

Лабораторная работа №13

Программирование в командном процессоре ОС UNIX. Ветвления и циклы

Пакавира Арсениу

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	14
6	Контрольные вопросы	15

Список литературы	19
-------------------	----

Список иллюстраций

4.1 Первая программа.....	Erro! Indicador não definido.
4.2 Вызов программы в терминале.....	7
4.3 Результат	7
4.4 Результат	7
4.5 Вторая программа.....	8
4.6 Вторая программа.....	8
4.7 Результат	8
4.8 Третья программа	9
4.9 Результат	9
4.10 Четвертая программа	10
4.11 Вызов программы в терминале.....	10
4.12 Результат	10

1 Цель работы

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;
 - `-ршаблон` — указать шаблон для поиска;
 - `-С` — различать большие и малые буквы;
 - `-п` — выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом `-р`.
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.Модифицироватьего так,чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду `find`).

3 Теоретическое введение

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;

- C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;
- оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C 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. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

[Prog: bash?]

4 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;
- `-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`. (рис. 4.1, 4.2, 4.3, 4.4)

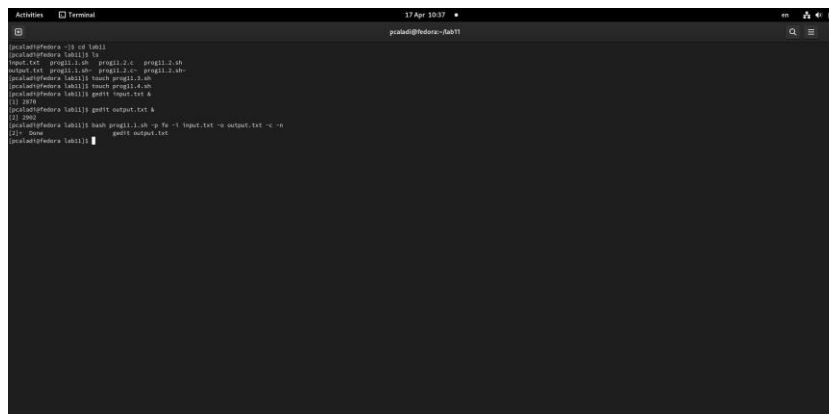


```
#!/bin/bash

while getopts "i: o: r: C: n:" opt; do
    case $opt in
        i) iflag=$OPTARG ;;
        o) oflag=$OPTARG ;;
        r) rflag=$OPTARG ;;
        C) cflag=$OPTARG ;;
        n) nflag=$OPTARG ;;
        *) echo "illegal option $opt" && exit 1 ;;
    esac
done

if [ -n "$iflag" ]; then
    if [ -f "$iflag" ]; then
        if [ -n "$rflag" ]; then
            if [ -n "$cflag" ]; then
                if [ -n "$nflag" ]; then
                    grep -C "$rflag" "$iflag" > "$oflag"
                else
                    grep -C "$rflag" "$iflag" > "$oflag"
                fi
            else
                if [ -n "$nflag" ]; then
                    grep -C "$rflag" "$iflag" > "$oflag"
                else
                    grep -C "$rflag" "$iflag" > "$oflag"
                fi
            fi
        else
            if [ -n "$nflag" ]; then
                grep -C "$rflag" "$iflag" > "$oflag"
            else
                grep -C "$rflag" "$iflag" > "$oflag"
            fi
        fi
    else
        echo "file $iflag does not exist" && exit 1
    fi
else
    echo "no input file specified" && exit 1
fi
```

Рис. 4.1: Первая программа



```
pcasali@fedora:~/lab11$ ls
pcasali@fedora:~/lab11$ ls -l
total 4
-rw-r--r-- 1 pcasali pcasali 1024 Apr 17 10:37 prog1.c
pcasali@fedora:~/lab11$ gcc -o prog1 prog1.c
pcasali@fedora:~/lab11$ ./prog1
10
10 is greater than 0
0
0 is equal to 0
-5
-5 is less than 0
pcasali@fedora:~/lab11$
```

