**Лабораторная работа №1**

**Знакомство с командной строкой Linux и bash-скриптами.**

**Цель лабораторной работы**: ознакомиться с основные команды ОС Linux, научиться писать скрипты и запускать с командной строки, а также изучить утилиту make и запускать компиляцию кода на С++.

1. **Теоретическая часть**

Операционная система Linux создана на основе ОС Unix и во многом имеет схожую структуру и систему команд. Пользователь может работать в текстовом режиме с помощью командной строки (оболочки). Оболочка является одной из важнейших частей системы Unix. *Оболочка* – это программа, запускающая команды (например, те, которые вводит пользователь). Оболочка выступает также в роли небольшой среды программирования. Программисты, работающие в Unix, часто разбивают обычные задачи на несколько небольших компонентов, а затем используют оболочку, чтобы управлять задачами и собирать части воедино.

Многие важные части системы в действительности являются *сценариями оболочки* – текстовыми файлами, которые содержат последовательность команд оболочки. Если вам до этого приходилось работать в системе Windows, то вы можете представлять сценарии оболочки как очень мощные файлы .BAT.

Одно из свойств оболочки позволяет, если вы совершили ошибку, сразу же увидеть, что именно набрано неверно, и попробовать заново.

Существуют различные варианты оболочки Unix, но все они заимствуют некоторые функции от оболочки Bourne shell (/bin/sh) – стандартной оболочки, разработанной в компании Bell Labs для ранних версий системы Unix.

Каждой системе Unix для правильной работы необходимо наличие оболочки Bourne shell. Система Linux использует улучшенную версию оболочки Bourne shell – bash, или «заново рожденную» оболочку. Оболочка bash является оболочкой по умолчанию в большинстве дистрибутивов Linux, а путь /bin/sh, как правило, – ссылка на эту оболочку. При запуске примеров следует применять оболочку bash.

После входа в систему откройте окно оболочки, которое часто называют *терминалом*. Проще всего это выполнить с помощью запуска терминального приложения из графического интерфейса пользователя Gnome или Unity в Ubuntu или сочетанием клавишей ***Ctrl+T***. При этом оболочка открывается в новом окне. После запуска оболочки в верхней части окна должно отобразиться приглашение, которое обычно заканчивается символом доллара ($). В Ubuntu это приглашение будет выглядеть примерно так: *name@host:path$*. Если вы хорошо знакомы с Windows, окно оболочки будет выглядеть подобно окну командной строки DOS; приложение Terminal в OS X, по сути, такое же, как окно оболочки Linux.

Чтобы обозначить приглашение, все они начинаются с символа $. Многие команды начинаются символом #. Такие команды следует запускать с правами пользователя superuser (root). Как правило, с этими командами следует обращаться осторожно.

Рассмотрим основные команды системы и их параметры.

* 1. Команда cat

Команда cat системы Unix является одной из простейших для понимания. Она просто выводит содержимое одного или нескольких файлов. Общий синтаксис команды cat выглядит следующим образом:

$ **cat** *file1 file2* ...

После запуска команда cat выполняет вывод содержимого файлов *file1*, *file2* и каких-либо еще (они обозначены символом ...), а затем завершает работу. Эта команда названа так, поскольку она выполняет конкатенацию (сцепление) содержимого файлов, если выводится более одного файла.

* 1. Команда ls

Команда ls выводит перечень содержимого какого-либо каталога. По умолчанию это текущий каталог. Используйте вариант ls -l, чтобы получить детализированный (длинный) список, или ls –F, чтобы отобразить информацию о типах файлов.

