

Отчёт по лабораторной работе №10 (*11 в лабнике, 10 в ТУИС)

Программирование в командном процессоре ОС UNIX. Ветвления и циклы

Куликов Александр Андреевич

Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Задание

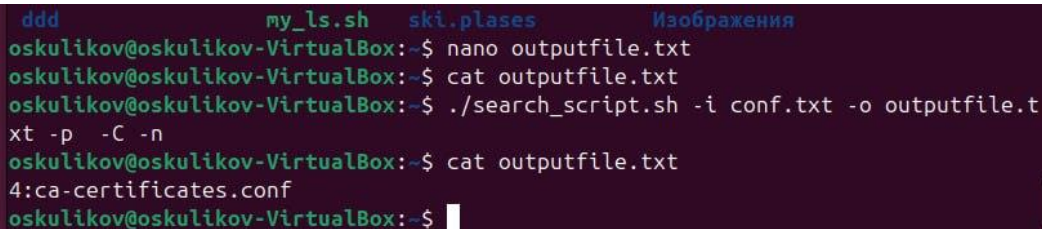
1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами:
 - `-iinputfile` — прочитать данные из указанного файла;
 - `-ooutputfile` — вывести данные в указанный файл;
 - `-rшаблон` — указать шаблон для поиска;
 - `-C` — различать большие и малые буквы;
 - `-n` — выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом `-r`.
2. Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено.
3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например `1.tmp`, `2.tmp`, `3.tmp`, `4.tmp` и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют).
4. Написать командный файл, который с помощью команды `tar` запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду `find`).

Теоретические сведения

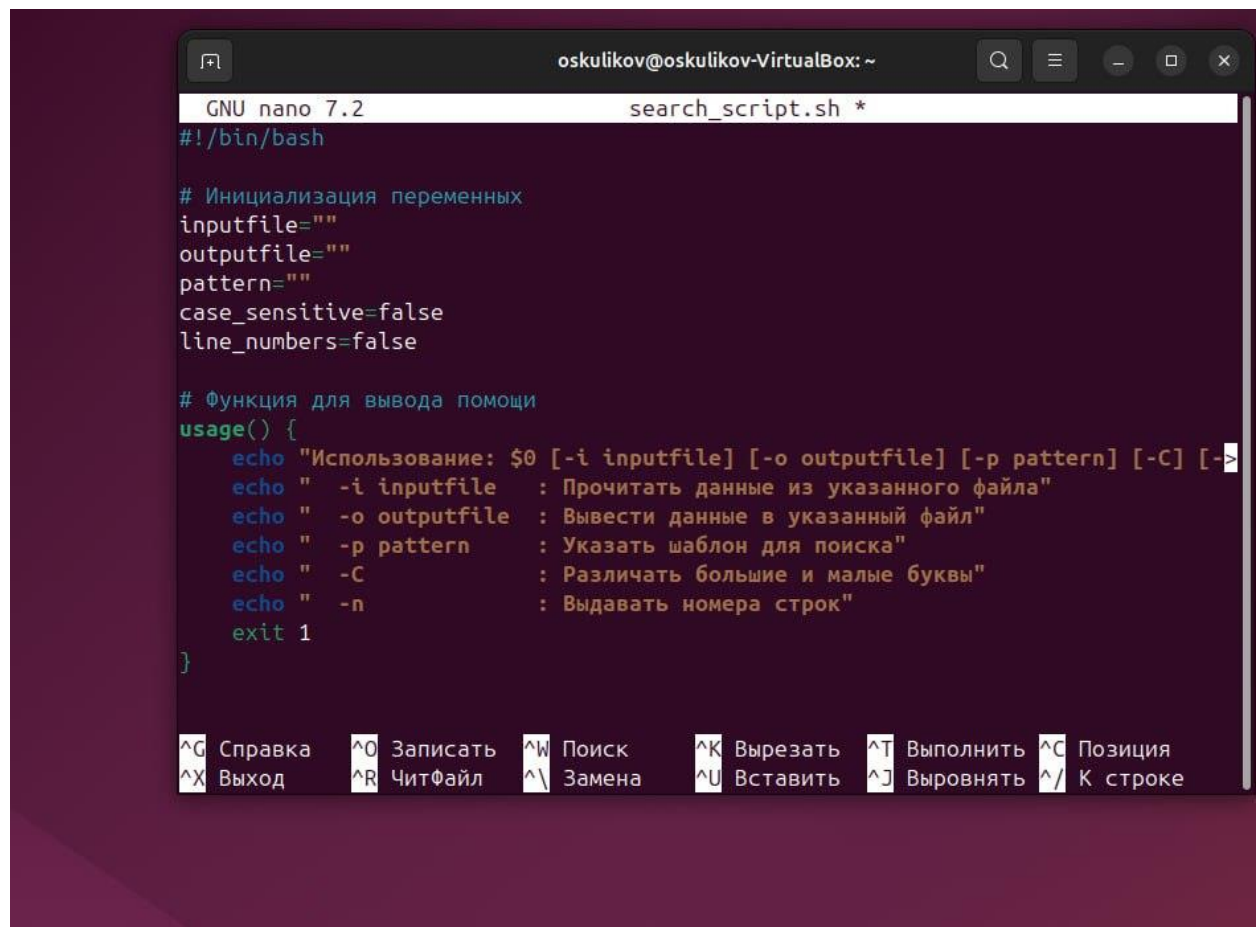
- Командный процессор (командная оболочка, интерпретатор команд shell) - это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:
 - оболочка Борна (Bourne shell или sh) - стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;
 - С-оболочка (или csh) - надстройка на оболочке Борна, использующая С-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).
- 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/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

