

Лабораторная работа №13

дисциплина: Архитектура компьютера

Орлов Илья Сергеевич

Содержание

1	Цель	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Контрольные вопросы	12
6	Выводы	16

Список иллюстраций

4.1	1	9
4.2	2	10
4.3	3	10
4.4	4	11

Список таблиц

1 Цель

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

2 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; `-r` — 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 `-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 `tag` 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`).

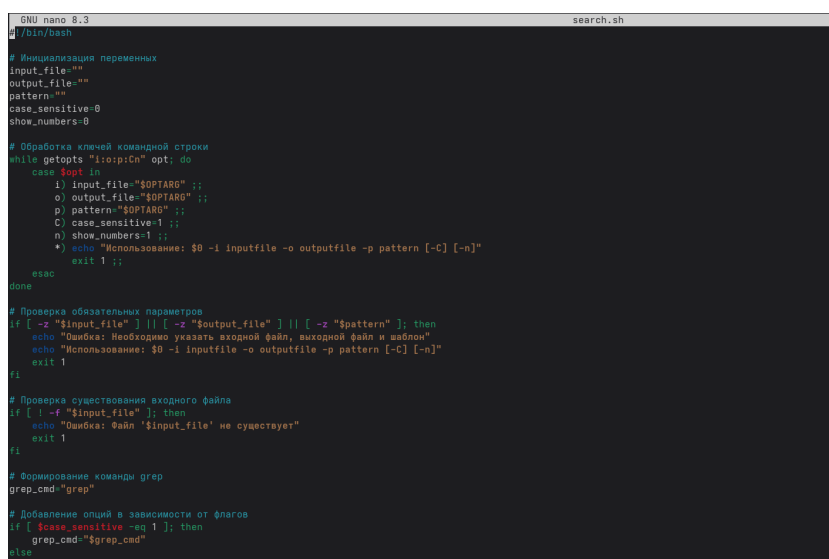
3 Теоретическое введение

Bash (Bourne Again Shell) — это мощная командная оболочка Unix, которая используется для выполнения различных задач в терминале. Bash предоставляет интерактивный интерфейс, в котором пользователи могут вводить команды, а затем получать результаты. Она также поддерживает скрипты оболочки, которые представляют собой текстовые файлы, содержащие последовательность команд Bash для автоматизации задач. Bash широко используется в средах Unix и Linux, а также поддерживается Windows с помощью подсистемы Windows для Linux (WSL). Перечислим основные возможности этой оболочки. Обработка команд. Bash может обрабатывать как простые, так и сложные команды. Простые состоят из одного действия и, возможно, некоторых аргументов. Сложные команды могут содержать несколько простых, объединенных с помощью операторов конвейера (`|`), перенаправления ввода и вывода (`<`, `>`, `>>`), условных операторов (`if/else`, `case/esac`, `while/do`). О них мы расскажем позже. Расширенный ввод, редактирование строк. Оболочка предоставляет функции расширенного ввода, такие как автодополнение, которое предлагает возможные варианты завершения команд и имен файлов по мере их ввода. Она также поддерживает историю, позволяя пользователям просматривать ранее введенные команды, а затем повторно их использовать. Возможность создания и запуска скриптов. Скрипты оболочки — это текстовые файлы, содержащие последовательность команд. Их можно создавать с помощью текстового редактора, а затем запускать в терминале, что дает возможность пользователям автоматизировать задачи и управлять системой. Скрипты могут содержать условные операторы, циклы, функции для обеспече-

ния дополнительной гибкости и контроля.

4 Выполнение лабораторной работы

1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами: `-i`inputfile — прочитать данные из указанного файла; `-o`outputfile — вывести данные в указанный файл; `-r`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 `-r`.(рис. fig. 4.1).



```
GNU nano 8.3 search.sh
#!/bin/bash

# Инициализация переменных
input_file=""
output_file=""
pattern=""
case_sensitive=0
show_numbers=0

# Обработка ключей командной строки
while getopts "i:o:p:C:n" opt; do
  case $opt in
    i) input_file="$OPTARG" ;;
    o) output_file="$OPTARG" ;;
    p) pattern="$OPTARG" ;;
    C) case_sensitive=1 ;;
    n) show_numbers=1 ;;
    *) echo "Использование: $0 -i inputfile -o outfile -p pattern [-C] [-n]"
       exit 1 ;;
  esac
done

# Проверка обязательных параметров
if [ -z "$input_file" ] || [ -z "$output_file" ] || [ -z "$pattern" ]; then
  echo "Ошибка: Необходимо указать входной файл, выходной файл и шаблон"
  echo "Использование: $0 -i inputfile -o outfile -p pattern [-C] [-n]"
  exit 1
fi

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

# Формирование команды grep
grep_cmd="grep"

# Добавление опций в зависимости от флагов
if [ $case_sensitive -eq 1 ]; then
  grep_cmd="$grep_cmd"
else
  grep_cmd="$grep_cmd -i"
fi
```

