COMP 8567

Advanced Systems Programming

Shell Programming

Outline

- Unix Shell- Introduction
- Path and External Commands
- Metacharacters
- Shell Programs-Scripts
- Shell Variables
- Defining a Global Variable
- Quoting
- Bash as a Programming Language
- Accessing variables
- String Expressions
- File Expressions
- Control Structures
  - If statement
  - While statement
  - Repeat until statement
  - For statement
  - Case statement
- The Trap Command
- Summary

Introduction

- Example:
  - A simple shell script (ex0.sh)

Unix Shell-Introduction

- A Unix shell is a command interpreter that starts running as soon as you log In.

- A shell command can be
  - **Internal(built-in)** : code is part of the shell (ex. cd, echo)
  - **External** : code resides in a sperate binary file (ls, cat, gcc etc.)

- The shell terminates when **CTR-D** is entered. //SIGQUIT

Path and External Commands

- For an external command, the shell searches for its file in the directories whose names are stored in the shell variable **PATH**.
- PATH is an environment variable which tells the shell which directory to search for (an executable) associated with a command
- How to define the shell variable PATH?
- Example : PATH = ./usr/bin
- PATH can be modifed, examples :
  - PATH=$PATH:/new/bin/
  - PATH=/usr/local/bin:$PATH
- **$ which** command finds the path of the files associated with other commands.

  Examples: which bash, which gcc
- **$ help** command lists all the internal commands
- $ **type cat**, $ **type cd** //Returns the type of commnd
- Note: PATH is initially set in shell start-up files

  like /etc/profile

Metacharacters

- These are special characters with special meanings :
- > Output redirection
  - E.g., ls > filleNames.txt
- < Input redirection
  - E.g., mail –s “Subject” user@uwindsor.ca < letter.txt
- >> Output redirection, appends to a file
  - E.g., ls >> filleNames.txt
- * filename Wild card, matches 0 or more characters
  - E.g., rm *ps, deletes all files ending with 'ps'.
- ? filename Wild card, matches 1 character.
  - E.g., rm *.? delete files with one character after '.'
  - ls ?? lists files/directories made up of 2 characters.

Metacharacters ..

- `command` (backticks) : command substitution, replaced by the command output.
- E.g. 1. echo The date is `date`
- E.g. 2. echo The directory listing is ls //Output: hello ls
- echo The directory listing is `ls` //Output: hello followed by the ls outputs.
- | Pipe between two commands.

  E.g., ls | wc -w //output of ls is piped to wc to get the number of files/directories.

  Note the utility wc displays a count of lines, words and characters (depending on the parameter)
- ; Used to sequence commands
  - E.g., date ; ls; date
- || : Executes a command if the previous one fails.
- E.g., cc prog1.c./hello || CC prog1.c || ./hello

Metacharacters..

- && : Executes a command if the previous one succeeds
  - E.g., ./hello && ./hello1
- #: characters after this are ignored until the end of the line
  - (comment)
- $: Expands the value of a shell variable
  - E.g., echo $PATH
- \ : Prevents special interpretation of next character.
- E.g., echo \$PATH ($ will not longer be a special character and output:$PATH)

Shell Programs: Scripts

Shells are more than command interpreters, they have their own programming languages.

A shell script, is a file that contains shell commands.

A shell language has the ability to:

- **To define, read and write shell variables**
- **Utilize control structures such as loop and if statements**

```
//Creating and running a sample script : sample.sh
//sha-bang or Hash-bang
#!/bin/bash
cat hello.c
ls
./hello
// End of sample.sh
```

```
$ chmod +x sample.sh
$ ./sample.sh
```

Bash is not the only shell!

`#!/bin/csh` #This is a sample C-shell script

```
echo -n the date of today is’ date
```

`#!/bin/ksh` #This is a sample K-shell script (Korn Shell)

```
echo “the date of today is \c” Date
```

`#!/bin/bash` #This is a sample BASH script

```
echo -n “the date is $date
```

However, Bash is the most popular shell

Shell Variables

- Two kinds of variables are supported by a shell
  - **Shell Environment Variables**
  - **User-Defined Variables**
    - (Both variables are stored as strings)
- **Shell environment variables (printenv)**
  - Used to customize the environment in which your shell runs.
  - Most of these variables are initialized by the start-up file /etc/profile.
  - Environment variables can also be added after the shell runs
- **User-defined variables**
  - Used within shell scripts for temporary storage.

