Государственное бюджетное образовательное учреждение среднего профессионального образования Новосибирской области

Новосибирский химико- 630102 г. Новосибирск, ул. Садовая, 26,

технологический колледж. Тел./факс: (383) 266-00-44, тел.: (383) 266-00-54,

nhtk@mail.ru, http://nhtk-edu.ru

им. Д.И. Менделеева

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ**

**ПРАКТИЧЕСКОЙ РАБОТЫ**

Разработчик:

Соломенный Ю.Ю.

Новосибирск – 2023

**Практическая работа №4 «Анализ оперативной памяти»**

**Теоретический материал**

**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.

Коментарии начинаются с символа # (кроме первой строки).

##### 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)

##### 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>

fi

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

##### 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

##### 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 "

##### Функции

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

some\_sunction() {

# 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

##### 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)

##### 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.

**Порядок выполнения практической работы**

1. Практическая работа выполняется в ***отдельном*** создаваемом студентом ***документе***!
2. Титульный лист с указанием наименования организации, ФИО студента и номер группы
3. Выполнить задания, в отчете написать номер задания, добавить скриншот, описать результат.
4. Ответить на контрольные вопросы
5. Вывод

**Практическое задание**

1. Создайте новый файл с именем "memoryanalysis.sh" в любой директории на вашей Linux системе.  
  
2. Откройте созданный файл с помощью текстового редактора и добавьте следующий скрипт:  
  
#!/bin/bash  
  
# Выведите общую информацию об использовании оперативной памяти  
echo "Использование оперативной памяти:"  
free -h  
  
# Выведите информацию о загрузке памяти в системе  
echo "Загрузка памяти:"  
cat /proc/meminfo | grep "MemAvailable\|Active\|Inactive\|Dirty\|AnonPages\|Mapped\|SwapTotal\|SwapFree"  
  
# Выведите информацию о файле подкачки  
echo "Файл подкачки:"  
cat /proc/swaps  
```  
  
3. Сохраните и закройте файл.  
  
4. Откройте терминал и перейдите в директорию, где находится созданный файл "memoryanalysis.sh".  
  
5. Выполните следующую команду в терминале, чтобы дать разрешение на выполнение скрипта:  
chmod +x memory\_analysis.sh  
  
  
6. Запустите скрипт, выполнив следующую команду:  
./memory\_analysis.sh  
  
  
7. Результаты анализа использования оперативной памяти, загрузки памяти и файла подкачки будут выведены в терминале.

8. Написать выводы о загрузке памяти в вашей системе Linux