Выполнение лабораторной работы

1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами:
 - `-iinputfile` — прочитать данные из указанного файла;
 - `-ooutputfile` — вывести данные в указанный файл;
 - `-р`шаблон — указать шаблон для поиска;
 - `-C` — различать большие и малые буквы;
 - `-n` — выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом `-p`. (рис. @fig:001, @fig:002)



```
ddd my_ls.sh ski.places Изображения
oskulikov@oskulikov-VirtualBox:~$ nano outputfile.txt
oskulikov@oskulikov-VirtualBox:~$ cat outputfile.txt
oskulikov@oskulikov-VirtualBox:~$ ./search_script.sh -i conf.txt -o outputfile.t
xt -p -C -n
oskulikov@oskulikov-VirtualBox:~$ cat outputfile.txt
4:ca-certificates.conf
oskulikov@oskulikov-VirtualBox:~$
```



```
oskulikov@oskulikov-VirtualBox: ~
GNU nano 7.2 search_script.sh *
#!/bin/bash

# Инициализация переменных
inputfile=""
outputfile=""
pattern=""
case_sensitive=false
line_numbers=false

# Функция для вывода помощи
usage() {
    echo "Использование: $0 [-i inputfile] [-o outputfile] [-p pattern] [-C] [->
    echo "  -i inputfile   : Прочитать данные из указанного файла"
    echo "  -o outputfile  : Вывести данные в указанный файл"
    echo "  -p pattern     : Указать шаблон для поиска"
    echo "  -C             : Различать большие и малые буквы"
    echo "  -n             : Выдавать номера строк"
    exit 1
}

^G Справка  ^O Записать ^W Поиск    ^K Вырезать ^T Выполнить ^C Позиция
^X Выход    ^R ЧитФайл ^\ Замена   ^U Вставить ^J Выровнять ^/_ К строке
```

