Лабораторная работа 13

Программирование в командном процессоре ОС UNIX. Ветвления и циклы

Мочалкина Софья Васильевна

Содержание

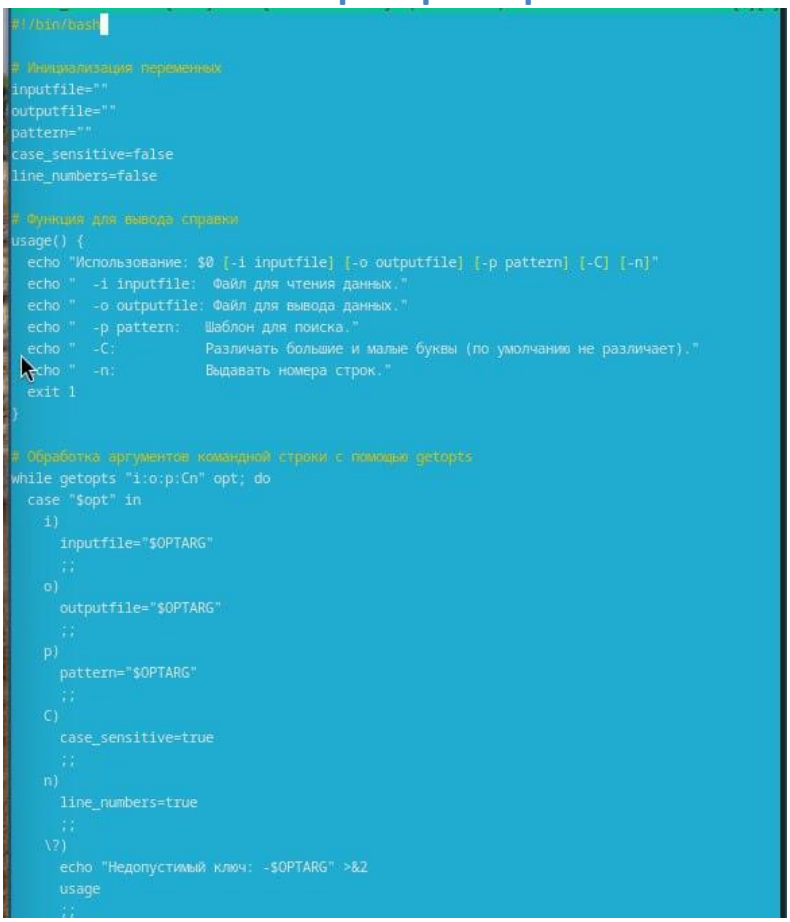
Цель работы

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; – -pшаблон — 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. 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. 3) Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до (например 1.tmp, 2.tmp, 3.tmp,4.tmp и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же ко- мандный 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).

Выполнение лабораторной работы



```
#!/bin/bash

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

# Функция для вывода справки
usage() {
  echo "Использование: $0 [-i inputfile] [-o outputfile] [-p pattern] [-C] [-n]"
  echo "  -i inputfile:  Файл для чтения данных."
  echo "  -o outputfile: Файл для вывода данных."
  echo "  -p pattern:    Шаблон для поиска."
  echo "  -C:            Различать большие и малые буквы (по умолчанию не различает)."
  echo "  -n:            Выдавать номера строк."
  exit 1
}

# Обработка аргументов командной строки с помощью getopts
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
      ;;
    \?)
      echo "Недопустимый ключ: -$OPTARG" >&2
      usage
      ;;
```

