

### Изучение команды find

#### Задание

1. Создать каталог и в нем несколько файлов
2. С использованием команды touch установить для созданных файлов различное время последнего доступа
3. С использованием find найти и переместить в указанный каталог, например ./old, файлы, к которым не было обращений более 14 дней
4. Каталог ./old исключить из поиска

#### Результаты выполнения задания

Создадим тестовый каталог ./test и три файла в нем

```
Before modification:
total 0
-rw-rw-r-- 1 andrey andrey 0 Oct  3 11:47 test1
-rw-rw-r-- 1 andrey andrey 0 Oct  3 11:47 test2
-rw-rw-r-- 1 andrey andrey 0 Oct  3 11:47 test3
```

Рисунок 1 – каталог после создания файлов

Изменим время последней модификации файла с помощью команды touch: поставим флаг -d и время так, чтобы первые два файла попали бы под работу команды find

```
After modification:
total 0
-rw-rw-r-- 1 andrey andrey 0 Jan  1 2020 test1
-rw-rw-r-- 1 andrey andrey 0 Jan  1 2023 test2
-rw-rw-r-- 1 andrey andrey 0 Sep 24 11:47 test3
```

Рисунок 2 – каталог после модификации файлов

Теперь переместим все файлы, которые не были модифицированы в течение 14 дней, в папку ./old, исключенную при поиске.

Для перемещения можно поставить флаг -exec и прописать команду перемещения так, чтобы вывод команды find поступал команде mv, которая и осуществляет перемещения.

```
After moving:
.:
total 4
drwxrwxr-x 2 andrey andrey 4096 Oct  3 11:47 old
-rw-rw-r-- 1 andrey andrey    0 Sep 24 11:47 test3

./old:
total 0
-rw-rw-r-- 1 andrey andrey 0 Jan  1 2020 test1
-rw-rw-r-- 1 andrey andrey 0 Jan  1 2023 test2
```

Рисунок 3 – каталог после перемещения файлов

Как мы видим, все файлы, которые не были модифицированы в течение 14 дней, были успешно перемещены в папку ./old.

### **Вывод**

Изучена команда find, изучены и реализованы простейшие манипуляции над файлами и взаимодействия с ними.

## Приложение А. Листинг программы

```
#!/bin/bash

# make test directory and three files here
mkdir test
cd test
touch test1 test2 test3

printf "%s\n" "Before modification:"
ls -l

# set last modification time

# 01 January, 2020
touch -d 20200101 test1
# 01 January, 2023
touch -d 20230101 test2
# 9 days ago relative to now
touch -d 'now - 9 days' test3

# check that last mod time has changed
printf "\n%s\n" "After modification:"
ls -l
# make ./old dir
mkdir old
# find files which hasn't been modified for 14 days
# and move to ./old dir
find . -name "test*" -not -path './old/*' -mtime +14 -exec mv -f {} old/ \;

# check files in directories
printf "\n%s\n" "After moving:"
ls -lR

# clear spece in the end
cd ..
rm -r test
```

## Изучение сценариев командной оболочки

### Задание

1. Написать скрипт `ptxt`, который выводит на экран заданный текст заданное количество раз через заданный интервал времени. Предусмотреть следующий синтаксис вызова `ptxt -n <число выводов текста> -t <таймаут вывода> -- <текст для вывода>`  
**использовать команды `case` и `shift` для получения параметров**
2. Написать скрипт `ptxt` как в задании 1  
**использовать `getopts`**
3. Написать скрипт, который сортирует и выводит переданные в него аргументы в алфавитном порядке  
**использовать массивы и сортировку массива любым методом**

### Результаты выполнения задания

Первая программа считывает переданные аргументы, пытается их присвоить соответствующим переменным (если аргументов нет или хотя бы один не корректен, то программа завершится досрочно, выведив сообщение об ошибке).

Проход по переменным осуществляется с использованием команд `case/shift`.

Если прописать соответствующую команду, то, как и ожидалось, скрипт выведет текст `n` раз с установленным таймаутом в `t` секунд.

```
● andrey@localhost:~/code/operating_systems$ ./ptxt_v1 -n 5 -t 1 -- Hello world!  
Hello world!  
Hello world!  
Hello world!  
Hello world!  
Hello world!
```

Рисунок 1 – Вывод программы `ptxt_v1`

Аналогично обрабатывает и вторая программа, однако проход по аргументам осуществляется с помощью команды `getopts`.

```
● andrey@localhost:~/code/operating_systems$ ./ptxt_v2 -n 5 -t 1 -- Hello world!  
Hello world!  
Hello world!  
Hello world!  
Hello world!  
Hello world!
```

Рисунок 2 – Вывод программы `ptxt_v2`

Третья программа сортирует все поступающие в нее аргументы и, упорядочивая их в алфавитном порядке, выводит в консоль. Сортировка здесь осуществляется методом пузырька.

```
● andrey@localhost:~/code/operating_systems$ ./sort My Hello world
Array in original order:
My Hello world
Array in sorted order:
Hello My world
```

Рисунок 3 – Вывод программы sort, только строки.

