Linux Bash Shell Scripting

Bok, JongSoon
javaexpert@nate.com
https://github.com/swacademy/fss/tree/main/Linux

Script

- A program that is executed by another program called an ***interpreter***.
- JavaScript, Perl, Python, etc.
- Run faster when automated rather than manually.
- Automation ensures script consistency by eliminating potential manual errors.

BASH
THE BOURNE-AGAIN SHELL

```
#!/bin/bash
```

https://velog.io/@gudrmsglgl/Shell-script-%EC%97%90-%EB%8C%80%ED%95%B4-%EC%95%8C%EC%95%84%EB%B3%B4%EC%9E%90

ubuntu

Script (Cont.)

- Backup script example:

```
#!/bin/bash

#tar = Tape Archive tool for compression
tar -cvf /backupfolder/backup.tar /home/nishkarshraj/Desktop/Automation-using-
Shell-Scripts

#Go to the backup folder location
cd /backupfolder

#Show the size of the folder
du -sh
```

- Script result:

```
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/Switch Basics.sh
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/RadixSort_Binaryconversion.sh
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/function_basic.sh
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/Swapping variables using Function.sh
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/read.txt
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/input.sh
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/User defined for loop.sh
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/if_script.sh
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/output.sh
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/Array Sorting.sh
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/Standard Input Stream.sh
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/Standard Output and Standard Error Stream.sh
/home/nishkarshraj/Desktop/Automation-using-Shell-Scripts/Shell Scripting Basic
s/Else If.sh
3.4M    .
root@NoiceCurse:/#
```

Shell Script

- Programs run by the shell.
- Linux commands + program components provided by the shell.
- Shell command execution sequence
  1. Aliases
  2. Keyword(if, while, until…)
  3. Function
  4. Internal command(cd, echo…)
  5. Script, Utilities in the PATH

```bash
#!/bin/bash

greet() {
  echo "Hello, ${1}"
}

read -p "What is you name: " name

greet "$name"
```

How to Create First Bash Script

1. Create a file named hello_world.sh

```
touch hello_world.sh
```

2. Find the path to bash shell.

```
which bash
```

3. Write the command.

```
#! /usr/bin/bash
echo "Hello World"
```

4. Provide execution rights to user.

```
chmod u+x hello_world.sh
```

5. Run the script.

```
$ ./hello_world.sh
```

Or

1. File edit with VI(or Nano) editor.

```
#!/bin/bash

printf "I love Linux! \n"
pwd
```

2. Run the script with **bash** command.

```
ubuntu@ubuntu-desktop:/tmp$ ls test_shell.sh
test_shell.sh
ubuntu@ubuntu-desktop:/tmp$ bash test_shell.sh
I love Linux!
/tmp
ubuntu@ubuntu-desktop:/tmp$
```

Basic Script Syntax

- Refer to :
- https://devhints.io/bash
- https://velog.io/@pingping95/Bash-Shell-Script-%EA%B8%B0%EC%B4%88-%EA%B0%9C%EB%85%90
- https://twpower.github.io/131-simple-shell-script-syntax
- https://www.gnu.org/software/bash/manual/bash.html
- https://www.freecodecamp.org/news/bash-scripting-tutorial-linux-shell-script-and-command-line-for-beginners/
- https://mug896.github.io/bash-shell/
- https://www.youtube.com/watch?v=38wy3gsiR6Q&list=PLApuRlvrZKog2XlvGJQh9KY8ePCvUG7Je

ubuntu

Basic Script Syntax (Cont.)

- File extension of `.sh`
  - By naming conventions, bash scripts end with a `.sh`.
  - However, bash scripts can run perfectly fine without the ***sh*** extension.
- Scripts start with a bash bang.
  - Are also identified with a ***shebang***.
  - ***Shebang*** is a combination of bash `#` and bang `!` followed the bash shell path.
  - This is the first line of the script.
  - ***Shebang*** tells the shell to execute it via bash shell.
  - ***Shebang*** is simply an absolute path to the bash interpreter.

```
#! /bin/bash
```

ubuntu

Basic Script Syntax (Cont.)