Ниже приведен пример длинного перечня, он содержит информацию о владельце файла (столбец 3), группе (столбец 4), размере файла (столбец 5), а также о дате и времени его изменения (между столбцом 5 и названием файла):

$ **ls -l**

total 3616

-rw-r--r-- 1 juser users 3804 Apr 30 2011 abusive.c

-rw-r--r-- 1 juser users 4165 May 26 2010 battery.zip

-rw-r--r-- 1 juser users 131219 Oct 26 2012 beav\_1.40-13.tar.gz

-rw-r--r-- 1 juser users 6255 May 30 2010 country.c

drwxr-xr-x 2 juser users 4096 Jul 17 20:00 cs335

-rwxr-xr-x 1 juser users 7108 Feb 2 2011 dhry

-rw-r--r-- 1 juser users 11309 Oct 20 2010 dhry.c

-rw-r--r-- 1 juser users 56 Oct 6 2012 doit

drwxr-xr-x 6 juser users 4096 Feb 20 13:51 dw

drwxr-xr-x 3 juser users 4096 May 2 2011 hough-stuff

* 1. Команда cp

В своей простейшей форме команда cp копирует файлы. Например, чтобы скопировать *file1* в файл *file2*, введите следующее:

$ **cp** *file1 file2*

Чтобы скопировать несколько файлов в какой-либо каталог (папку) с названием *dir*, попробуйте такой вариант:

$ **cp** *file1* **...** *fileN dir*

* 1. Команда mv

Команда mv (от англ. move – «переместить») подобна команде cp. В своей простейшей форме она переименовывает файл. Например, чтобы переименовать файл *file1* в *file2*, введите следующее:

$ **mv** *file1 file2*

Можно также использовать команду mv, чтобы переместить несколько файлов в другой каталог:

$ **mv** *file1* **...** *fileN dir*

* 1. Команда touch

Команда touch создает файл. Если такой файл уже существует, команда touch не изменяет его, но обновляет информацию о времени изменения файла, выводимую с помощью команды ls -l. Например, чтобы создать пустой файл, введите следующее:

$ **touch** *file*

Теперь примените к этому файлу команду ls -l. Вы должны увидеть результат, подобный приведенному ниже. Символом ➊ отмечены дата и время запуска команды touch:

$ **ls -l** *file*

-rw-r--r-- 1 juser users 0 May 21 18:32➊ *file*

* 1. Команда rm

Чтобы удалить файл, воспользуйтесь командой rm (от англ. remove – «удалить»). После удаления файла он исчезает из системы и, как правило, не может быть восстановлен.

$ **rm** *file*

* 1. Команда cd

*Текущий рабочий каталог* – это каталог, в котором в данный момент находится процесс (например, оболочка). Команда cd изменяет текущий рабочий каталог оболочки:

$ **cd** *dir*

Если вы опустите параметр *dir*, оболочка вернет вас в *домашний каталог*, с которого вы начали работу при входе в систему.

* 1. Команда mkdir

Команда mkdir создает новый каталог с именем *dir*:

$ **mkdir** *dir*

* 1. Команда rmdir

Команда rmdir удаляет каталог с именем *dir*:

$ **rmdir** *dir*

Если каталог *dir* не пуст, эта команда не сработает. Чтобы удалить каталог со всем его содержимым, используйте команду rm -rf *dir*. Однако будьте осторожны! Это одна из немногих команд, которая может причинить существенный вред, особенно если вы работаете как superuser. Параметр –r задает *рекурсивное удаление*, которое последовательно удаляет все, что находится внутри каталога *dir*, а параметр -f делает эту операцию принудительной. Не используйте флаги -rf с такими джокерными символами, как звездочка (\*). Перепроверяйте команды перед их запуском.

* 1. Джокерные символы

Оболочка способна сопоставлять простые шаблоны с именами файлов и каталогов. Этот процесс называется *универсализацией файловых имен*. Он напоминает применение джокерных символов в других системах. Простейшим из таких символов является звездочка (\*), которая указывает оболочке на то, что вместо него можно подставить любое количество произвольных символов. Например, следующая команда выдаст список файлов в текущем каталоге:

$ **echo \***

Оболочка сопоставляет аргументы, которые содержат джокерные символы, с именами файлов, заменяет аргументы подходящими именами, а затем запускает исправленную командную строку. Такая подстановка называется *развертыванием*, поскольку оболочка подставляет все подходящие имена файлов. Вот несколько примеров того, как можно использовать символ \* для развертывания имен файлов:

* at\* – развертывается во все имена файлов, которые начинаются с символов at;
* \*at – развертывается во все имена файлов, которые заканчиваются символами at;
* \*at\* – развертывается во все имена файлов, которые содержат символы at.