-
- результат
- 2. Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено (рис. @fig:003, @fig:004, @fig:005).
-

```
oskulikov@oskulikov-VirtualBox: ~
GNU nano 7.2 check_number.sh *
#!/bin/bash

# Запускаем программу на языке C
./check_number

# Сохраняем код завершения
exit_code=$?

# Анализируем код завершения и выводим сообщение
if [ $exit_code -eq 0 ]; then
    echo "Введенное число равно нулю."
elif [ $exit_code -eq 1 ]; then
    echo "Введенное число больше нуля."
elif [ $exit_code -eq 2 ]; then
    echo "Введенное число меньше нуля."
else
    echo "Неизвестный код завершения: $exit_code"
fi
```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выворнять ^/_ К строке

```
oskulikov@oskulikov-VirtualBox: ~
GNU nano 7.2 manage_files.sh
#!/bin/bash

# Функция для вывода помощи
usage() {
    echo "Использование: $0 <число файлов> [-d]"
    echo "  <число файлов> : Число файлов для создания (если без -d)"
    echo "  -d              : Удалить все созданные файлы"
    exit 1
}

# Проверка аргументов
if [ $# -eq 0 ]; then
    usage
fi

# Обработка опции удаления
if [ "$1" == "-d" ]; then
    for file in *.tmp; do
        if [ -e "$file" ]; then
            rm "$file"
        fi
    done
fi
```

[Прочитано 40 строк]

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выворнять ^/_ К строке

Результат

3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например 1.tmp, 2.tmp, 3.tmp, 4.tmp и т.д.). Число файлов,

которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют) (рис. @fig:006, @fig:007).

```
oskulikov@oskulikov-VirtualBox:~$ ./manage_files.sh 10
Создан файл 1.tmp
Создан файл 2.tmp
Создан файл 3.tmp
Создан файл 4.tmp
Создан файл 5.tmp
Создан файл 6.tmp
Создан файл 7.tmp
Создан файл 8.tmp
Создан файл 9.tmp
Создан файл 10.tmp
oskulikov@oskulikov-VirtualBox:~$ ls
10.tmp  abc1          dfile.old      play          Загрузки
1.tmp   australia    feathers       print_args.sh Изображения
2.tmp   backup       file           reports       Музыка
3.tmp   backup_script.sh file.txt       search_script.sh Общедоступные
4.tmp   check_number manage_files.sh ski.plases    'Рабочий стол'
5.tmp   check_number.c may            snap          Шаблоны
6.tmp   check_number.sh mounthly
7.tmp   conf.txt     my_ls.sh
8.tmp   count_files.sh my_os
9.tmp   ddd          outputfile.txt
oskulikov@oskulikov-VirtualBox:~$
```

```
oskulikov@oskulikov-VirtualBox: ~
GNU nano 7.2 manage_files.sh
#!/bin/bash

# Функция для вывода помощи
usage() {
    echo "Использование: $0 <число файлов> [-d]"
    echo "  <число файлов> : Число файлов для создания (если без -d)"
    echo "  -d              : Удалить все созданные файлы"
    exit 1
}

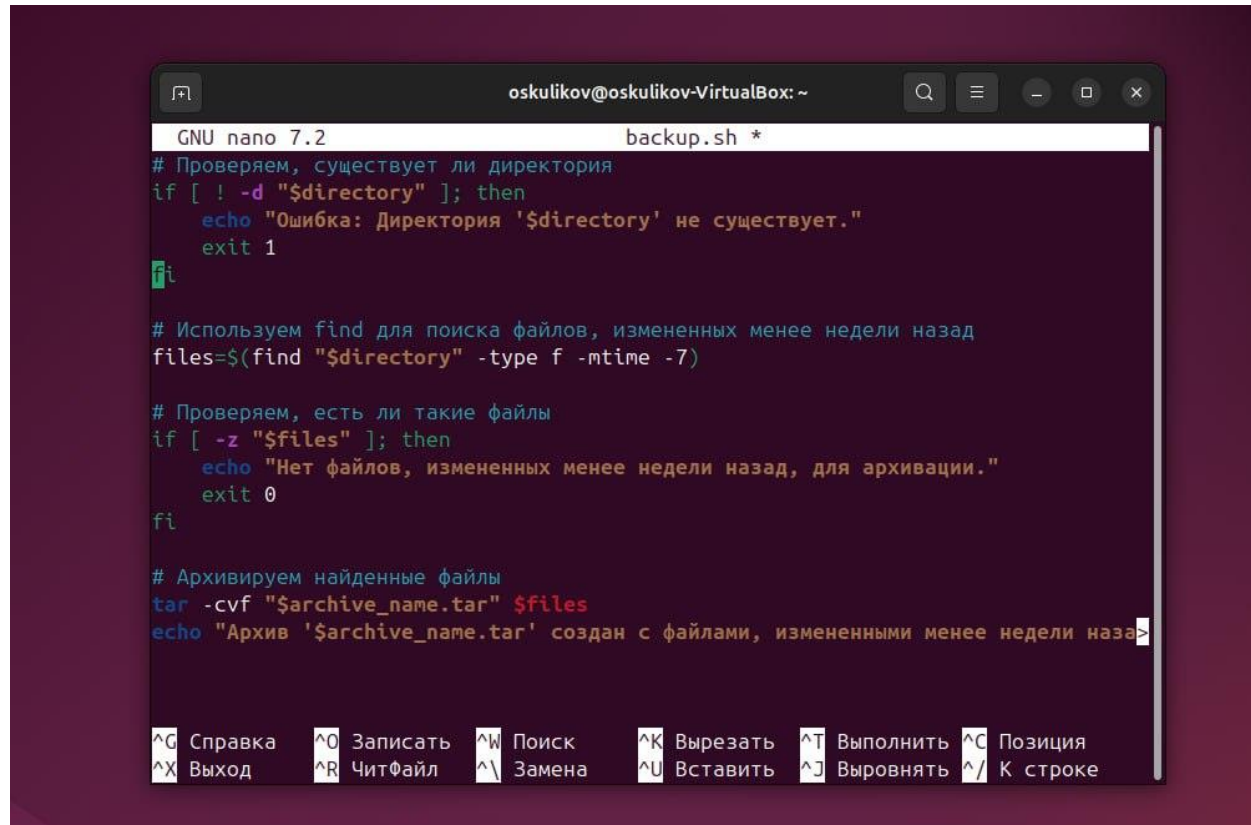
# Проверка аргументов
if [ $# -eq 0 ]; then
    usage
fi

# Обработка опции удаления
if [ "$1" == "-d" ]; then
    for file in *.tmp; do
        if [ -e "$file" ]; then
            rm "$file"
        fi
    done
fi

[ Прочитано 40 строк ]
^G Справка  ^O Записать ^W Поиск   ^K Вырезать ^T Выполнить ^C Позиция
^X Выход    ^R ЧитФайл ^\ Замена  ^U Вставить ^J Выводить  ^/_ К строке
```

Результат

4. Написать командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find) (рис. @fig:008, @fig:009).