- Single line and inline bash comment
- Both single line and inline comments in bash scripts start with the hash sign (#):

```
# This is a comment
```

```
#/bin/bash

#.....................
#Author : Jane Doe
# Date : 06/15/2021

#Description :Here is how you can document a script
#
#
#Usage :
# ./myScript.sh param1 [param2]
#param 1:
#param2:
#.....................
#Version: 2.0.1

#Declared variables

#.....................
#Scrit body
```

Basic Script Syntax (Cont.)

- Shell Variables
  - Are case sensitive.
  - Declare using **=** and use using **$**
  - **{}** is parameter substitution, substituting a variable in the part enclosed with **$**
  - It is safer to use it by wrapping it in **“”** (because can use values that include spaces in the string).
  - **=** must be written without spaces.
  - Add **local** to local variables.

```
ubuntu@ubuntu-desktop:/tmp$ cat test_shell.sh
#!/usr/bin/env bash

name="John"
echo "Hello $name!"
```

```
ubuntu@ubuntu-desktop:/tmp$ cat test_shell.sh
#!/usr/bin/env bash

name="John"
echo $name
echo "$name"
echo "${name}!"
ubuntu@ubuntu-desktop:/tmp$ bash test_shell.sh
John
John
John!
ubuntu@ubuntu-desktop:/tmp$
```

Basic Script Syntax (Cont.)

- Shell Variables (Cont.)
  - If add **export** before the variable name, it is set as an ***environment variable*** and can be used in child scripts.

```
ubuntu@ubuntu-desktop:/tmp$ export
declare -x DBUS_SESSION_BUS_ADDRESS="unix:path=/run/user/1000/bus"
declare -x HOME="/home/ubuntu"
declare -x LANG="en_US.UTF-8"
declare -x LC_ADDRESS="ko_KR.UTF-8"
declare -x LC_IDENTIFICATION="ko_KR.UTF-8
declare -x LC_MEASUREMENT="ko_KR.UTF-8"
declare -x LC_MONETARY="ko_KR.UTF-8"
declare -x LC_NAME="ko_KR.UTF-8"
declare -x LC_NUMERIC="ko_KR.UTF-8"
declare -x LC_PAPER="ko_KR.UTF-8"
declare -x LC_TELEPHONE="ko_KR.UTF-8"
declare -x LC_TIME="ko_KR.UTF-8"
declare -x LESSCLOSE="/usr/bin/lesspipe %
```

```
ubuntu@ubuntu-desktop:/tmp$ student="John"
ubuntu@ubuntu-desktop:/tmp$ echo $student
John
ubuntu@ubuntu-desktop:/tmp$ export student
ubuntu@ubuntu-desktop:/tmp$ printenv student
John
ubuntu@ubuntu-desktop:/tmp$
```

Basic Script Syntax (Cont.)

- Shell Variables (Cont.)
  - Differences between **set** and **export**

```
ubuntu@ubuntu-desktop:/tmp$ season="winter"
ubuntu@ubuntu-desktop:/tmp$ echo $season
winter
ubuntu@ubuntu-desktop:/tmp$ bash
ubuntu@ubuntu-desktop:/tmp$ echo $season

ubuntu@ubuntu-desktop:/tmp$ exit
exit
ubuntu@ubuntu-desktop:/tmp$ export student="John"
ubuntu@ubuntu-desktop:/tmp$ echo $student
John
ubuntu@ubuntu-desktop:/tmp$ bash
ubuntu@ubuntu-desktop:/tmp$ echo $student
John
ubuntu@ubuntu-desktop:/tmp$
```

Basic Script Syntax (Cont.)

