
644 irulev:irulev 94 2024-04-25 21:01:39.379165735 +0300 ./work/os/lab06/q
Directory: ./work/os/lab_prog
755 irulev:irulev 23889 2024-06-02 10:55:37.104492380 +0300 ./work/os/lab_prog/calcul
644 irulev:irulev 1435 2024-06-02 10:50:58.221302600 +0300 ./work/os/lab_prog/calculate.c
644 irulev:irulev 175 2024-06-02 10:51:50.067068831 +0300 ./work/os/lab_prog/calculate.h
644 irulev:irulev 3752 2024-06-02 10:55:24.093283818 +0300 ./work/os/lab_prog/calculate.o
644 irulev:irulev 410 2024-06-02 10:55:11.934088914 +0300 ./work/os/lab_prog/main.c
644 irulev:irulev 1992 2024-06-02 10:55:31.244398447 +0300 ./work/os/lab_prog/main.o
644 irulev:irulev 286 2024-06-02 10:58:38.585229372 +0300 ./work/os/lab_prog/Makefile
Directory: ./Видео
Directory: ./Документы
Directory: ./Загрузки
Directory: ./Изображения
Directory: ./Музыка
Directory: ./Общедоступные
Directory: ./Рабочий стол
Directory: ./Шаблоны
irulev@irulev:~$
```

Напишем 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.

```
irulev@irulev:~$ nano count_files.sh
GNU nano 7.2 count_files.sh Изменён
#!/bin/bash

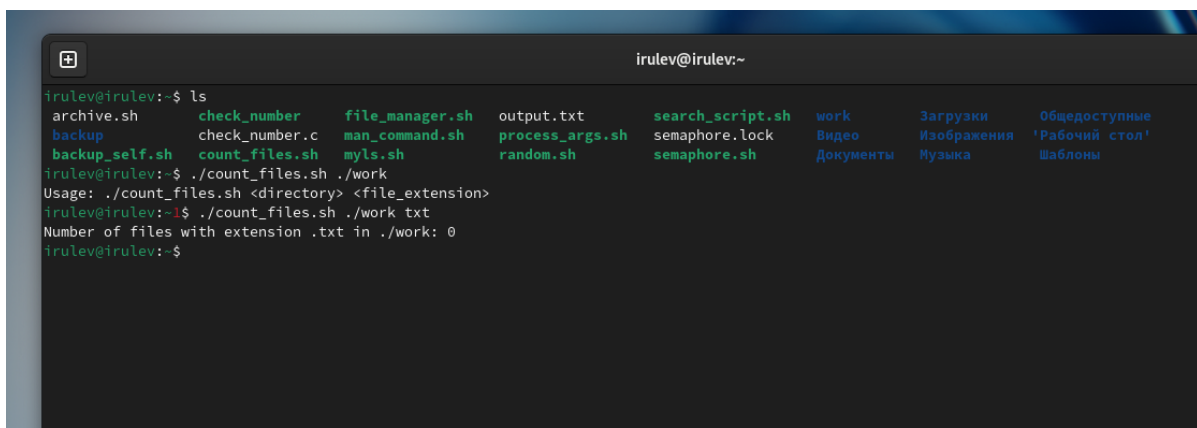
if [ $# -ne 2 ]; then
    echo "Usage: $0 <directory> <file_extension>"
    exit 1
fi

directory=$1
file_extension=$2

count=$(find "$directory" -type f -name ".*$file_extension" | wc -l)

echo "Number of files with extension .$file_extension in $directory: $count"
```

⌘ Справка ⌘ Записать ⌘ Поиск ⌘ Вырезать ⌘ Выполнить ⌘ Позиция ⌘ Отмена ⌘ Установить ⌘ На скобку  
⌘ Выход ⌘ Читфайл ⌘ Замена ⌘ Вставить ⌘ Выводить ⌘ К строке ⌘ Повтор ⌘ Копировать ⌘ Обр. поиск



```
irulev@irulev:~$ ls
archive.sh      check_number    file_manager.sh  output.txt      search_script.sh  work            Загрузки        Общедоступные
backup          check_number.c  man_command.sh  process_args.sh semaphore.lock     Видео           Изображения     'Рабочий стол'
backup_self.sh  count_files.sh  myls.sh          random.sh        semaphore.sh       Документы       Музыка          Шаблоны
irulev@irulev:~$ ./count_files.sh ./work
Usage: ./count_files.sh <directory> <file_extension>
irulev@irulev:~$ ./count_files.sh ./work txt
Number of files with extension .txt in ./work: 0
irulev@irulev:~$
```

## Выводы

Изучили основы программирования в оболочке ОС UNIX/Linux. Научились писать небольшие командные файлы.

## Ответы на контрольные вопросы

1. **Что такое командная оболочка? Приведите примеры командных оболочек. Чем они отличаются?** Командная оболочка — это программа, которая позволяет пользователям взаимодействовать с операционной системой путем выполнения команд, сценариев и программ. Примеры командных оболочек: Bash, pdksh, tcsh и zsh.
2. **Что такое POSIX?** POSIX (Portable Operating System Interface) — это набор стандартов для операционных систем, включая Unix и Linux, определяющий общий API для взаимодействия с операционной системой.
3. **Как определяются переменные и массивы в Bash?** В Bash переменные определяются с помощью оператора `=`, а массивы определяются с помощью круглых скобок `()` и запятой `,` для разделения элементов.
4. **Какова цель операторов let и read?** Оператор let используется для выполнения арифметических операций, а оператор read используется для чтения ввода от пользователя или файла.
5. **Какие арифметические операции можно выполнять в Bash?** Bash поддерживает базовые арифметические операции, такие как сложение, вычитание, умножение и деление, а также более сложные операции, такие как по модулю и возведение в степень.
6. **Что означает операция (( ))?** Операция `(( ))` используется для выполнения арифметических операций и вычисления выражений в Bash.
7. **Какие стандартные имена переменных вы знаете?** Стандартные имена переменных в Bash включают, среди прочего, SHELL, PATH, HOME и USER.
8. **Что такое метасимволы?** Метасимволы — это специальные символы в Bash, имеющие определенное значение, например `*`, `?` и `[]`, которые используются для сопоставления с образцом и подстановки под шаблон.
9. **Как избежать метасимволов?** Метасимволы можно экранировать с помощью обратной косой черты `\` или заключая их в кавычки.
10. **Как создавать и запускать командные файлы?** Командные файлы, также известные как сценарии, можно создавать с помощью текстового редактора и запускать с использованием нотации `./`, за которой следует имя сценария.

11. **Как определяются функции в Bash?** Функции в Bash определяются с помощью ключевого слова `function`, за которым следует имя функции и аргументы в круглых скобках.
12. **Как определить, является ли файл каталогом или обычным файлом?** Вы можете использовать команду `test` или оператор `[`, чтобы определить, является ли файл каталогом или обычным файлом, используя параметры `-d` и `-f` соответственно.
13. **Какова цель команд `set`, `typeset` и `unset`?** Команда `set` используется для установки параметров оболочки, `typeset` используется для объявления переменных, а `unset` используется для удаления переменных или функций.
14. **Как параметры передаются в командные файлы?** Параметры можно передавать в командные файлы с использованием синтаксиса `$1`, `$2` и т. д., который представляет первый, второй и т. д. аргумент командной строки.
15. **Что такое специальные переменные в Bash и каково их назначение?** К специальным переменным в Bash относятся, среди прочего, `SHELL`, `PATH`, `HOME` и `USER`, которые используются для хранения информации об оболочке и среде пользователя.