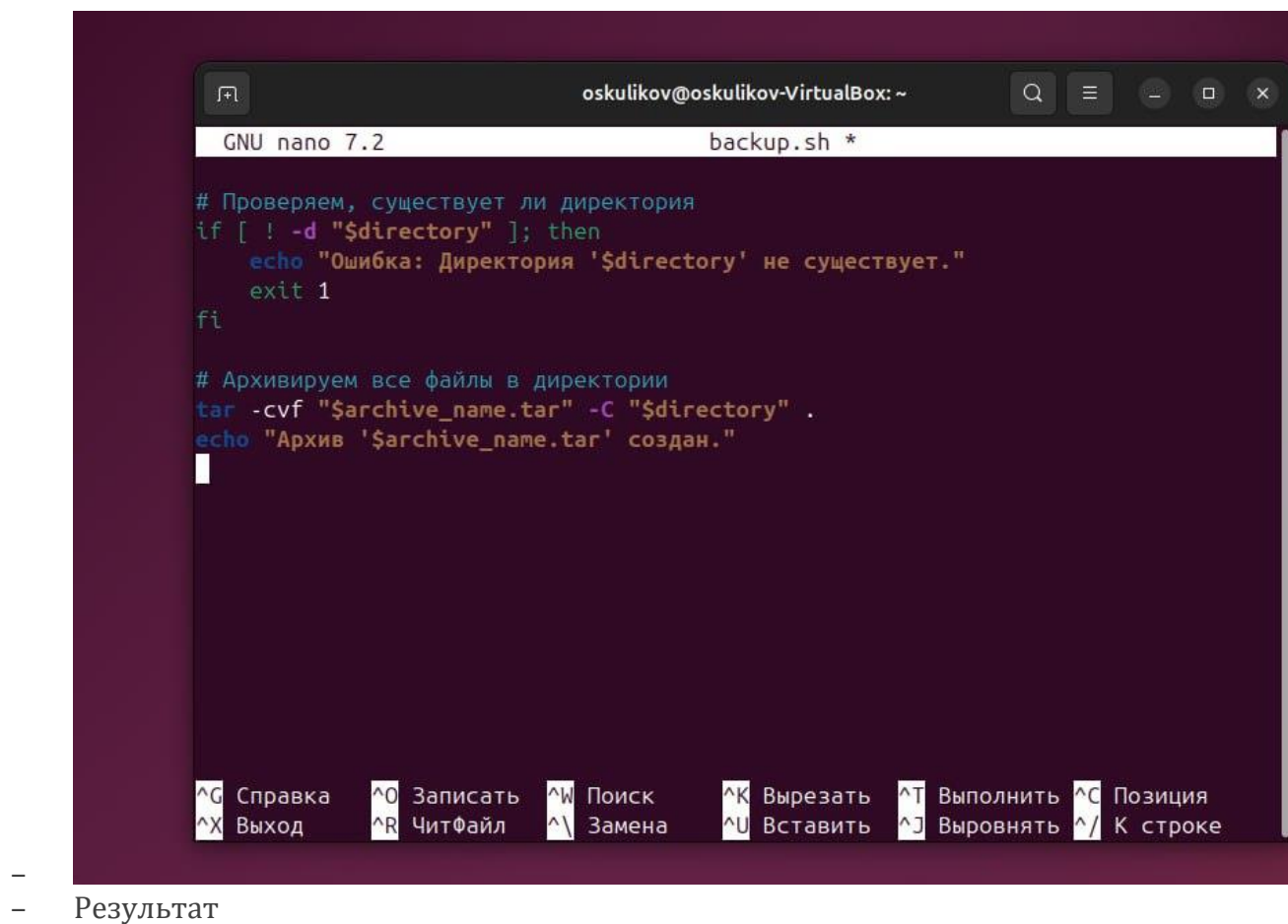
```
oskulikov@oskulikov-VirtualBox: ~
GNU nano 7.2 backup.sh *
# Проверяем, существует ли директория
if [ ! -d "$directory" ]; then
    echo "Ошибка: Директория '$directory' не существует."
    exit 1
fi

# Используем find для поиска файлов, измененных менее недели назад
files=$(find "$directory" -type f -mtime -7)

# Проверяем, есть ли такие файлы
if [ -z "$files" ]; then
    echo "Нет файлов, измененных менее недели назад, для архивации."
    exit 0
fi

# Архивируем найденные файлы
tar -cvf "$archive_name.tar" $files
echo "Архив '$archive_name.tar' создан с файлами, измененными менее недели наза>

^G Справка  ^O Записать  ^W Поиск    ^K Вырезать ^T Выполнить ^C Позиция
^X Выход     ^R ЧитФайл  ^\ Замена  ^U Вставить ^J Выровнять ^_ К строке
```