рис.1

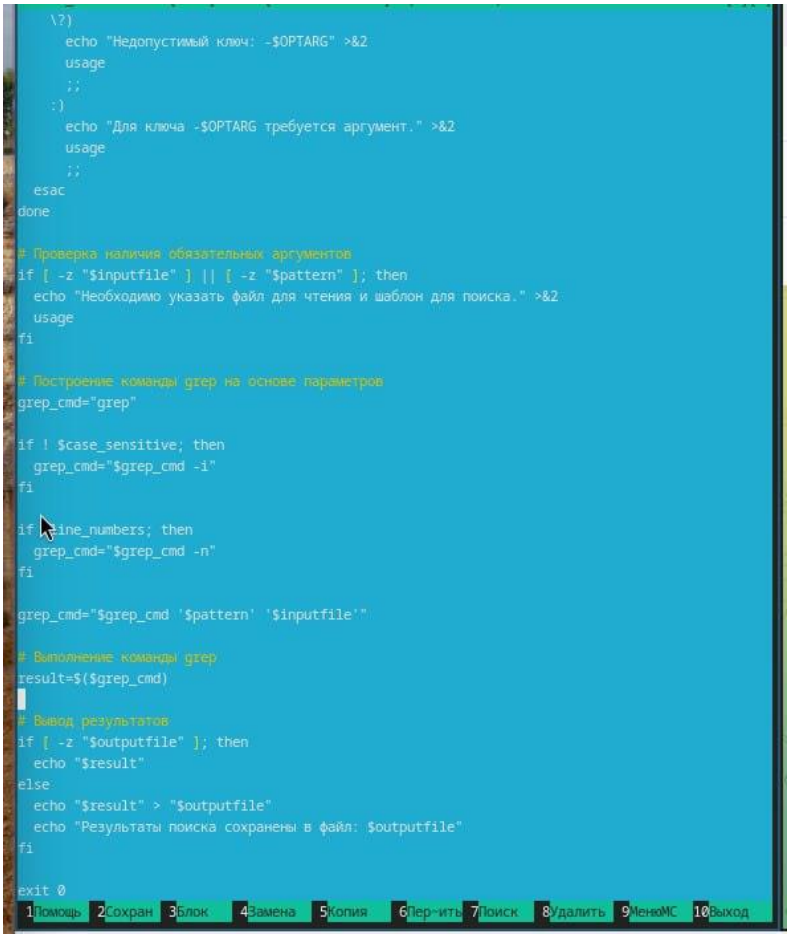

\?)
echo "Недопустимый ключ: -$OPTARG" >&2
usage
;;
:)
echo "Для ключа -$OPTARG требуется аргумент." >&2
usage
;;
esac
done
# Проверка наличия обязательных аргументов
if [-z "$inputfile"] || [-z "$pattern"]; then
echo "Необходимо указать файл для чтения и шаблон для поиска." >&2
usage
fi
# Построение команды grep на основе параметров
grep_cmd="grep"
if ! $case_sensitive; then
grep_cmd="$grep_cmd -i"
fi
if $line_numbers; then
grep_cmd="$grep_cmd -n"
fi
grep_cmd="$grep_cmd '$pattern' '$inputfile'"
# Выполнение команды grep
result=$($grep_cmd)
# Вывод результатов
if [-z "$outputfile"]; then
echo "$result"
else
echo "$result" > "$outputfile"
echo "Результаты поиска сохранены в файл: $outputfile"
fi
exit 0
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер~ить 7Поиск 8Удалить 9МенюMC 10Выход

