SHELL SCRIPT BASIC

UNIX Programming 2014 Fall by Euiseong Seo

Shell Script

- Interactive shell sequentially executes a series of commands
- Some tasks are repetitive and automatable
  - They are what programs are for
- Shell script is a set of shell commands and directives
  - Similar to programs
  - To be executed sequentially, and sometimes repeatedly

A Simple Shell Script

```
#!/bin/bash
echo “Hello, World!”
```

Another Example

- Logout when a specific file does not exist

  -
    ```
    #!/bin/bash
    if test ! -f $FILE
    then
       if test "$WARN" = "yes"
       then
       echo "$FILE does not exist"
       logout
       fi
    fi
    ```

Editors

- UNIX provides huge number of editor options
  - cat
  - ed
- Line editors are too primitive
- Screen (or visual) editors and GUI editors are being popularly used

VI Editor

What is VI

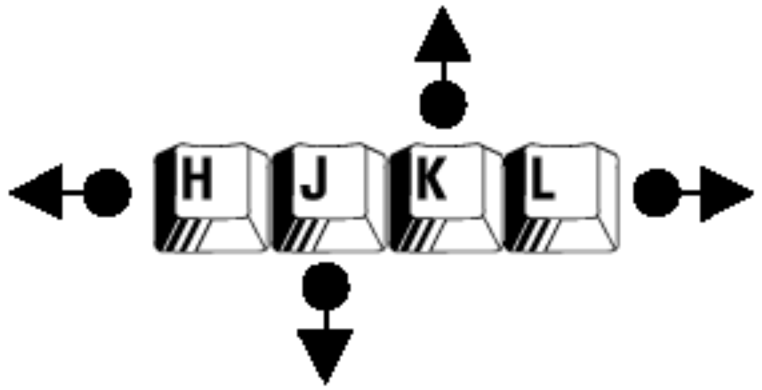
- Installed on most UNIX systems
- De-facto standard editor for CLI
  - Also defined by POSIX
- Originally written by Bill Joy in 1976
- VI derivatives
  - Vim(proved)
    - Syntax highlighting, mouse support and many other new features
  - Elvis
  - nvi
  - vile
  - busybox
    - A set of standard Linux utilities in a single executable
    - Including a tiny VI clone
    - Being used for embedded systems

Modes of Operation

- Insertion mode
  - What you input will be put into the current file
- Command mode
  - You can input commands to manipulate the current file or move cursor
  - Default mode
- Mode switch
  - Insertion to command
    - Esc key
  - Command to insertion
    - ‘a’, ‘i’ and ‘o’

Cursor Move

- Basic move



- '$' – end of the current line
- '0' – beginng of the current line
- ':30' – 30th line from the current line

Deletion

- ‘x’ – delete only the current character
- ‘dd’ – delete the current line
- ‘D’ – delete to the end of line
- ‘d10’ – delete 10 lines from the current line
- ‘p’ – past the deleted lines after the current line

Copy and Paste

- Yanking
  - Copying in VI
- ‘yy’ – yank a single line
- ‘y10’ – yang the following 10 lines including the current line
- ‘p’ – past the yanked lines

Save and Quit

- ‘:w’ – save the current file
- ‘:q’ – quit vi
- ‘:wq’ – save and quit

Search and Replace

- ‘/[pattern]’ – search forward for the pattern
- ‘?[pattern]’ – search backward for the pattern
- ‘n’ – search for the next instance of a string
- ‘:%s/foo/bar/g’ – find each occurrence of ‘foo’ in all lines, and replace them with ‘bar’
- ‘:s/foo/bar/g’ – find each occurrence of ‘foo’ in the current line, and replace them with ‘bar’

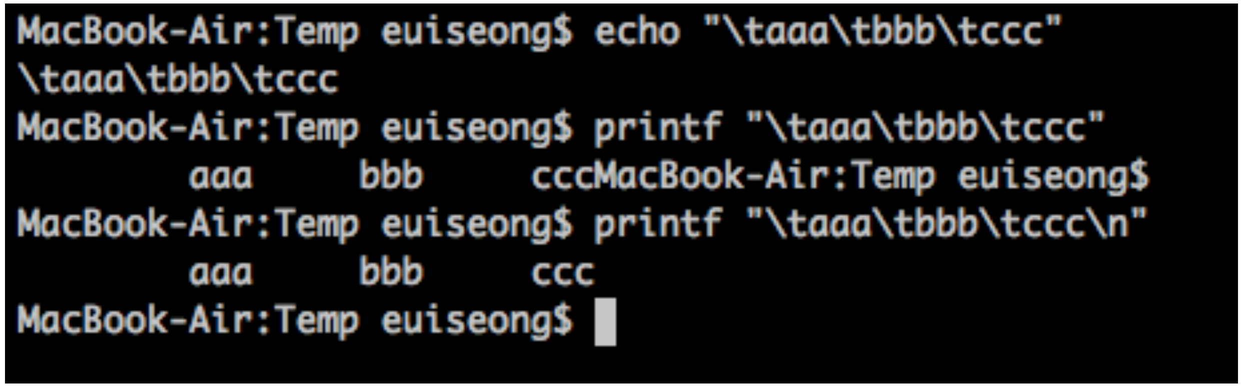
Shell Script Syntax

Shebang Statement

- Every shell script begins with a shebang statement
- Declaration of the interpreter to interpret the script
- Format
  - #!<interpreter path>
  - #!/bin/bash
  - #!/usr/bin/perl
  - #!/usr/bin/python
- Script must be executable to be interpreted
  - chmod a+x *scriptname*

Input and Output

- echo and printf
  - echo is crude but easy
  - If you want formatting, use printf