Рис. 4.2: Вызов программы в терминале

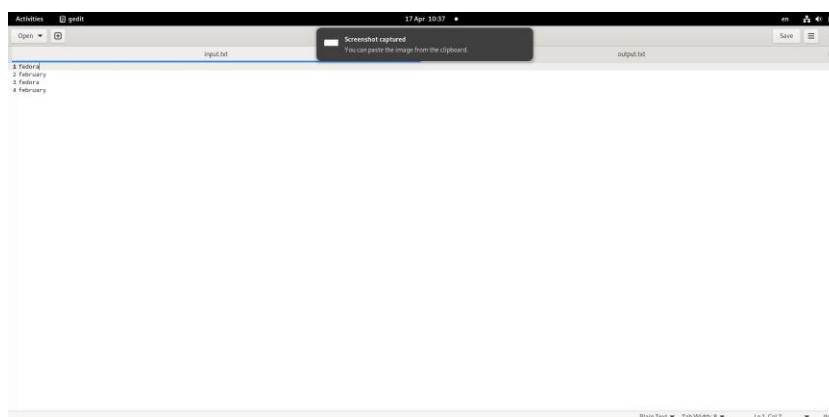


Рис. 4.3: Результат

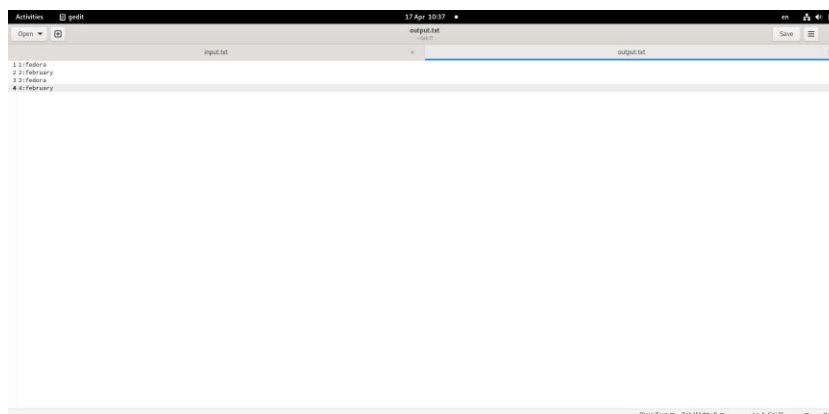


Рис. 4.4: Результат

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 отом,какое число было введено. (рис. 4.5, 4.6, 4.7)

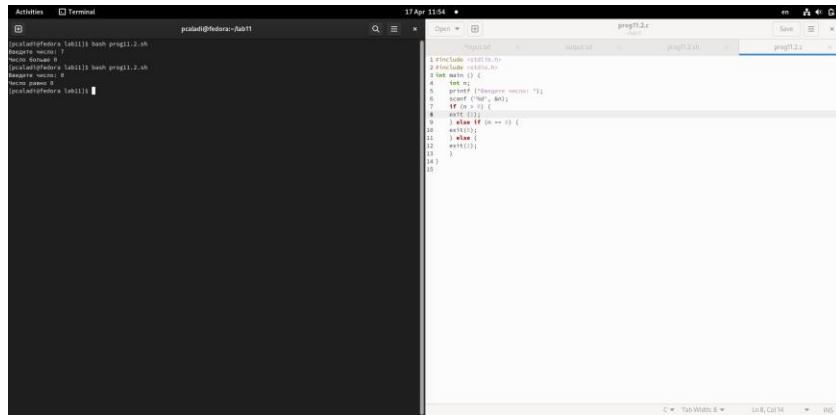


Рис. 4.5: Вторая программа

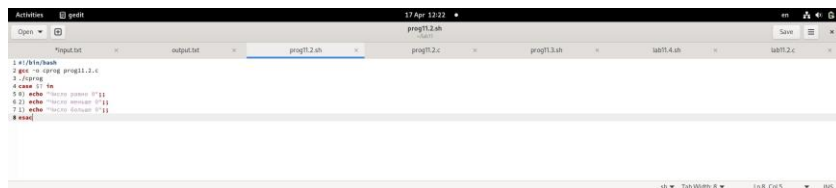


Рис. 4.6: Вторая программа

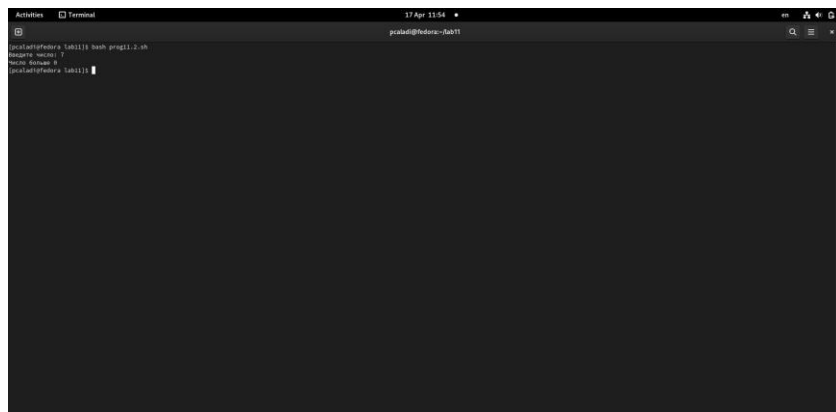


Рис. 4.7: Результат

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. 4.8, 4.9)

