Open Source Programming

Lecture-03
Shell Script

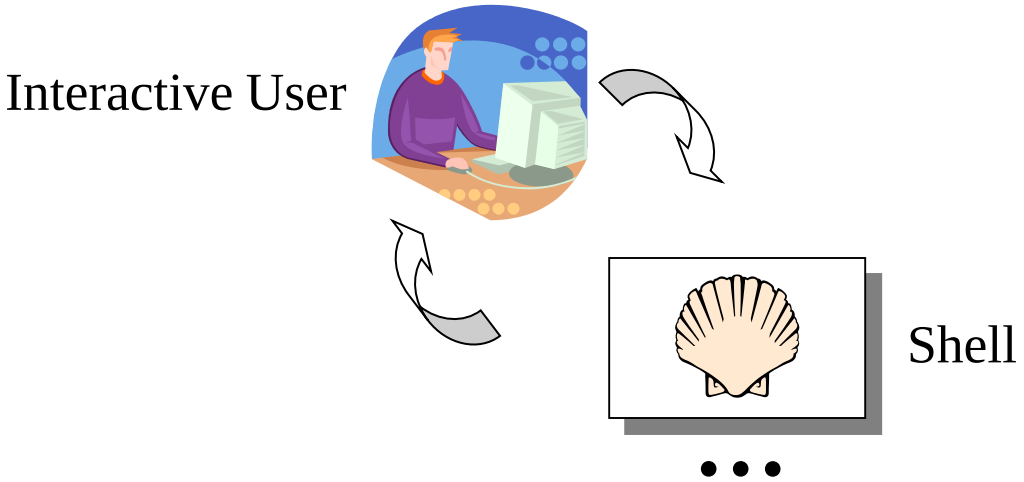
Contents

- **Shell / Shell Script**

- **Shell Script Syntax / Examples**
  - if_then_else
  - case
  - while loop
  - until
  - for loop
  - select
  - continue_break
  - function
  - local variable
  - signal
  - array

What is shell?

- **Shell**
  - Command interpreter for UNIX/LINUX
    - No compilation
  - Interactive interface to the UNIX/LINUX system
  - An environment in which we can run our commands, programs, and shell scripts
  - User can issue command and see the output



- **Shell prompt**
  - Command prompt: "$"

Shell Script

- **What is shell script?**
  - **Ordered list of commands in a text file** to be executed in Linux/UNIX shell environment
  - A kind of a *simple programming tool*
  - Example)

```
#!/bin/bash
echo "Hello world!"
pwd
ls -la
```

- **Commands include**
  - Anything you can type on the command line
  - Shell variables
  - Control statements (if, while, for)

Shell Script

- **Practice**
  - 터미널 창에서 다음 명령어들을 직접 실행

```
$ echo “Hello world!”
$ pwd
$ ls -a
```

  - vi를 사용해서 myprog.sh라는 파일 생성

```
$ vi myprog.sh
```

    - 'i’ : 내용 입력 ➔
    - Esc: command mode로 돌아옴
    - ‘:wq’: 저장후 종료

```
#!/bin/bash
echo "Hello world!"
pwd
ls -la
```

  - 실행가능하도록 파일의 속성을 변경

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

  - Shell Script 실행

```
$ ./myprog.sh
```

Shell Variables ($)

```
$ set tim bill ann fred
      $1  $2   $3  $4

$ echo $*
tim bill ann fred

$ echo $#
4

$ echo $1
tim

$ echo $3 $4
ann fred

$ echo $0
-bash
```

Parameter	Meaning
$0	Name of the current shell script
$1-$9	Positional parameters 1 through 9
$#	The number of positional parameters
$*	All positional parameters, "$*" is one string
$@	All positional parameters, "$@" is a set of strings
$?	Return status of most recently executed command
$$	Process id of current process