Если ни один из файлов не удовлетворяет шаблону, оболочка не выполняет развертывание и вместо него буквально использует указанные символы, в том числе и \*. Попробуйте, например, набрать такую команду, как echo \*dfkdsafh.

ПРИМЕЧАНИЕ

Если вы привыкли к работе в системе MS-DOS, то вы могли бы инстинктивно набрать символы \*.\*, которые подходят для всех файлов. Расстаньтесь теперь с этой привычкой. В системе Linux и других версиях системы Unix вы должны использовать символ \*. В оболочке Unix комбинация \*.\* соответствует только тем файлам и каталогам, названия которых содержат точку. Для названий файлов в системе Unix расширения не являются обязательными, и файлы часто обходятся без них.

Еще один джокерный символ, вопросительный знак (?), указывает оболочке на то, что необходимо подставить только один произвольный символ. Например, комбинации b?at будут соответствовать имена boat и brat.

Если вы желаете, чтобы оболочка не развертывала джокерный символ в команде, заключите его в одиночные кавычки (' '). Так, например, команда '\*' выдаст символ звездочки. Вы увидите, что это удобно применять в некоторых командах, например grep и find (они рассматриваются в следующем разделе, подробности об использовании кавычек вы узнаете из раздела 11.2).

ПРИМЕЧАНИЕ

Важно помнить о том, что оболочка выполняет развертывание перед запуском команд и только в это время. Следовательно, если символ \* приводит к отсутствию развертывания, оболочка ничего не будет с ним делать; и уже от команды зависит решение о дальнейших действиях.

В современных версиях оболочки существуют дополнительные возможности настройки шаблонов, однако сейчас вам необходимо знать лишь джокерные символы \* и ?.

* 1. Команда grep

Команда grep выдает строки из файла или входного потока, которые соответствуют какому-либо выражению. Например, чтобы напечатать строки из файла /etc/passwd, которые содержат текст root, введите следующее:

$ **grep root /etc/passwd**

Команда grep чрезвычайно удобна, когда приходится работать одновременно с множеством файлов, поскольку она выдает название файла в дополнение к найденной строке. Например, если вы желаете отыскать все файлы в каталоге /etc, которые содержат слово root, можно применить данную команду так:

$ **grep root /etc/\***

Двумя наиболее важными параметрами команды grep являются –i (для соответствий, нечувствительных к регистру символов) и –v (который инвертирует условие поиска, то есть выдает все строки, не отвечающие условию). Существует также более мощный вариант команды под названием egrep (это всего лишь синоним команды grep -E).

Команда grep понимает шаблоны, которые известны как *регулярные выражения*. Они укоренились в теории вычислительной техники и являются весьма обычными для утилит системы Unix. Регулярные выражения более мощные, чем шаблоны с джокерными символами, и имеют другой синтаксис. Следует помнить два важных момента, относящихся к регулярным выражениям:

* сочетание .\* соответствует любому количеству символов (подобно джокерному символу \*);
* символ . соответствует одному произвольному символу.

ПРИМЕЧАНИЕ

Страница руководства grep(1) содержит подробное описание регулярных выражений, но оно может оказаться трудным для восприятия. Чтобы узнать больше, можете прочитать книгу Mastering Regular Expressions («Осваиваем регулярные выражения»), 3-е издание (O’Reilly, 2006) или главу о регулярных выражениях в книге Programming Perl («Программирование на языке Perl»), 4-е издание (O’Reilly, 2012). Если вы любите математику и вам интересно, откуда возникли регулярные выражения, загляните в книгу Automata Theory, Languages and Computation («Теория автоматов, языки и вычисления»), 3-е издание (Prentice Hall, 2006).

* 1. Команда less

Команда less становится удобной тогда, когда файл довольно большой или когда выводимый результат длинен и простирается за пределы экрана.

Чтобы постранично просмотреть такой большой файл, как /usr/share/dict/words, воспользуйтесь командой less /usr/share/dict/words. После запуска команды less вы увидите содержимое файла, разбитое на части и умещающееся в пределах одного экрана. Нажмите клавишу Пробел, чтобы переместиться далее по содержимому, или клавишу B, чтобы вернуться назад на один экран. Чтобы выйти, нажмите клавишу Q.