рис.2

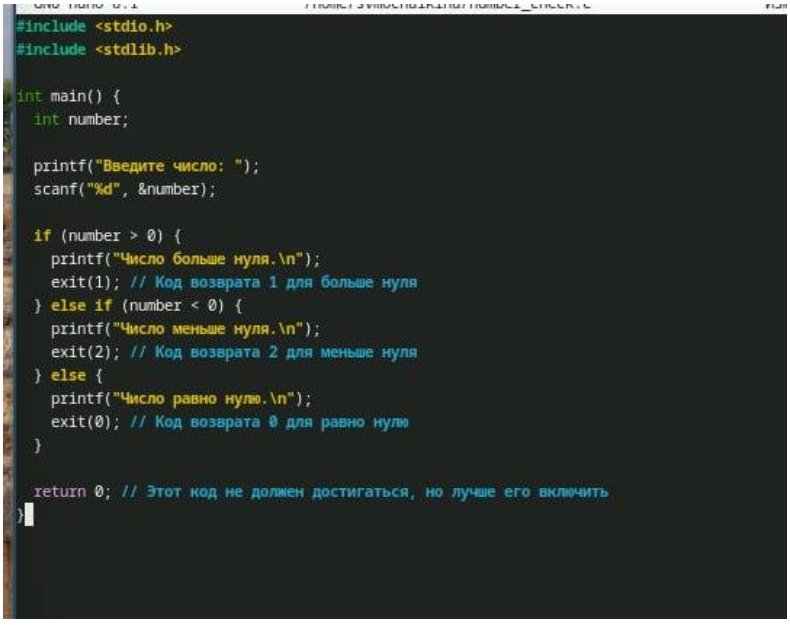
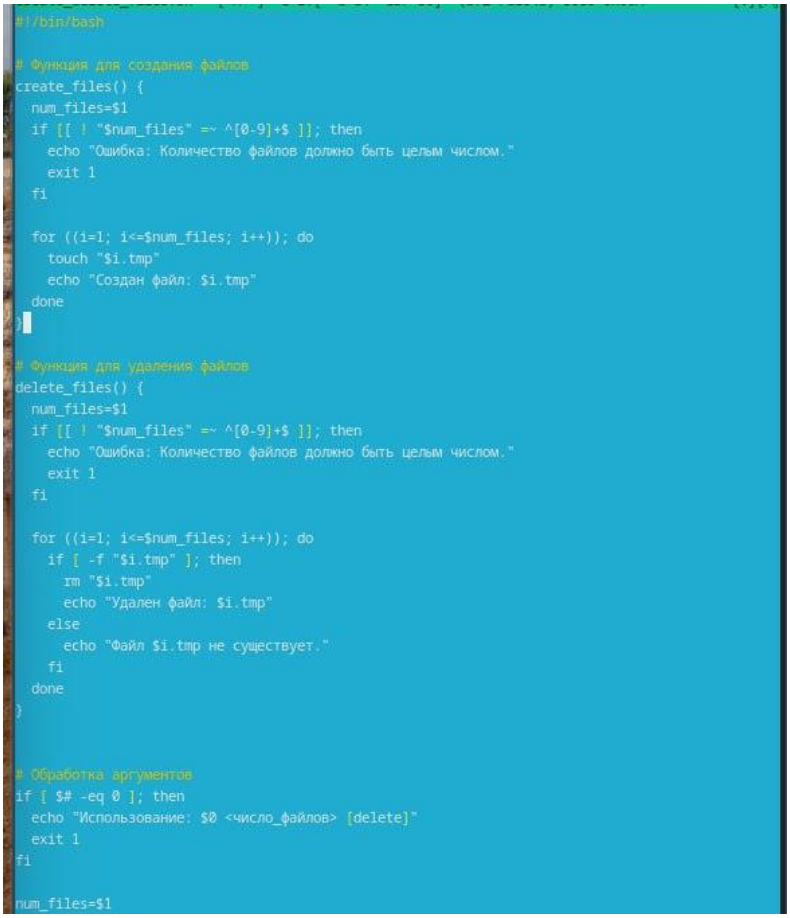
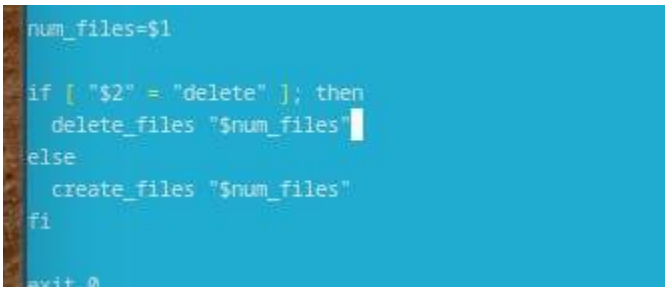

#include <stdio.h>
#include <stdlib.h>
int main() {
int number;
printf("Введите число: ");
scanf("%d", &number);
if (number > 0) {
printf("Число больше нуля.\n");
exit(1); // Код возврата 1 для больше нуля
} else if (number < 0) {
printf("Число меньше нуля.\n");
exit(2); // Код возврата 2 для меньше нуля
} else {
printf("Число равно нулю.\n");
exit(0); // Код возврата 0 для равно нулю
}
return 0; // Этот код не должен достигаться, но лучше его включить
}