Однако, если передать в скрипт числа, то он также будет обрабатывать их как строки.

```
● andrey@localhost:~/code/operating_systems$ ./sort 5 1 Hello world!
Array in original order:
5 1 Hello world!
Array in sorted order:
1 5 Hello world!
```

Рисунок 1 – Вывод программы sort

## Выводы

Изучены механизмы написания скриптов, обработки аргументов, их парсинга, а также основных конструкций bash-скриптов. Реализованы программы, обрабатывающие аргументы и проводящие с их помощью различные манипуляции.

## Приложение 1. Код программ

### 1-я программа

```
#!/bin/bash

# ptxt_v1 - script prints text n times with timeout in t seconds
# using case-shift construction

script_name=$0

# func that prints help string
function usage() { echo "Usage: $script_name -n <text print loops> -t <output
timeout> -- <text>"; }

# func that check param before parsing
function check_num() {
    # if parameter are not a number
    if [[ ! "$2" =~ ^[0-9]+$ ]];
    then
        # print incorrect arg
        echo "Incorrect argument: $1 = $2"
        # print usage string and exit
        usage
        exit 1
    fi;
}

# if no arguments
if [[ $# -eq 0 ]];
then
    echo "No arguments"
    usage
    exit 1
fi

# parse arguments by case-shift
while [[ $# -gt 0 ]]; do
    case $1 in
        # help
        -h)
            usage
            exit 0
            ;;
        # number of loops
        -n)
            # check that it is a num
            check_num $1 $2

            # parse integer
            n=$(( $2 ))
            shift 2
```

```

        ;;
        # output timeout
        -t)
            # check that it is a num
            check_num $1 $2

            t=$2
            shift 2
            ;;
        # text to print
        --)
            shift 1
            text="$@"
            break
            ;;
        # unknown options
        *)
            echo "Invalid flag: $1"
            usage
            exit 1
            ;;
    esac
done

# print text n times
for (( i=1; i<=$n; i++))
do
    echo $text
    # timeout on t seconds
    sleep $t
done

```

## 2-я программа

```

#!/bin/bash

# ptxt_v2 - script prints text n times with timeout in t seconds
# using getops command

script_name=$0

# func that prints help string
function usage() { echo "Usage: $script_name -n <print loops num> -t <output
timeout> -- <text>"; }

# func that check param before parsing
function check_num() {

```

```

# if parameter are not a number
if [[ ! "$2" =~ ^[0-9]+$ ]];
then
    # print incorrect arg
    echo "Incorrect argument: -$1 = $2"
    # print usage string and exit
    usage
    exit 1
fi;
}

# if no arguments
if [[ $# -eq 0 ]];
then
    echo "No arguments"
    usage
    exit 0
fi

# parse arguments by getopt
while getopt "n:t:h" flag
do
    case $flag in
        # help
        h)
            usage
            exit 1
            ;;
        # number of loops
        n)
            check_num $flag $OPTARG
            n=$((OPTARG))
            ;;
        # output timeout
        t)
            check_num $flag $OPTARG
            t=$OPTARG
            ;;
        # invalid options
        *)
            echo "Incorrect flag: $flag"
            usage
            exit 0
    esac
done

# shift parameters to remaining args and read them as text
shift $(expr $OPTIND - 1)
text="$@"

# print text n times

```



```
for (( i=1; i<=$n; i++))
do
    echo $text
    # timeout on t seconds
    sleep $t
done
```

### 3-я программа

```
#!/bin/bash

# sort - script gives arguments and sort them as an array of strings

# target array and his length
arr=("$@")
len=$#

printf "%s\n" "Array in original order:"
echo ${arr[*]}

# performing bubble sort
for ((i = 0; i < $len; i++))
do
    for((j = 0; j < $len-i-1; j++))
    do
        # if string is "greater" than another one
        # (sorting by alphabetic order as strings)
        if [[ ${arr[j]} > ${arr[$((j+1))]} ]]
        then
            # swap items
            temp=${arr[j]}
            arr[j]=${arr[$((j+1))]}
            arr[$((j+1))]=$temp
        fi
    done
done

printf "%s\n" "Array in sorted order:"
echo ${arr[*]}
```

