Review

- Linux System Roadmap
  - Libraries
- What is the shell
- Simple Bash Commands
- Redirecting Input and Output
- Pipeline

Preview

- Shell Scripts
- Shell Programming
- Shell Syntax
  - Variables
  - Quoting
  - Environment Variables
  - Parameter Variables
  - Condition
  - The test or "[" command

Shell Script

- A **shell script** is a plain-text file (ASCII code) that contains shell commands.
- It can be executed by typing its name into a shell, or by placing its name in another shell script (a shell script can be called inside a shell script).
- To be executable, a shell script file must meet some conditions:
  - Need provide where bash sell program (interpreter) is located
  - Need change shell script mode to executable

Shell Script

- The file must have a special first line that names an appropriate command processor.
  #!/bin/sh
- If this example doesn't work, you need to find out where executable Bash shell is located and substitute that location in the above example. Here is one way to find out:
  whereis sh ;; or **whereis bash**
- When a text file is created, default permission for new file is
  rw-rw-rw— //depends of umask value 0022
- The file must be made executable by changing its permission bits by using chmod command. An example:
  chmod +x (shell script filename)

Shell Script

- A shell script file may optionally have an identifying suffix, like ".sh". This only helps the user remember which files are which.
- One normally executes a shell script this way:
  - ./scriptname.sh

Shell Script

chmod: (Change Mode)

- Each file or directory has permission code called MODE.
- **chmod** command changes the permissions of each given file or directory according to MODE.
- The MODE can be either an **octal number** representing the bit pattern for the new permissions or a **symbolic representation** of changes to make.

Shell Script

A file MODE

user	group	others
rwx	rwx	rwx
- - -	- - -	- - -

Shell Script

chmod: (Change Mode)

- Octal number representation
  - Three bits is needed to save octal number.
  - 000, 001, 010, 011, 100, 101, 110, 111
  - Each file mode composed with three digit octal number.
  - 777, 577 or,….
  - We can change a file mode by chmod with octal number representation

    chmod 755 filename

Shell Script

user	group	others
rwx	rwx	rwx
111	000	000
7	0	0

Shell Script

user	group	other
rwx	rwx	rwx
111	101	000
7	5	0

Shell Script

user	group	other
rwx	rwx	rwx
111	111	111
7	7	7

Shell Script

chmod: (Change Mode)

- Symbolic representation
  - Combination of letter 'a' (all), 'u' (user), 'g' (group) and 'o' (other) controls which **users**' access to the file will be changed:

```
chmod a+rwx filename
chmod a-rwx filename
chmod g+x filename
chmod u+x filename
```

The Shell Programming

```
#!/bin/sh
# first.sh
# This file looks through all the files in the current
# directory for the string "main", and then print the
# name of those files to the standard output.

for file in *
do
       if grep -q main $file #-q: do not write on std out
       then
              echo $file
       fi
done
exit 0
```


Shell Syntax
(variables)

- We don't need declare variables ; we just use it when we need to used it.
- All variables are considered and stored as c-strings, even when they are assigned numeric values.
- Linux is a case-sensitive system.
- If numeric values, which are stored as strings, if it need to be used as a numeric value, the shell and utilities will convert.

Shell Syntax
(variables)

- Within the shell we can access the contents of a variable by preceding it name with a character **'$'**.
  - salutation=hello
  - echo $salutation
- If there are spaces in a string, the string must be written between quotation marks
  - salutation="How are you?"
  - echo $salutation

Shell Syntax
(variables)

- We can assign user input to a variable by using the read commend.
  - read salutation
  - How are you
  - echo $salutation
- We don't need quotation mark for reading a string for assigning to a variable.

Shell Syntax
(Quoting)

- Parameters in scripts are separated by a space.
- If you want a parameter to contain more than one space, you need use quotation for the parameter.
- The behavior of variable such as $foo inside quotes depends on the type of quotes used.

Shell Syntax
(Quoting)

```
#! /bin/sh
# quote.sh for testing quotation
name="Sang-eon Park"
echo $name                  #Display Sang-eon Park
echo "$name"                #Display Sang-eon Park
echo '$name'                #Display $name
echo \$name                 #Display $name
echo Enter any name
read name
echo "$name" now equals $name
exit 0
```

Environment Variables

- In all Unix and Unix-like systems, each process has its own private set of environment variables.
- It is initiated when a process is created from values in the environment.
  - **PS1 –** defines the shell's command-line prompt.
  - **HOME** – defines the home directory for a user.
  - **PATH** – defines a list of directories to search through when looking for a command to execute.
  - $0 – name of shell script
  - $# - the number of parameter passed
  - **IFS** – input field separator; a list of characters that are used to separate words when the shell is reading.
  - …
- To list the current values of all environment variables, issue the command **env**

Environment Variables

- To list a specific variable with the echo command, prefixing the variable name with a dollar sign

  echo $HOME

  echo $PATH
- When you enter a command or name of executable file, the shell looks in each of the directories specified in PATH to try to find it.
- If it can't find the command in any of those directories, you'll see a message
- "Command not found"

Environment Variables

- If you decide to put your own programs in a bin directory under your home directory, you have to modify the PATH to include that directory

  PATH=$PATH:$HOME/bin
- So if PATH was set to **/bin:/usr/bin:/usr/local/bin** beforehand, it would now have the value **/bin:/usr/bin:/usr/local/bin:/home/separk/bin**.