рис.3



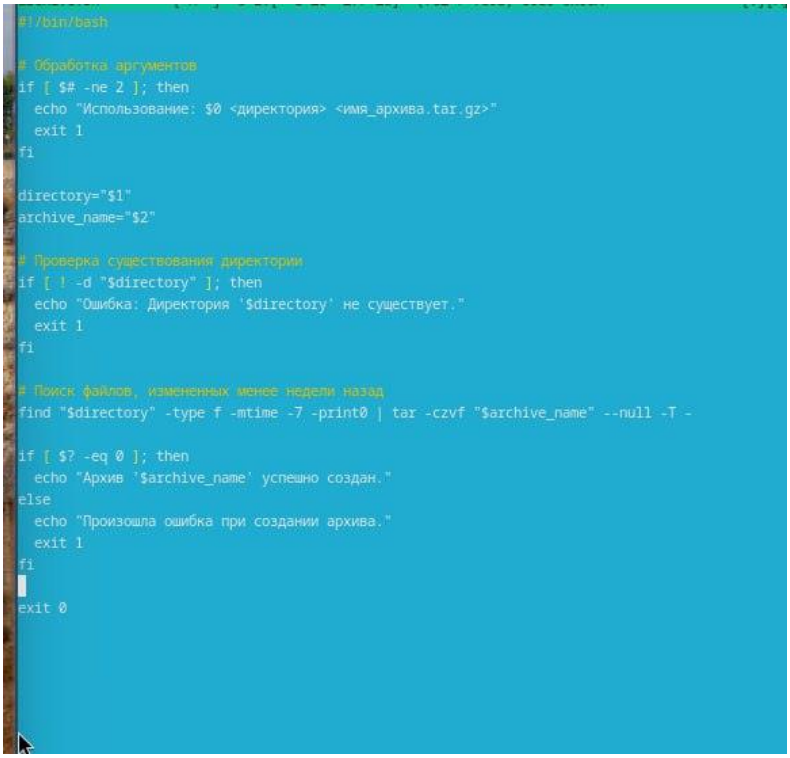
```
#!/bin/bash

# Функция для создания файлов
create_files() {
  num_files=$1
  if [[ ! "$num_files" =~ ^[0-9]+$ ]]; then
    echo "Ошибка: Количество файлов должно быть целым числом."
    exit 1
  fi

  for ((i=1; i<=$num_files; i++)); do
    touch "$i.tmp"
    echo "Создан файл: $i.tmp"
  done
}

# Функция для удаления файлов
delete_files() {
  num_files=$1
  if [[ ! "$num_files" =~ ^[0-9]+$ ]]; then
    echo "Ошибка: Количество файлов должно быть целым числом."
    exit 1
  fi

  for ((i=1; i<=$num_files; i++)); do
    if [ -f "$i.tmp" ]; then
      rm "$i.tmp"
      echo "Удален файл: $i.tmp"
    else
      echo "Файл $i.tmp не существует."
    fi
  done
}


# Обработка аргументов
if [ $# -eq 0 ]; then
  echo "Использование: $0 <число_файлов> [delete]"
  exit 1
fi

num_files=$1
```

рис.4



```
num_files=$1

if [ "$2" = "delete" ]; then
  delete_files "$num_files"
else
  create_files "$num_files"
fi

exit 0
```

рис.5



```
#!/bin/bash

# Обработка аргументов
if [ $# -ne 2 ]; then
  echo "Использование: $0 <директория> <имя_архива.tar.gz>"
  exit 1
fi

directory="$1"
archive_name="$2"

# Проверка существования директории
if [ ! -d "$directory" ]; then
  echo "Ошибка: Директория '$directory' не существует."
  exit 1
fi

# Поиск файлов, измененных менее недели назад
find "$directory" -type f -mtime -7 -print0 | tar -czvf "$archive_name" --null -T -

if [ $? -eq 0 ]; then
  echo "Архив '$archive_name' успешно создан."
else
  echo "Произошла ошибка при создании архива."
  exit 1
fi

exit 0
```

рис.6

Контрольные вопросы:

Ответы на вопросы: Предназначение команды getopts:

Команда getopts предназначена для разбора аргументов командной строки в shell-скриптах (bash, sh и т.д.). Она позволяет удобно обрабатывать параметры и опции, передаваемые скрипту при его запуске. getopts особенно полезна для разбора опций, начинающихся с дефиса (-) или двойного дефиса (–), а также для обработки аргументов, требуемых этими опциями. Она упрощает написание скриптов, принимающих различные настройки через командную строку.