### Изучение сценариев командной оболочки

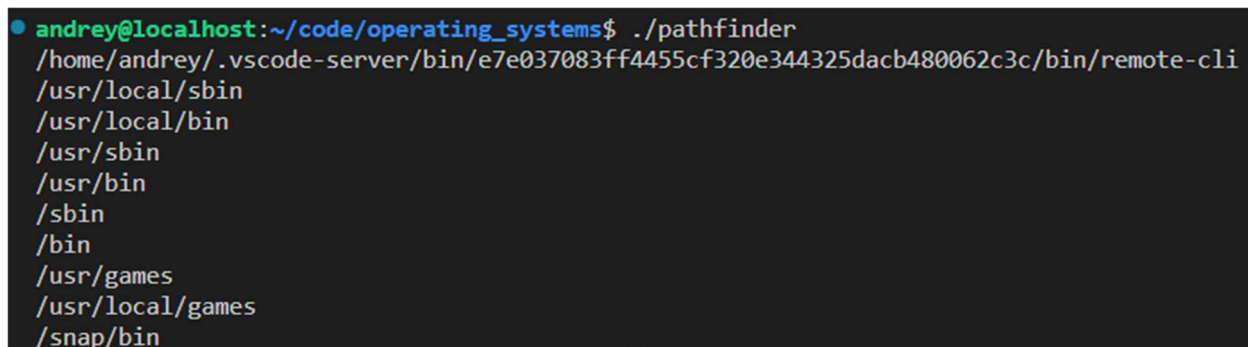
#### Задание

1. Переменная PATH содержит пути поиска исполняемых файлов, разделенные символом ':'  
Написать сценарий, который построчно выводит на экран все пути из PATH, использовать цикл for in
2. Написать сценарий, который построчно выводит на экран имена файлов из указанного каталога и всех его подкаталогов, отобразить уровень вложенности подкаталога числом в начале строки с именем файла в формате n), где n - уровень вложенности подкаталога начиная с 1, использовать рекурсию для отображения данных вложенных каталогов и цикл for in для получения списка файлов

#### Результаты выполнения задания

Первая программа считывает содержимое переменной \$PATH и записывает его в массив, разделяя поля по указанному сепаратору IFS ":", с помощью команды read.

Далее печатается содержимое массива полей через конструкцию for in.



```
andrey@localhost:~/code/operating_systems$ ./pathfinder
/home/andrey/.vscode-server/bin/e7e037083ff4455cf320e344325dacb480062c3c/bin/remote-cli
/usr/local/sbin
/usr/local/bin
/usr/sbin
/usr/bin
/sbin
/bin
/usr/games
/usr/local/games
/snap/bin
```

Рисунок 1 – Результат работы первой программы

Вторая программа выводит содержимое всех каталогов, начиная с указанного в параметре -d, т.е. все файлы и подкаталоги. Однако уровень рекурсии устанавливается через параметр -n, что позволяет ограничить рекурсию и вывести содержимое всех n уровней. Например, если запустить программу с установленным каталогом выше места запуска, то выводится следующее:

```
● andrey@localhost:~/code/operating_systems$ ./find_files -n 3 -d ..  
rlevel = 3  
  
../course_paper:  
../course_paper/analysis.py  
../course_paper/dll00.c  
../course_paper/dll00.o  
../course_paper/dll00.so  
../course_paper/matrix_parallel.py  
../course_paper/matrix_usual.py  
../course_paper/run.py  
  
../operating_systems:  
../operating_systems/find  
../operating_systems/find_files  
../operating_systems/pathfinder  
../operating_systems/ptxt_v1  
../operating_systems/ptxt_v2  
../operating_systems/sort
```

Рисунок 2 – Вывод второй программы

Очевидно, что если в указанной папке не достигается уровень вложенности, больший чем  $n$ , то скрипт просто выводит все содержимое, имитируя утилиту `ls` с параметрами `-lR`.

### Вывод

Изучены конструкции `for in`, функции `bash`-скриптов, а также реализованы программы обхода каталогов и содержимого переменных наподобие `$PATH`.

## Приложение 1. Код программ

### 1-я программа

```
#!/bin/bash

# ./pathfinder - script prints all fields in $PATH

# internal field separator (separator in sets)
IFS=':'
# read fields from $PATH and store in array
read -r -a array <<< "$PATH"

# loop by entries
for s in ${array[*]}
do
    echo $s
done
```

### 2-я программа

```
#!/bin/bash

# ./find_files - script prints all files in a mentioned dir

script_name=$0

# func that prints help string
function usage() { echo "Usage: $script_name -d <target dir> -n <recursion level number>"; }

function loop(){
    # loop in dir
    for file in $1/*
    do
        # if entry is a dir and rlevel is not zero
        if [[ -d $file ]] && [[ $2 -gt 0 ]];
        then
            # print dirname (like ls command)
            printf "\n%s\n" $file:
            # recursion
            loop $file $(expr $2 - 1)
        else
            # print name
            echo $file
        fi
    done
}

# func that check param before parsing
function check_num() {
    # if parameter are not a number
    if [[ ! "$2" =~ ^[0-9]+$ ]];
    then
```

```

        # print incorrect flag and its value
        echo "Incorrect argument: $1 = $2"
        # print usage string and exit
        usage
        exit 1
    fi;
}

# if no arguments
if [[ $# -eq 0 ]];
then
    usage
    exit 1
fi

# parse arguments by getopt
while getopts "n:d:h" flag
do
    case $flag in
        # help
        h)
            usage
            exit 1
            ;;
        # target directory
        d)
            target=$OPTARG
            ;;
        # nesting level number
        n)
            check_num n $OPTARG
            n=$((OPTARG))
            ;;
        # invalid options
        *)
            echo "Incorrect flag: $flag"
            usage
            exit 0
        esac
    done

    echo rlevel = $n
    loop $target $n

```