Control Structure

- **if-then-else**
- **case**
- **loops**
  - for
  - while
  - until
  - select

IF statement

```
if command
then
   statements
fi
```

Example:

```
if test -w "$1"
  then
  echo "file $1 is write-able"
fi
```

IF-then-else Statement

```
if [ condition ]; then
    statements
elif [ condition ]; then
    statement
else
    statements
fi
```

Operators in the condition

Meaning	Numeric	String
Greater than	-gt	
Greater than or equal	-ge	
Less than	-lt	
Less than or equal	-le	
Equal	-eg	= or ==
Not equal	-ne	!=
str1 is less than str2		str1 < str2
str1 is greater str2		str1 > str2
String length is greater than zero		-n str
String length is zero		-z str

Example

```
#!/bin/bash

read -p "Enter years of work: " Years

if [ ! "$Years" -lt 20 ]; then
   echo "You can retire now."
else
   echo "You need 20+ years to retire"
fi
```

Compound Logical Expressions

- ! (not)
- && (and)
- || (or)

- "And" and "Or" must be within
  [[... && ...]]

Using &&

```
#!/bin/bash

Bonus=500
read -p "Enter Status: " Status
read -p "Enter Shift: " Shift

if [[ "$Status" = "H" && "$Shift" = 3 ]]
then
   echo "shift $Shift gets \$$Bonus bonus"
else
   echo "only hourly workers in"
   echo "shift 3 get a bonus"
fi
```

File Testing

	Meaning
-d file	True if ‘file’ is a directory
-r file	True if ‘file’ is readable
-w file	True if ‘file’ is writable
-x file	True if ‘file’ is executable
-s file	True if length of ‘file’ is nonzero

```
drwxrwxr-x  2 ubuntu ubuntu 4096 Aug 20 15:17 mycode
-rw-rw-r--  1 ubuntu ubuntu   20 Aug 20 15:15 newfile.txt
```

File Testing Example

```
#!/bin/bash

echo "Enter a filename: "
read filename

if [ ! -x "$filename" ]
 then
   echo "File is not executable"
 exit 1
fi
```

File Testing Example

```
#! /bin/bash

read -p "Enter a filename: " filename
set $filename

if [ $# -lt 1 ]; then
        echo “Error: none filename"
        exit 1
fi

if [[ ! -r "$1" || ! -w "$1" ]]
then
      echo "File $1 is not accessible"
else
      echo "File $1 is accessible"
fi
```

If-then-else Example

```
#!/bin/bash

read -p "Enter Income Amount: " Income
read -p "Enter Expenses Amount: " Expense

let Net=$Income-$Expense

if [ "$Net" -eq "0" ]; then
   echo "Income and Expenses are equal - breakeven."
elif [ "$Net" -gt "0" ]; then
   echo "Profit of: " $Net
else
   echo "Loss of: " $Net
fi
```

Case Statement

Syntax:

```
case word in
    pattern1) command-list1
    ;;
    pattern2) command-list2
    ;;
    patternN) command-listN
    ;;
esac
```

Case Example

```
#!/bin/bash
echo "Enter Y to see all files including hidden files"
echo "Enter N to see all non-hidden files"
echo "Enter Q to quit"

read -p "Enter your choice: " reply

case $reply in
  Y|YES) echo "Displaying all (really…) files"
         ls -a ;;
  N|NO)  echo "Display all non-hidden files..."
         ls ;;
  Q)     exit 0 ;;

  *) echo "Invalid choice!"; exit 1 ;;
esac
```

While Loop

- To execute commands in “command-list” as long as “expression” evaluates to true

Syntax:

```
while [ expression ]
do
    command-list
done
```

While Loop Example

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]
do
    echo The counter is $COUNTER
    let COUNTER=$COUNTER+1
done
```

While Loop Example 2

```
#!/bin/bash
# copies files from directory1 - into the directory2
# A new directory is created every hour