Important read/write shell environmental variables

- HOME Full path name of your home directory
- PATH List of directories to search for commands
- MAIL Full path name of your mailbox
- USER Your user-name
- SHELL Your login shell
- PWD Current working directory
- TERM Type of your terminal
- Note: In order to access a shell variable, you must precede its name by the **$ sign.**

  E.g., `echo $MAIL`

Some Important Read only environmental variables //rov.sh

- $0 name of the program that is running
- $1 values of command line argument 1 (Similar for $2 to $9)
- $* values of all command line arguments
- $# total number of command line arguments ($1, $2…..)
- $$ Process ID of current process
- $? Exit status of most recent command
- $! PID of most recent background process

Defining/setting an environmental global variable

- $COLOR=yellow
- $echo $COLOR //output: yellow
- $export COLOR
- $printenv //COLOR=yellow will be listed
- $unset COLOR
- $printenv //COLOR=yellow will be unlisted

Quoting

- **Single quotes** (') inhibit wildcard (*)/variable ($)/ ` ` (Backticks)
- **Double quotes** (“) inhibit wildcard replacement only.
- When quotes are nested, only the outer quotes matter.
- Examples
  - echo The list of c files in this directory are `ls *.c` // “The list of c files in this directory are” + listing of c files
  - echo ‘The list of c files in this directory are `ls *.c`’ // The list of c files in this directory are `ls *.c`
  - echo I am $USER   // I am pranga
  - echo 'I am $USER’   // I am $USER
  - Ehco “I am $USER”   // I am pranga // “ “ do not inhibit $ and `` , * only

Shell Programming in BASH

- In addition to the basic facilities, shells have built-in programming languages that support:
  - conditions,
  - loops,
  - input/output
  - basic arithmetic

```bash
#!/bin/bash
# ex1.sh
echo -n "Enter a value> "
read a
echo -n "Enter another value> "
read b
echo "Doing arithmetic> "
sum=$(( a + b ))
echo "The sum $a + $b is $sum"
difference=$(( a - b ))
echo "The difference $a - $b is $difference"
product=$((a * b))
echo "The product $a * $b is $product"
if [[ $b -ne 0 ]]; then
quotient=$((a / b))
echo "The division $a / $b is $quotient"
else
echo "The division $a/$b is not possible"
fi
```

```
#!/bin/bash
#Demonstrates the working of read-only environmental variables    ex2.sh
if [ $# != 2 ]; then
echo "Usage: $0 integer1 integer2"
else
echo Doing arithmetic>
r=$(($1 + $2))
echo "the sum $1 + $2 is $r"
r=$(($1 - $2))
echo "the subtraction $1 - $2 is $r"
r=$(($1 * $2))
echo "the product $1 * $2 is $r"
if [ $2 -ne 0 ] ; then
r=$(($1 / $2))
echo "the division $1 / $2 is $r"
else
echo "the division $1 / $2 is not possible"
fi
fi
```

File Options

- rwx rwx rwx (user groups others) Ex chmod 111 *filename //001 001 001 ensures that you will not be able to either read or write, but only execute the file*
  - *$ chmod +x filename (sets x in UGO to 1)*
  - *$ chmod u+x filename (sets x in U to 1)*
  - *$ chmod –x filename (sets x in UGO to O)*
- File expressions : -option filename // Ex: the expression **-w filename** returns a 1, if the file has write permission set for the user, else returns a 0
- The value is 1 if the selected option is true and 0 otherwise.
- The available **options** are:
  - r Shell has read permission
  - w Shell has write permission
  - x Shell has execute permission
  - e file exists
  - O file is owned by shell's uid //Upper Case
  - Z file exists but is of size 0 //Upper Case
  - f file is a regular fille and not a directory
  - d file is a directory

```
#!/bin/bash
#ex4.sh
echo -n "Enter file name> "
read file
if [ -w $file ]; then
ls >> $file
echo "More input has been appended"
elif [ -e $file ]; then
echo “The file exists, but you have no write permission on $file"
else
echo "$file does not exist"
fi
```

//ex4b.sh (file operations)

```
#!/bin/bash
#ex4b.sh
echo -n "Enter file name/directory name> "
read fsd
echo The name of the file/directory is $fsd

if [ -d $fsd ]; then
echo "This is a directory"
elif [ -e $fsd ]; then
echo "This is a file"
else
echo "File or directory does not exist"
fi
```