Листинги

- 1) script1
#!/bin/bash

Инициализация переменных
inputfile=""
outputfile=""
pattern=""
case_sensitive=false
line_numbers=false

Функция для вывода помощи
usage() {
echo "Использование: \$0 [-i inputfile] [-o outputfile] [-p pattern] [-C] [-n]"
echo " -i inputfile : Прочитать данные из указанного файла"
echo " -o outputfile : Вывести данные в указанный файл"
echo " -p pattern : Указать шаблон для поиска"
echo " -C : Различать большие и малые буквы"
echo " -n : Выдавать номера строк"
}

```

    exit 1
}

# Анализ аргументов командной строки
while getopts "i:o:p:Cn" opt; do
    case $opt in
        i) inputfile=$OPTARG ;;
        o) outputfile=$OPTARG ;;
        p) pattern=$OPTARG ;;
        C) case_sensitive=true ;;
        n) line_numbers=true ;;
        *) usage ;;
    esac
done

# Проверка обязательных аргументов
if [ -z "$inputfile" ] || [ -z "$pattern" ]; then
    usage
fi

# Проверка существования входного файла
if [ ! -f "$inputfile" ]; then
    echo "Ошибка: Файл '$inputfile' не существует."
    exit 1
fi

# Построение команды grep
grep_command="grep"

if [ "$case_sensitive" = false ]; then
    grep_command+="-i"
fi

if [ "$line_numbers" = true ]; then
    grep_command+="-n"
fi

grep_command+="-e \"$pattern\" \"$inputfile\""

# Выполнение команды grep и сохранение результата
if [ -z "$outputfile" ]; then
    eval $grep_command

```



```

        else
            eval $grep_command > "$outputfile"
        fi
2) script2
#include <stdio.h>

#include <stdlib.h>

int main() {
    int number;

    printf("Введите число: ");
    scanf("%d", &number);

    if (number > 0) {
        printf("Число больше нуля.\n");
        exit(1); // Код завершения 1 для числа больше нуля
    } else if (number < 0) {
        printf("Число меньше нуля.\n");
        exit(2); // Код завершения 2 для числа меньше нуля
    } else {
        printf("Число равно нулю.\n");
        exit(0); // Код завершения 0 для числа, равного нулю
    }
}

```

```
#!/bin/bash
```

```
# Запускаем программу на языке C
```

```
./check_number
```

```
# Сохраняем код завершения
exit_code=$?

# Анализируем код завершения и выводим сообщение
if [ $exit_code -eq 0 ]; then
    echo "Введенное число равно нулю."
elif [ $exit_code -eq 1 ]; then
    echo "Введенное число больше нуля."
elif [ $exit_code -eq 2 ]; then
    echo "Введенное число меньше нуля."
else
    echo "Неизвестный код завершения: $exit_code"
fi
```

```
3) script3
#!/bin/bash
```

```
# Функция для вывода помощи
usage() {
    echo "Использование: $0 <директория> <имя архива>"
    exit 1
}
```

```
# Проверка аргументов
if [ $# -ne 2 ]; then
    usage
fi
```

```
# Получаем директорию и имя архива
directory=$1
archive_name=$2

# Проверяем, существует ли директория
if [ ! -d "$directory" ]; then
    echo "Ошибка: Директория '$directory' не существует."
    exit 1
fi

# Архивируем все файлы в директории
tar -cvf "$archive_name.tar" -C "$directory" .
echo "Архив '$archive_name.tar' создан."

#!/bin/bash

# Функция для вывода помощи
usage() {
    echo "Использование: $0 <директория> <имя архива>"
    exit 1
}

# Проверка аргументов
if [ $# -ne 2 ]; then
    usage
fi
```

```

# Получаем директорию и имя архива
directory=$1
archive_name=$2

# Проверяем, существует ли директория
if [ ! -d "$directory" ]; then
    echo "Ошибка: Директория '$directory' не существует."
    exit 1
fi

# Используем find для поиска файлов, измененных менее недели назад
files=$(find "$directory" -type f -mtime -7)

# Проверяем, есть ли такие файлы
if [ -z "$files" ]; then
    echo "Нет файлов, измененных менее недели назад, для архивации."
    exit 0
fi

# Архивируем найденные файлы
tar -cvf "$archive_name.tar" $files

echo "Архив '$archive_name.tar' создан с файлами, измененными менее недели назад."