DIR1=test/dir1
DIR2=test/dir2

while true; do
    DATE=`date +%Y%m%d`
    HOUR=`date +%H`
    mkdir $DIR2/"$DATE"
    while [ $HOUR -ne "00" ]; do
        DESTDIR=$DIR2/"$DATE"/"$HOUR"
        mkdir "$DESTDIR"
        mv $DIR1/*.txt "$DESTDIR"/
        sleep 3600
        HOUR=`date +%H`
    done
done
```

Until Loop

- To execute commands in “command-list” as long as “expression” evaluates to false

Syntax:

```
until [ expression ]
do
    command-list
done
```

Until Example

```
#!/bin/bash

COUNTER=20
until [ $COUNTER -lt 10 ]
do
    echo $COUNTER
    let COUNTER-=1
done
```

Until Example

```
#!/bin/bash

Stop="N"
until [ $Stop = "Y" ]; do
  ps -A
  read -p "want to stop? (Y/N)" reply
  Stop=`echo $reply | tr [:lower:] [:upper:]`
done
echo "done"
```

For Loop

- **To execute commands as many times as the number of words in the "argument-list"**

Syntax:

```
for variable in argument-list
do
    commands
done
```

For Loop Example

```
#!/bin/bash

for i in 7 9 2 3 4 5
do
    echo $i
done
```

For Loop Example

```
#!/bin/bash
# compute the average weekly temperature

for num in 1 2 3 4 5 6 7
do
   read -p "Enter temp for day $num: " Temp
   let TempTotal=$TempTotal+$Temp
done

let AvgTemp=$TempTotal/7
echo "Average temperature: " $AvgTemp
```

Iterating over Parameters

```
#!/bin/bash

for parm
do
        echo $parm
done
```

Select Command Example

```
#!/bin/bash

select var in alpha beta gamma
do
     echo $var
done
```

Select Command Example

```
#!/bin/bash

# set PS3 prompt
PS3="Enter the space shuttle name : "

# set shuttle list
select shuttle in columbia endeavor (이어서…)
  challenger discovery atlantis enterprise
  pathfinder
do
    echo "$shuttle selected"
done
```

More Example of Select

```
#!/bin/bash

echo "script to make files private"
echo "Select file to protect:"

select FILENAME in *
do
  echo "You picked $FILENAME ($REPLY)"
  chmod go-rwx "$FILENAME"
  echo "it is now private"
done
```

Continue and Break

```
while [ condition ]
do
    command
    continue
    command
    break
    command
done
echo "done"
```

Continue and Break Example

```
#!/bin/bash

for index in 1 2 3 4 5 6 7 8 9 10
do
        if [ $index -le 3 ]; then
              echo "continue"
              continue
        fi

        echo $index

        if   [ $index -ge 8 ]; then
              echo "break"
              break
        fi
done
```

Functions

- For *reusing* codes
- Shell script within a shell script
- Placed *at the beginning* of the script
- Function must appear before being called

```
#!/bin/bash

function-name() {
    statements
}

commands
function-name
```

Function Example

```
#!/bin/bash

# A somewhat more complex function
fun() {
  JUST_A_SECOND=1
  let i=0
  REPEATS=30
  echo "And now the fun really begins."
  while [ $i -lt $REPEATS ]
  do
       echo "-------FUNCTIONS are fun-------->"
       sleep $JUST_A_SECOND
       let i+=1
  done
}

# run function
fun
```

Function Parameters

- **Sending data to the function**
- **Named parameter not necessary**
- **The function refers to passed arguments by their position (not by name) as `$1, $2, $3` …**
  - `$#` Number of parameters
  - `$0` Script name

```
#!/bin/bash
testfile() {
  if [ $# -gt 0 ]; then
     if [[ -r $1 ]]; then
        echo $1 is a readable file
     else
        echo $1 is not a readable file
     fi
  fi
}

testfile hello.txt
testfile test/hello.txt
```

Local Variables

- **A variable declared as local** is one that is visible only **within the block** of code in which it appears.
- Variables defined within functions are global unless **keyword 'local'** is used.

- Before a function is called, *all* variables declared within the function are invisible outside the body of the function, not just those explicitly declared as *local*.

Local Variable Example