Parameter Variables

- If your script is executed with parameters, some additional variables are created.
- Even if no parameter are passed, the preceding environment variable **$0** still exists.
- **$1, $2**, … : Parameters given to the script
- **$* $@**: a list of all parameters in a single variable separated by the first character in the IFS (Internal Field Separator).
- **$#** : number of parameter passed

Parameter Variables

```
#!/bin/sh
#paral.sh testing parameter variables
echo "you pass $# arguments"
for ARG in $@; do
       echo $ARG
done
exit 0
```


Parameter Variables

```
#!/bin/sh
#para.sh testing parameter variables
salutation=Hello
echo $salutation
echo "The program $0 is now running"
echo "The first parameter was $1"
echo "The second parameter was $2 "
echo "The number of parameters passed $#"
echo "The list of parameters were $*"
echo "The user's home directory is $HOME"
echo "The user's current working directory is $PWD "
echo "Your initial greeting is $salutation"
echo "You can change your greeting now."
echo "Please enter a new greeting"
read salutation
echo "Your new salutation is \" $salutation \" "
exit 0
```

Conditions

- Every programming language has the ability to test condition and perform different action based on the test result.
- Since a shell script condition can test the exit code of any commend and script written by a programmer.
- It is important to include and exit command at the end of any scripts that you write.

The test, or [, command

- On most systems, the "[" and "test" command are synonymous. When "[" is used for testing condition, a trailing "]" is also used just for readability.
- There must be a space after the [command.
- The condition types with the test command fall into three types
  - String comparison
  - Arithmetic comparison
  - File conditionals

The test, or [, command
(String Comparison)

- [string1 = string2] : true if two strings are equal
- [string1 != string2]: true if two strings are not equal
- –n string : true if the string is not null
- –z string: true if the string is null

The test, or [, command
(String Comparison)

```
#!/bin/sh
# condition.sh

myname="Sang-Eon"
echo "What is your name?"
read yourname

if [ -z $yourname ]; then
        echo "no input from keyboard"
        exit 1
elif [ $myname = $yourname ]; then
        echo "We have same name $myname"
else
        echo "My name is $myname. Your name is $yourname"
fi
exit 0;
```


The test, or [, command
(Arithmetic Comparison)

- [expression1 **–eq** expression2]; true if two expression are equal
- [expression1 **–ne** expression2]; true if two expression are not equal
- [expression1 **–gt** expression2]; true if expression1 is greater than expression2
- [expression1 **–ge** expression2]; true if expression1 is greater or equal to expression2
- [expression1 **–lt** expression2]; true if expression1 is less than expression2
- [expression1 **–le** expression2]; true if expression1 is less than or equal to expression2

The test, or [, command
(Arithmetic Comparison)

```
# arithmatic.sh
# shows [ commend
#!/bin/sh

# or if test 4 -eq 4; then
if [ 4 -eq 4 ]; then
   echo " 4 is equal to 4"
else
   echo " 4 is not equal to 4"
fi

a=4
b=4
if [ $a -eq $b ]; then
   echo " a is equal to b"
else
   echo " a is not equal to b"
fi

exit 0
```

The test, or [, command
(Arithmetic Comparison)

```
# arithmatic1.sh
# shows [ commend for expression
#!/bin/sh

echo "give two integers"
read a b
if [ $a -eq $b ]; then
   echo " $a is equal to $b"
elif [ $a -gt $b ]; then
   echo " $a is greater than to $b"
else
       echo " $a is less than $b"
fi
exit 0
```


The test, or [, command
(file Conditional)

- –d file: true if the file is a directory
- –e file: true if the file is exist
- –s file: true if the file has nonzero size
- –f file: true if the file is a regular file
- –g file: true if set-group-id is set on the file
- –u file: true if set-user-id is set on the file
- –r file: true if the file is readable
- –w file: true if the file is writable
- –x file: true if the file is executable

The test, or [, command
(file Conditional)

```
#!/bin/sh
# testCond.sh
# testing condtion with [
echo "file name to check?"
read fname
if [ -e $fname ]; then
   echo "the file $fname exist in the current directory"
else
   echo "There is no such a $fname file exist in the
   current directory"
   exit 1;
fi
exit 0;
```


The test, or [, command
(file Conditional)

```
#!/bin/sh
# testCond1.sh
# testing condtion with [

echo "file name to check?"
read fname
if [ -e $fname ]; then
          echo "the file $fname exist in the current directory"
else
          echo "There is no sush a $fname file exist in the current directory"
          exit 1;
fi

if [ -f $fname ]; then
          echo " the file $fname is regular file"
else
          echo " the file $fname is not regular file"
fi

exit 0;
```