ПРИМЕЧАНИЕ

Команда less является улучшенной версией устаревшей команды more. Большинство рабочих станций и серверов на основе системы Linux содержат команду less, однако она не является стандартной для многих встроенных и других систем на основе Unix. Если вы когда-либо столкнетесь с невозможностью использовать команду less, попробуйте применить команду more.

Можно также воспользоваться поиском текста внутри команды less. Например, чтобы отыскать слово *word*, наберите /word. Для поиска в обратном направлении применяйте вариант ? word. Когда результат будет найден, нажмите клавишу N для продолжения поиска.

Как вы узнаете из раздела 2.14, можно отправить стандартный вывод практически из любой команды непосредственно на стандартный вход другой команды. Это исключительно полезно, если команда выводит большое количество результатов, которые вам пришлось бы просеивать с использованием какой-либо команды вроде less. Вот пример отправки результатов команды grep в команду less:

$ **grep ie /usr/share/dict/words | less**

Попробуйте самостоятельно поработать с этой командой. Возможно, вы станете часто применять команду less подобным образом.

* 1. Команда find и locate

Бывает досадно, если вы знаете, что какой-либо файл точно расположен где-то в данном дереве каталогов, но вы не помните точно, где именно. Запустите команду find, чтобы отыскать файл *file* в каталоге *dir*:

$ **find** *dir* **-name** *file* **-print**

Подобно большинству команд в этом разделе, команда find обладает некоторыми интересными особенностями. Однако не пробуйте применять параметры, описанные здесь как -exec, пока не выучите наизусть их форму и станете понимать, зачем нужны параметры -name и -print. Команда find допускает применение специальных шаблонных символов вроде \*, но вы должны заключать такие символы в одиночные кавычки ('\*'), чтобы оградить их от функции универсализации, которая действует в оболочке.

В большинстве систем есть также команда locate для поиска файлов. Вместо отыскивания файла в реальном времени эта команда осуществляет поиск в индексе файлов, который система периодически создает. Поиск с помощью команды locate происходит гораздо быстрее, чем с помощью find, но если искомый файл появился после создания индекса, команда locate не сможет его найти.

* 1. Команда head и tail

Чтобы быстро просмотреть фрагмент файла или потока данных, используйте команды head и tail. Например, команда head /etc/passwd отобразит первые десять строк файла с паролем, а команда tail /etc/passwd покажет заключительные десять строк.

Чтобы изменить количество отображаемых строк, применяйте параметр -*n*, в котором число *n* равно количеству строк, которые необходимо увидеть (например, head -5 /etc/passwd). Чтобы вывести строки, начиная со строки под номером *n*, используйте команду tail +*n*.

1. **Введение в сценарий оболочки**

**Bash** - это sh-совместимый интерпретатор командного языка, выполняющий команды, прочитанные со стандартного входного потока или из файла.

**Скрипт-файл** – это обычный текстовый файл, содержащий последовательность команд bash, для которого установлены права на выполнение. Пример скрипта, выводящего содержимое текущего каталога на консоль и в файл:

#!/bin/bash

dir

dir > 1.txt

Любой bash-скрипт должен начинаться со строки:

#!/bin/bash

в этой строке после #! указывается путь к bash-интерпретатору, поэтому если он у вас установлен в другом месте, поменяйте её на ваш путь. Комментарии начинаются с символа # (кроме первой строки).

**2.1 Переменные**

Следующие переменные используются командным интерпретатором.

* $0, $1, $2, $3... - значения аргументов командной строки при запуске скрипта. Где $0 - имя самого файла скрипта, $1 - первый аргумент, $2 - второй аргумент, и т. д.
* $@ - все аргументы командной строки, каждый в кавычках.
* $# - количество аргументов командной строки.
* $? - код возврата последней команды.