```
MacBook-Air:Temp euiseong$ echo "\taaa\tbbb\tccc"
\taaa\tbbb\tccc
MacBook-Air:Temp euiseong$ printf "\taaa\tbbb\tccc"
        aaa     bbb     cccMacBook-Air:Temp euiseong$
MacBook-Air:Temp euiseong$ printf "\taaa\tbbb\tccc\n"
        aaa     bbb     ccc
MacBook-Air:Temp euiseong$ 
```

Input and Output

- **read** command to prompt for input

```
#!/bin/bash

echo -n "Enter your name: "
read user_name

if [ -n "$user_name" ]; then
	echo "Hello $user_name!"
	exit 0
else
	echo "You did not tell me your name!"
	exit 1
fi
```

Command Line Arguments

- $0 – the name of the script
- $1 – the first argument
- $2 – the second argument
- $# - the number of arguments excluding $0
- $* - all arguments excluding $0

Command Line Arguments

```
#!/bin/bash

function show_usage {
	echo "Usage: $0 source_dir dest_dir"
	exit 1
}

# Main program starts here

if [ $# -ne 2 ]; then
	show_usage
else # There are two arguments
	if [ -d $1 ]; then
		source_dir=$1
	else
		echo 'Invalid source directory'
		show_usage
	fi
	if [ -d $2 ]; then
		dest_dir=$2
	else
		echo 'Invalid destination directory'
		show_usage
	fi
fi

printf "Source directory is ${source_dir}\n"
printf "Destination directory is ${dest_dir}\n"
```

Functions

- function *function_name* { }
- Function arguments
  - $0 – the script name
  - $1 – the first function argument
  - $# – the number of function arguments
- Revised version

```
function show_usage {
      echo "Usage: $0 source_dir dest_dir"
      if [ $# -eq 0 ]; then
            exit 99 # Exit with arbitrary nonzero return code
      else
            exit $1
      fi
}
```

Variable and Scope

- Variables are global within a script
- Functions can create their own local variables
  - With a `local` declaration

```
#!/bin/bash

function localizer {
      echo "==> In function localizer, a starts as '$a'"
      local a
      echo "==> After local declaration, a is '$a'"
      a="localizer version"
      echo "==> Leaving localizer, a is '$a'"
}

a="test"
echo "Before calling localizer, a is '$a'"
localizer
echo "After calling localizer, a is '$a'"
```

If-Statement

- Syntax

  -
    ```
    if [ condition ] ; then
      statement1
      …
    fi
    ```

  -
    ```
    if [ condition ] ; then
      statement1
    elif [ condition ] ; then
      statement2
    else
      statement3
    fi
    ```

If-Statement

- Nested if statement

  -
    ```
    if [ condition ]; then
      if [ condition ]; then
            statement1
      elif
            statement2
      else
            statement3
      fi
    else
      statement4
    fi
    ```

If-Statement

- Comparison operators for condition
  - x = y ➔ x –eq y
  - x != y ➔ x –ne y
  - x < y ➔ x –lt y
  - x > y ➔ x –gt y
  - x <= y ➔ x –le y
  - x >= y ➔ x –ge y
  - x is not null ➔ –n x
  - x is null ➔ -z x

If-Statement

- File evaluation operators for condition
  - file exists ➔ -e
  - file exists and a directory ➔ -d
  - file exists and a regular file ➔ -f
  - file exists and not empty ➔ -s
  - file has readable permission for you ➔ -r
  - file has writable permission for you ➔ -w
  - file1 is newer than file2 ➔ file1 –nt file2
  - file1 is older than file2 ➔ file1 –ot file2

If-statement

- Combining conditions
  - And
    - [condition1] && [condition2]
  - Or
    - [condition1] || [condition2]

Case statement

- Syntax
  - case *expression* in

```
    pattern1)
            statement ;& // Fall through
    pattern2)
            statement
            statement
            ;;
    *)
            statement
            ;;
esac
```

Case statement

- Example
  - case $# in

```
case $# in
    0)
            echo "No arguments"
            ;&
    1)
            echo "Insufficient arguments"
            ;;
    *)
            do_operation
            ;;
esac
```

Examples

- Write an log message printing function, logmsg
- This function gets two parameters, log level and log message
  - There are five log levels depending on the importance
    - Level 0: Error
    - Level 1: Warning
    - Level 2: Info
    - Level 3: Debug
    - Level 4: Other
- The function prints log message in the following format
  - "Info: Configuration file not found"
- There is a global variable PRINT_LEVEL that determines the highest level that will be printed