Отношение метасимволов к генерации имён файлов (globbing):

Метасимволы (также известные как подстановочные знаки или шаблоны) используются для генерации имён файлов (globbing) в shell. Они позволяют указывать шаблоны, которые shell раскрывает в список соответствующих файлов. Наиболее распространённые метасимволы:

: Соответствует любому количеству символов (включая отсутствие символов). Например, .txt соответствует всем файлам с расширением .txt. ?: Соответствует одному любому символу. Например, file?.txt соответствует file1.txt, fileA.txt, но не file12.txt. []: Соответствует одному символу из указанного набора. Например, file[1-3].txt соответствует file1.txt, file2.txt, file3.txt. Также можно использовать [!...] или [^...] для соответствия любому символу, не входящему в указанный набор. Когда shell

встречает метасимвол в командной строке, он пытается найти все файлы, соответствующие шаблону, и подставляет эти имена файлов в команду.

Операторы управления действиями (управления потоком выполнения):

В shell-скриптах существует несколько операторов управления действиями:

Условные операторы:

if, then, else, elif, fi: Для выполнения кода в зависимости от условия. case, esac: Для выполнения кода в зависимости от значения переменной (похож на switch в других языках). Циклы:

for: Для перебора элементов списка. while: Для выполнения кода, пока условие истинно. until: Для выполнения кода, пока условие ложно. select: Для создания простого меню для пользователя. Операторы группировки команд:

{ команда1; команда2; ...; }: Группирует команды, выполняемые в текущей оболочке. (команда1; команда2; ...;): Группирует команды, выполняемые в подоболочке. Операторы перенаправления и конвейеры:

, <, >>, |: Для перенаправления ввода/вывода и создания конвейеров (pipelines). Операторы для прерывания цикла:

break: Немедленно завершает выполнение текущего цикла (for, while, until). continue: Прерывает текущую итерацию цикла и переходит к следующей. Предназначение команд false и true:

true: Простейшая команда, которая всегда возвращает код завершения 0 (успех). Часто используется как заглушка или для создания бесконечного цикла (например, while true; do ... done).

false: Простейшая команда, которая всегда возвращает код завершения 1 (неудача). Часто используется для принудительного выхода из условных конструкций или для явного указания ложного условия.

Значение строки if test -f man*s*/i.$s:

Эта строка является частью условной конструкции if. Она проверяет, существует ли файл с определённым именем.

test: Команда test используется для проверки различных условий (существование файла, тип файла, равенство строк, числовые сравнения и т.д.). Альтернативно можно использовать [...] вместо test, но они функционально эквивалентны. -f: Опция команды test, которая проверяет, является ли указанный путь обычным файлом. Если файл существует и является обычным файлом, test возвращает код завершения 0 (истина), иначе - 1 (ложь).
man*s*/i.
s: Предполагаемоеимяфайла, котороестроитсяизпеременных: *man*: Вероятно, константа, указываю
s: Значение переменной s (предположительно, секция документации). $i: Значение

переменной i (предположительно, имя документации). /: Разделитель директорий. .: Разделитель имени файла и расширения. Таким образом, строка проверяет, существует ли файл man<значение $s>/<значение $i>.<значение $s>. Например, если s равно 1 и i равно ls, то будет проверено существование файла man1/ls.1. Различия между конструкциями while и until: Обе конструкции используются для организации циклов, но отличаются условием продолжения выполнения:

while: Цикл while выполняет команды до тех пор, пока условие истинно (код завершения 0). Как только условие становится ложным (код завершения не 0), цикл завершается.

until: Цикл until выполняет команды до тех пор, пока условие ложно (код завершения не 0). Как только условие становится истинным (код завершения 0), цикл завершается.

Другими словами, until является логическим отрицанием while. Оба цикла позволяют выполнять один и тот же код, но иногда использование until делает код более читаемым и понятным, особенно если условие остановки цикла логичнее выразить как “пока не…” вместо “пока…”.

Выводы

Я изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.