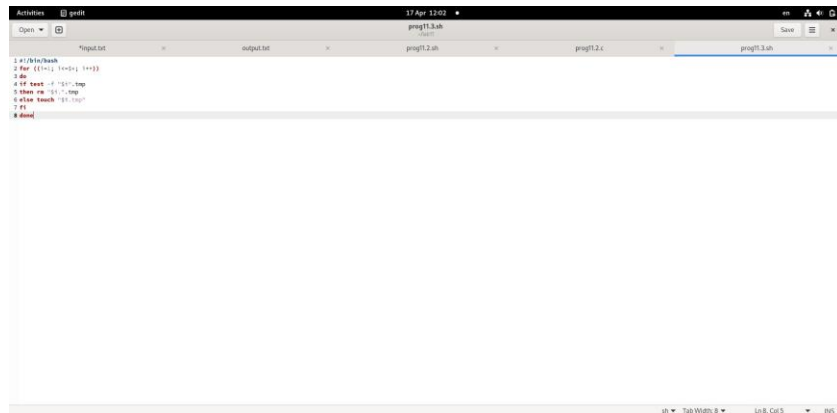


Рис. 4.8: Третья программа

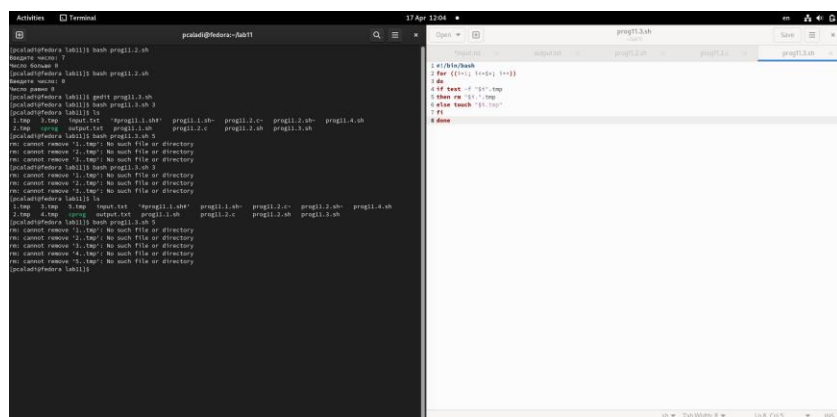


Рис. 4.9: Результат

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. 4.10, 4.11, 4.12)

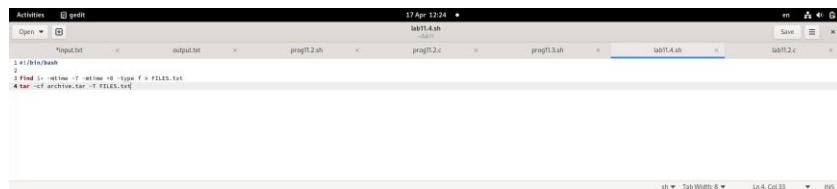


Рис. 4.10: Четвертая программа

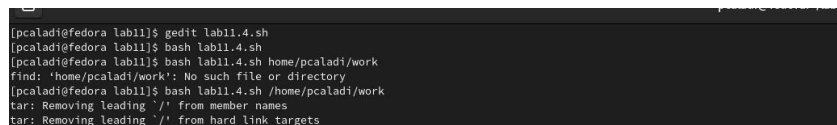


Рис. 4.11: Вызов программы в терминале



Рис. 4.12: Результат

5 Выводы

В процессе выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

6 Контрольные вопросы

1. Каково предназначение команды getopt?

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`. Предположим, необходимо распознать командную строку следующего формата: `testprog -ifile_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 равна `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`, встреченная в командном файле?

Строка `if test -f manS/i.s,manS/i.s` 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.

Список литературы