```

Контрольные вопросы

1. Каково предназначение команды `getopts`?
 - Осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable [arg ...]` Флаги - это опции командной строки, обычно помеченные знаком минус; Например, `-F` является флагом для команды `ls -F`. Иногда эти флаги имеют аргументы, связанные с ними.

Программы интерпретируют эти флаги, соответствующим образом изменяя свое поведение. Строка опций `option-string` - это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за этой буквой должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введенные данные с помощью оператора `case`.

2. Какое отношение метасимволы имеют к генерации имён файлов?

- При перечислении имён файлов текущего каталога можно использовать следующие символы:
 - соответствует произвольной, в том числе и пустой строке;
 - `?` - соответствует любому одинарному символу;
 - `[c1-c2]` - соответствует любому символу, лексикографически находящемуся между символами `c1` и `c2`. Например, `echo`;
 - выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`;
 - `ls .c` - выведет все файлы с последними двумя символами, совпадающими с `.c`.
 - `echo prog.?` - выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются `prog.`.
 - `[a-z]` - соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита.

3. Какие операторы управления действиями вы знаете?

- Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости отрезультатов проверки некоторого условия. Для решения подобных задач язык программирования `bash` предоставляет возможность использовать такие управляющие конструкции, как `for`, `case`, `if` и `while`. С точки зрения командного процессора эти управляющие конструкции являются обычными командами и могут использоваться как при создании командных файлов, так и при работе в интерактивном режиме. Команды, реализующие подобные конструкции, по сути, являются операторами языка программирования `bash`. Поэтому при описании языка программирования `bash` термин оператор будет использоваться наравне с термином команда. Команды ОС UNIX возвращают код завершения, значение которого может быть использовано для принятия решения о дальнейших действиях. Команда `test`, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения.

4. Какие операторы используются для прерывания цикла?

- Два несложных способа позволяют вам прерывать циклы в оболочке `bash`. Команда `break` завершает выполнение цикла, а команда `continue` завершает данную итерацию блока операторов. Команда `break` полезна для завершения цикла `while` в ситуациях, когда условие перестаёт быть правильным.

Команда `continue` используется в ситуациях, когда больше нет необходимости выполнять блок операторов, но вы можете захотеть продолжить проверять данный блок на других условных выражениях.

5. Для чего нужны команды `false` и `true`?
 - Следующие две команды ОС UNIX используются только совместно с управляющими конструкциями языка программирования `bash`: это команда `true`, которая всегда возвращает код завершения, равный нулю (т.е. истина), и команда `false`, которая всегда возвращает код завершения, не равный нулю (т. е. ложь).
6. Что означает строка `if test -f mans/i.$s`, встреченная в командном файле?
 - Строка `if test -f mans/i.s`, `mans/i.s` и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернет нулевое значение (ложь).
7. Объясните различия между конструкциями `while` и `until`.
 - Выполнение оператора цикла `while` сводится к тому, что сначала выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `while`, а затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `do`, после чего осуществляется безусловный переход на начало оператора цикла `while`. Выход из цикла будет осуществлён тогда, когда последняя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `while`, возвратит ненулевой код завершения (ложь). При замене в операторе цикла `while` служебного слова `while` на `until` условие, при выполнении которого осуществляется выход из цикла, меняется на противоположное. В остальном оператор цикла `while` и оператор цикла `until` идентичны

Выводы

В процессе выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX. Научился применять ветвление и циклы в написании скриптов.

Список литературы

Руководство к лабораторной работе