Рис. 4.1: 1

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. Команд- ный файл должен вызывать эту программу

и, проанализировав с помощью команды \$?, выдать сообщение о том, какое число было введено. (рис. fig. 4.2).

```
GNU nano 8.3
#!/bin/bash

# Запуск программы проверки числа
./check_number

# Анализ кода завершения
exit_code=$?

case $exit_code in
  0) echo "Было введено число: 0" ;;
  1) echo "Было введено положительное число" ;;
  2) echo "Было введено отрицательное число" ;;
  *) echo "Неизвестный код завершения: $exit_code" ;;
esac
```

Рис. 4.2: 2

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. 4.3).

```
#!/bin/bash

# Функция для создания файлов
create_files() {
  local count=$1
  local i=1
  echo "Создание $count файлов..."
  while [ $i -le $count ]; do
    touch "${i}.tmp"
    echo "Создан файл: ${i}.tmp"
    i=$((i + 1))
  done
  echo "Файлы созданы успешно!"
}

# Функция для удаления файлов
delete_files() {
  echo "Удаление файлов *.tmp..."
  local deleted_count=0
  for file in *.tmp; do
    if [ -f "$file" ]; then
      rm "$file"
      echo "Удален: $file"
      deleted_count=$((deleted_count + 1))
    fi
  done
  if [ $deleted_count -eq 0 ]; then
    echo "Файлы для удаления не найдены"
  else
    echo "Удалено файлов: $deleted_count"
  fi
}

# Функция для показа справки
```

Рис. 4.3: 3

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. 4.4).

```
#!/bin/bash

# Покажем помощь если нужно
if [ "$1" = "--help" ] || [ "$1" = "-h" ] || [ $# -eq 0 ]; then
    echo "=== Архиватор файлов ==="
    echo ""
    echo "Использование:"
    echo "  $0 папка архив.tar.gz      - архивировать все файлы"
    echo "  $0 папка архив.tar.gz recent - только файлы измененные за неделю"
    echo ""
    echo "Примеры:"
    echo "  $0 /home/user/docs backup.tar.gz"
    echo "  $0 ./my_folder backup.tar.gz recent"
    echo "  $0 --help                  - показать эту помощь"
    exit 0
fi

# Проверим количество аргументов
if [ $# -lt 2 ]; then
    echo "Ошибка: Недостаточно аргументов!"
    echo "Используйте: $0 папка архив.tar.gz"
    exit 1
fi

# Запомним аргументы
FOLDER="$1"
ARCHIVE="$2"
MODE="$3"

# Проверим существует ли папка
if [ ! -d "$FOLDER" ]; then
    echo "Ошибка: Папка '$FOLDER' не существует!"
    exit 1
fi

# Проверим права на чтение
if [ ! -r "$FOLDER" ]; then
```

Рис. 4.4: 4

5 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`. 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`. Предположим, необходимо распознать командную строку следующего формата: `testprog -infile_in.txt -ofile_out.doc -L -t -r` Вот как выглядит использование оператора `getopts` в этом случае:

```
while getopts o:i:Ltr optletter do
case
optletter in
o) iflag = 1; oval =OPTARG;;
i) iflag=1; ival=$OPTARG;;
L) Lflag=1;;
t) tflag=1;;
r) rflag=1;;
*) echo Illegal option $optletter
esac
done
```

 Функция `getopts` включает две специальные переменные среды – `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента (будет равно `file_in.txt` для опции `i` и `file_out.doc` для опции `o`). `OPTIND` является числовым индексом на упомянутый аргумент.

Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать ее в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.

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, if test -f mans/i.s проверяет, существует ли файл 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.

6 Выводы

Мы изучили основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.