Пример простого скрипта, выводящего на консоль и в файл содержимое каталога, где имя каталога передаётся скрипту в качестве аргументов при запуске:

#!/bin/bash

dir $1

dir $1 > 1.txt

Запуск скрипта: ./script.sh ~/test\_catalog

Можно создать собственную переменную и присвоить ей значение:

A=121

A="121"

let A=121

let "A=А+1"

Вывод значения на консоль:

echo $A

Пример скрипта с переменными:

#!/bin/bash

# указываем где у нас хранится bash-интерпретатор

parametr1=$1

# присваиваем переменной parametr1 значение первого параметра скрипта

script\_name=$0

# присваиваем переменной script\_name значение имени скрипта

echo "Вы запустили скрипт с именем $script\_name и параметром $parametr1"

# команда echo выводит определенную строку, обращение к переменным осуществляется через $имя\_переменной.

echo 'Вы запустили скрипт с именем $script\_name и параметром $parametr1'

# здесь мы видим другие кавычки. Разница в том, что в одинарных кавычках не происходит подстановки переменных.

exit 0

# выход с кодом 0 (удачное завершение работы скрипта)

**2.2 Условия**

Структура **if-then-else** используется следующим образом:

if <команда или набор команд, возвращающих код возврата (0 или 1)>

then

<если выражение после if истинно, то выполняется этот блок>

else

<если выражение после if ложно, то этот>

В качестве команд, возвращающих код возврата, могут выступать структуры [[ ]], [ ], test, (( )) или любая другая linux-команда.

test — используется для логического сравнения.

[ ] — синоним команды test

[[ ]] — расширенная версия [ ], внутри которой могут быть использованы || (или), & (и).

(( )) — математическое сравнение.

Для построения многоуровневых условий вида:  
if ... then ... else if ... then ... else ...  
для краткости и читаемости кода, можно использовать структуру  
if ... then ... elif ... then ... elif ....

Если необходимо сравнивать одну переменную с большим количеством возможных значений, то целесообразней использовать оператор case.

#!/bin/bash

echo "Выберите редатор для запуска:"

echo "1. Запуск программы nano"

echo "2. Запуск программы vi"

echo "3. Запуск программы emacs"

echo "4. Выход"

read doing # читаем в переменную $doing со стандартного ввода

case $doing in

1) /usr/bin/nano # если $doing содержит 1, то запустить nano

;;

2) /usr/bin/vi # если $doing содержит 2, то запустить vi

;;

3) /usr/bin/emacs # если $doing содержит 3, то запустить emacs

;;

4) exit 0

;;

\*) #если введено с клавиатуры то, что в case не описывается, выполнять следующее:

echo "Введено неправильное действие"

esac #окончание оператора case.

Условия **сравнения**:

Файлы:

* + -e - Проверить что файл существует (-f, -d)
  + -f - Файл существует (!-f - не существует)
  + -d - Каталог существует
  + -s - Файл существует и не пустой
  + -r - Файл существует и доступен на чтение
  + -w - Файл существует и доступен на запись
  + -x - Файл существует и доступен на выполнение
  + -h - Символическая ссылка

Строки:

* + -z - Пустая строка
  + -n - Не пустая строка
  + == - Равно (!= - не равно)

Числа

* + -eq - Равно
  + -ne - Не равно
  + -lt - Меньше
  + -le - Меньше или равно
  + -gt - Больше
  + -ge - Больше или равно

**2.3. Циклы**

Оператор for-in предназначен для поочередного обращения к значениям, перечисленным в списке. Каждое значение поочередно в списке присваивается переменной.

Синтаксис оператора, следующий:

for переменная in список\_значений do

команды

done

Пример использования **цикла for**:

#!/bin/bash

for i in 0 1 2 3 4

# переменной $i будем поочередно присваивать значения от 0 до 4 включительно

do

echo "Console number is $i" >> /dev/pts/$i

# Пишем в файл /dev/pts/$i (файл виртуального терминала) строку "Console number is $i"

done

#цикл окончен

exit 0

Обход списка файлов в каталоге, используя **цикл for**:

for f in $HOME/tmp/\*; do

filename=$(basename "$f")

extension=${filename##\*.}

if [ "$filename" == "stop.txt" ]; then

break

fi

if [ $extension != 'png' ]; then continue; fi

echo $f

done

Операторы while и until работает подобно if, только выполнение команд циклически продолжается до тех пор, пока верно условие, и прерывается, если условие не верно. Конструкция его выглядит так:

while [условие]

do

команды

done

Например

i=1

while[$i -le5 ]; do

echo -n $i ‘ ‘; sleep 1

i=$((i+1))

done

Результат выполнения сценария:1 2 3 4 5. Цикл while работает до тех пор, пока команда, указанная в качестве ее аргумента, возвращает успешный код завершения. Наоборот, until работает, пока команда-аргумент заканчивается неудачей.

**2.4. Операции**

Команда let производит арифметические операции над числами и переменными.

Рассмотрим небольшой пример, в котором произведем некоторые вычисления над введенными числами:

#!/bin/bash

echo "Введите a: "

read a

echo "Введите b: "

read b

let "c = a + b" # сложение

echo "a+b= $c"

let "c = a / b" # деление

echo "a/b= $c"

let "c <<= 2" # сдвигает c на 2 разряда влево

echo "c после сдвига на 2 разряда: $c"

let "c = a % b" # находит остаток от деления a на b

echo "$a / $b. остаток: $c "

**2.5. Функции**

**Функции** в bash могут принимать аргументы, возвращать вычисленное значение и позволяют исключить дублирование кода в скриптах.

some\_function() {

# Объявляем переменную $str локальной и читаем в нее стандартный поток ввода

local str

read str

first\_argument="$1"

second\_argument="$2"

# Читаем построчно входной поток

while read line; do

# Возвращаем список строк для последующей обработки

echo -n "${first\_argument} and $second\_argument"

done <<< file.txt

# Вернуть код завершения (0 - при успешном завершении)

# Код ответа доступен после выполнения ф-ции в переменной $?

return 0

}

# Пример вызова функции

echo 'content' | some\_sunction arg1 arg2

# или так

some\_var='второй аргумент'

result=$(some\_sunction 'arg1' "$some\_var")

# или так

result=`some\_sunction`

ret\_code=$? # получить код возврата функции

**2.6. Потоки**

Файл, из которого осуществляется чтение, называется **стандартным потоком ввода**, а в который осуществляется запись – **стандартным потоком вывода**.

Стандартные потоки:

* 0, stdin, ввод;
* 1, stdout, вывод;
* 2, stderr, поток ошибок.

При перенаправлении потоков, вы можете указывать ссылки на определенные потоки. Например, перенаправим вывод и ошибки команды в файл:

command 2>&1 # ошибки (stderr) в stdout

command > ~/out.txt 2>&1 # stdout в файл

command &> ~/out.txt # весь вывод в файл

Для перенаправления потоков используются основные команды: <, >, >>, <<<, |. Рассмотрим, как можно перенаправлять стандартные потоки.

Перенаправление потока вывода:

* > - перенаправить поток вывода в файл (файл будет создан или перезаписан)
* >> - дописать поток вывода в конец файла

Перенаправление потока ввода (прием данных):

* < - файл в поток ввода (файл будет источником данных)
* <<< - чтение данных из строки вместо содержимого файла (для bash 3 и выше)

Перенаправление вывода ошибок:

* 2> - перенаправить поток ошибок в файл
* 2>> - дописать ошибки в файл (файл будет создан или перезаписан)

**2.6. Конвейеры**

**Конвейер** — очень мощный инструмент для работы с консолью Bash. Синтаксис команда1 | команда 2 — означает, что вывод команды 1 передастся на ввод команде 2.

Конвейеры можно группировать в цепочки и выводить с помощью перенаправления в файл, например:

ls -la | grep "hash" | sort > sortilg\_list.txt

Вывод команды ls -la передается команде grep, которая отбирает все строки, в которых встретится слово hash, и передает их команде сортировке sort, которая пишет результат в файл sorting\_list.txt.