```
#!/bin/bash

global="pretty good variable"

foo() {
        local inside="not so good variable"
        echo $global
        echo $inside
        global="better variable"
}

echo $global
foo
echo $global
echo $inside ???
```

Linux Signals

- **List processes**

```
$ ps
$ ps -ef
```

- **Signal**
  - IPC (Inter-Processes Communication)
  - Asynchronous
  - Notification to the process of an event

- **Sending a signal: `kill`**
  - `$kill -HUP 1234`
  - `$kill -2 1235`
  - `$kill -9 1236`

Commonly Used Signals

Signal Name	Signal Number	Description
SIGHUP	1	Hang up detected on controlling terminal or death of controlling process
SIGINT	2	Issued if the user sends an interrupt signal (Ctrl + C)
SIGQUIT	3	Issued if the user sends a quit signal (Ctrl + D)
SIGFPE	8	Issued if an illegal mathematical operation is attempted
SIGKILL	9	If a process gets this signal it must quit immediately and will not perform any clean-up operations
SIGALRM	14	Alarm clock signal (used for timers)
SIGTERM	15	Software termination signal (sent by kill by default)

List of Signals

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Handling Signals

- **Default action**
  - To terminate the process
- **Installing custom signal handler**

```
trap 'handler commands' signals
```

```
#!/bin/bash

# kill -2 won't kill this process
# kill -9 will
echo $$
trap 'echo dont interrupt' 2

while true
do
        echo "try interrupt"
        sleep 1
done
```

Multiple Signal Handlers

```
#!/bin/bash

# plain kill or kill -9 will kill this
trap 'echo 1' 1
trap 'echo 2' 2
echo $$

while true; do
    echo -n .
    sleep 1
done
```

ex) Removing Temporary Files

```
#!/bin/bash
trap 'cleanup; exit' 2

cleanup() {
        /bin/rm -f /tmp/tempfile.$$.?
}

for i in 1 2 3 4 5 6 7 8
do
        echo "$i.iteration"
        touch /tmp/tempfile.$$.$i
        sleep 1
done
cleanup
```

Run Signal Handler Once

- `trap` without a command list will remove a signal handler

```
#!/bin/bash

trap 'justonce' 2
justonce() {
  echo "not yet"
  trap 2              # now reset it
}

while true; do
  echo -n "."
  sleep 1
done
```

Variables

- **Numeric variables**
  - Declaration and setting value:

    ```
    declare -i var=100
    ```

  - Expressions in the style of C:
    - `(( expression ))`
    - `e.g. (( var+=1 ))`

- **String variables**
  - By default a variable is a string type
    - `var=100`

Variables

- **Array is a list of values**
  - Don't have to declare size
- **Reference a value by `${name[index]}`**
  - `${a[3]}`
  - `$a` (same as `${a[0]}`)
- **Use the declare -a command to declare an array**
  - `declare -a sports`
  - `sports=(ball frisbee puck)`
  - `sports[3]=bat`
- **Array initialization**
  - `sports=(football basketball)`
  - `moresports=(${sports[*]} tennis)`
- **`${array[@]}` or `${array[*]}`** refers to the entire array contents
- **`echo ${moresports[*]}`** produces

  `football basketball tennis`

Array Example

```
#!/bin/bash

declare -a sports
sports=(ball frisbee puck)
sports[3]=bat

echo ${sports[*]}

for i in "${sports[@]}";
do
   echo "$i";
done
```

Array Example

```
#!/bin/bash

declare -a sports
sports=(ball frisbee puck)
for i in "${!sports[@]}";
do
    echo "$i";
done

echo "the array contains ${#sports[@]} elements"

sports+=(soccer baseball)
echo ${sports[*]}

unset sports[1]
echo ${sports[*]}

unset sports
echo ${sports[*]}
```

Any Questions...
Just Ask!