Control structures: **If statement (various forms)**

```
if [<exp>];then
<commands>
fi
```

```
if [<exp>]; then
<commands1>
else
<commands2>
fi
```

```
if [<exp1>];then
<commands 1>
elif [<exp2>];then
<commands 2>
else
<commands 3>
fi
```

```
if [ -e $fname ]; then
echo File/Directory exists
fi
```

```
if [ -d $fsd ]; then
echo "This is a directory"
else
echo "This is a file“
fi
```

```
if [ -w $file ]; then
ls >> $file
echo "More input has been appended"
elif [ -e $file ]; then
echo "You have no write permission on $file"
else
echo "$file does not exist"
fi
```

```
if [ -e $fsd ] && [ -w $fsd ]; then
echo "File or directory exists and you have write permission"
fi
```

```
#!/bin/bash
# ex5.sh various forms of if statements
echo -n "Enter file name> "
read file
if [ ! -e $file ]; then   #File does not exist
echo "Sorry, $file does not exist."
elif [ ! -w $file ]; then            # File exists, but you have no write permission
            echo "You have no write permission on $file”
            if [ -O $file ]; then   #file exists, no write permission, you are the owner
            chmod u+w $file   #(grant write permission)
            echo "Write permission granted"
            else
            echo "Write permission cannot be granted“
            echo "because you don't own this file“  #You are not the owner
            fi
else   # File exists, and it has the write permission, add contents of ls
ls >> $file
echo "More input has been appended"
fi
```

Comparison Operators (Integer Comparison)

- -eq is equal to
- -ne is not equal to
- -gt is greater than
- -ge is greater than or equal to
- -lt is less than
- -le is less than or equal to
- < is less than (within double parentheses)
- <= is less than or equal to (within double parentheses)
- > is greater than (within double parentheses)
- >= is greater than or equal to (within double parentheses)

Comparison Operators (String Comparison)

= is equal to **`if [ $a = $b ]`**

== is equal to **`if [ $a == $b ]`**

!= is not equal to **`if [ $a != $b ]`**

< is less than, in ASCII alphabetical order **`if [ $a < $b]]`**

> is greater than, in ASCII alphabetical order **`if [ $a > $b ]`**

-z string is *null*, that is, has zero length **if [-z $a]**

-n string is not null **if [-n $a]**

While Statement

while statement:

While[expression]

do

commandList

done

While-Example //whilex.sh

-
  ```
  #!/bin/bash
  counter=$1
  factorial=1
  while [ $counter -gt 0 ]
  do
     factorial=$(( $factorial * $counter ))
     counter=$(( $counter - 1 ))
  done
  echo $factorial
  ```

Repeat Until

```
until [ expression ]     //the loop runs as long as the expression is FALSE
do
command list
done
```

Repeat Until //until.sh

-
  ```
  #!/bin/bash
  counter=$1
  factorial=1
  until [ $counter -eq 0 ]
  do
     factorial=$(( $factorial * $counter ))
     counter=$(( $counter - 1 ))
  done
  echo $factorial
  ```

For Statement

```
for VAR in {VAR value list}
do
{ code }
done

for (( i=0; i<5; i++ ))
do
{ code }
done

# using command line arguments
for k in $1 $2 $3 $4
do
echo $k
done

# using all command line arguments
for k in $*
do
echo $k
done
```

For-Examples //exfor.sh

```
#!/bin/bash
# exfor.sh


echo For loop with an explicit list


for i in 2 4 6 8 15
do
echo $i
done


echo For Loop with range and default increment of 1
for i in {1..10}
do
echo $i
done
```

```
echo For loop with increments of 2 within a range

for i in {1..10..2}
do
echo $i
done

echo For loop similar to C

for ((i=0;i<10;i++))
do
echo $i
Done

# end exfor.sh
```

Case Statement

```
case EXPRESSION in
PATTERN_1)
STATEMENTS
;;
PATTERN_2)
STATEMENTS
;;
PATTERN_N)
STATEMENTS
 ;;
*)
STATEMENTS
 ;;
esac
```

//case3.sh
//While and case

```
#!/bin/bash
while [ true ]
do
echo Select a day: MON WED or FRI
read option
case $option in
"MON") echo you selected MON;;
"WED") echo you selected WED;;
"FRI") echo you selected FRI;;
*) echo sorry,your input was incorrect
break ;;
esac
done
```

Case Statement //case1t.sh

```
#!/bin/bash
while [ true ]
do
echo Enter 1 for ls, 2 for ls -1, 3 for ls -l and 4 to exit
read option
case $option in
“1”) echo you selected ls
   ls;;
“2”) echo you selected ls -1
   ls -1 ;;
“3”) echo you selected ls -l
   ls -l;
“4”) break;;
esac
done
```

trap Command

```
#!/bin/bash
# trap.sh

trap "echo CTRL+C does not work over here" SIGINT
echo "The script is going to run until you hit Ctrl+Z"
echo "Try CTRL+C if you want to"

while [ true ]
do
sleep 1
done
# On a related note: you cannot make Ctrl-C work in this shell
because it has been trapped
```

Thank you