1. **Задание**
2. Изучить теоретическую часть лабораторной работы.
3. В консольном режиме создать в домашнем каталоге подкаталог: /номер\_группы/фамилия\_студента, где в дальнейшем будут храниться все рабочие файлы студента.
4. В любом текстовом редакторе (например, vim: vim 1.c) написать программу 1.c, выводящую на экран фразу "HELLO Ubuntu". Компилировать полученную программу компилятором gcc: gcc 1.c –o 1.exe. Запустить полученный файл 1.exe на выполнение: ./1.exe
5. Написать скрипт, выводящий на консоль и в файл все аргументы командной строки.
6. Написать скрипт, выводящий в файл (имя файла задаётся пользователем в качестве первого аргумента командной строки) имена всех файлов с заданным расширением (третий аргумент командной строки) из заданного каталога (имя каталога задаётся пользователем в качестве второго аргумента командной строки).
7. Написать скрипт, компилирующий и запускающий программу (имя исходного файла и exe- файла результата задаётся пользователем в качестве аргументов командной строки). В случае ошибок при компиляции вывести на консоль сообщение об ошибках и не запускать программу на выполнение.
8. Выполнить индивидуальное задание.

**3.1 Варианты индивидуальных заданий:**

1. Написать скрипт для поиска файлов заданного размера в заданном каталоге (имя каталога задаётся пользователем в качестве третьего аргумента командной строки). Диапазон (min - max) размеров файлов задаётся пользователем в качестве первого и второго аргумента командной строки.
2. Написать скрипт с использованием цикла for, выводящий на консоль размеры и права доступа для всех файлов в заданном каталоге и всех его подкаталогах (имя каталога задается пользователем в качестве первого аргумента командной строки).
3. Написать скрипт для поиска заданной пользователем строки во всех файлах заданного каталога и всех его подкаталогах (строка и имя каталога задаются пользователем в качестве первого и второго аргумента командной строки). На консоль выводятся полный путь и имена файлов, в содержимом которых присутствует заданная строка, и их размер. Если к какому-либо каталогу нет доступа, необходимо вывести соответсвующее сообщение и продолжить выполнение.
4. Написать скрипт поиска одинаковых по их содержимому файлов в двух каталогах, например, Dir1 и Dir2. Пользователь задаёт имена Dir1 и Dir2 в качестве первого и второго аргумента командной строки. В результате работы программы файлы, имеющиеся в Dir1, сравниваются с файлами в Dir2 по их содержимому. На экран выводятся число просмотренных файлов и результаты сравнения.
5. Написать скрипт, находящий в заданном каталоге и всех его подкаталогах все файлы, владельцем которых является заданный пользователь. Имя владельца и каталог задаются пользователем в качестве первого и второго аргумента командной строки. Скрипт выводит результаты в файл (третий аргумент командной строки) в виде "полный путь, имя файла, его размер". На консоль выводится общее число просмотренных файлов.
6. Написать скрипт, находящий в заданном каталоге и всех его подкаталогах все файлы заданного размера и принадлежащие определенному пользователю. Диапазон (min - max) размеров файлов задаётся пользователем в качестве первого и второго аргумента командной строки. Имя владельца и каталог задаются пользователем в качестве третьего и четвертого аргумента командной строки. Скрипт выводит результаты поиска в файл (пятый аргумент командной строки) в виде "полный путь, имя файла, его размер". На консоль выводится общее число просмотренных файлов.
7. Написать скрипт, подсчитывающий суммарный размер файлов в заданном каталоге и всех его подкаталогах (имя каталога задаётся пользователем в качестве первого аргумента командной строки). Скрипт выводит результаты подсчета в файл (второй аргумент командной строки) в виде "каталог (полный путь), суммарный размер файлов, число просмотренных файлов".
8. **Контрольные вопросы**
9. Что такое bash?
10. Какой командой можно посмотреть весь файл, не открывая его?
11. Если файл большой, как в командной строке сделать так, чтобы он выводился по частям?
12. Напишите конструкцию условного оператора if в bash.
13. Напишите конструкцию циклы for.
14. **Список используемых источников**
15. <https://habr.com/ru/post/47163/>
16. <http://www.opennet.ru/docs/RUS/bash_scripting_guide/>
17. https://onedev.net/post/254