- Shell Variables (Cont.)
  - Differences between ***local variable*** and ***global variable***

```
#!/bin/bash

# set global variable
string="hello world"
echo ${string}

string_test(){
    local string="local"
    echo ${string}
}

string_test

echo ${string}
```

```
ubuntu@ubuntu-desktop:/tmp$ vi test_shell.sh
ubuntu@ubuntu-desktop:/tmp$ bash test_shell.sh
hello world
local
hello world
ubuntu@ubuntu-desktop:/tmp$
```

Basic Script Syntax (Cont.)

- Refer to https://phoenixnap.com/kb/bash-commands

명령	설명
echo	콘솔에 정보 표시
read	사용자 입력 내용 읽기
subStr	문자열의 하위 문자열 가져오기
+	숫자 두 개를 더하거나 문자열 결합
file	파일 열기
mkdir	디렉터리 만들기
cp	파일 복사
mv	파일 이동 또는 이름 바꾸기
chmod	파일에 권한 설정
rm	파일, 폴더 등 삭제
ls	디렉터리 나열

Basic Script Syntax (Cont.)

- Operators
  - Arithmetic Operators : `+`, `-`, `*`, `/`, `%`, `++`, `--`
  - Relational Operators : `==`, `!=`, `<`, `<=`, `>`, `>=`, `-eq`, `-ne`, `-gt`, `-ge`, `-lt`, `-le`
  - Logical Operators : `&&`, `||`, `!`
  - Bitwise Operators : `&`, `|`, `^`, `!`, `<<`, `>>`

```
#!/bin/bash

sum=$(($1 + $2))
echo $1 + $2 equals $sum
```

Basic Script Syntax (Cont.)

- Expression
  - Are a way to answer questions that arise when a script or program runs.

```
#!/bin/bash

sum=$(($1 + $2))

echo $1 + $2 equals $sum
```

ubuntu

Control Flow Statements

- `If` Statement
  - Syntax

```
#!/bin/bash

if [ condition ]; then
  # code to be executed if the condition is true
fi
```

  - Example

```bash
#!/bin/bash

echo -n "Enter a number: "
read VAR

if [[ $VAR -gt 10 ]]
then
  echo "The variable is greater than 10."
fi
```

Control Flow Statements (Cont.)

- **`If ~ else`** Statement
  - Syntax

```
#!/bin/bash

if [ condition ]; then

# code to be executed if the condition is true

else

# code to be executed if the condition is false

fi
```

  - Example

```bash
#!/bin/bash

echo -n "Enter a number: "
read VAR

if [[ $VAR -gt 10 ]]
then
  echo "The variable is greater than 10."
else
  echo "The variable is equal or less than 10."
fi
```

Control Flow Statements (Cont.)

- **`If ~ elif ~ else`** Statement
  - Syntax

```
#!/bin/bash

if [ condition1 ]; then
# Code to be executed if condition1 is true
elif [ condition2 ]; then
# Code to be executed if condition2 is true
elif [ condition3 ]; then
# Code to be executed if condition3 is true
else
# Code to be executed if none of the conditions are true
fi
```

  - Example

```bash
#!/bin/bash

echo -n "Enter a number: "
read VAR

if [[ $VAR -gt 10 ]]
then
  echo "The variable is greater than 10."
elif [[ $VAR -eq 10 ]]
then
  echo "The variable is equal to 10."
else
  echo "The variable is less than 10."
fi
```

`test` Command

- Is used to test the validity of a command.
- Checks whether the command/expression is **`true`** or **`false`**.
- Syntax

```
test [expression]
```

- Example

```
number1=5
number2=10
if test $number1 -eq $number2; then
    echo "Numbers are equal."
else
    echo "Numbers are not equal."
fi
```

Integer Comparison Operators

```
if [ "$1" -gt "$2" ];
then
        echo “….”
fi
```

```
if (("$1">"$2"))
then
        echo “…”
fi
```

```
if [[ $1 -gt $2 ]]
then
        echo “…”
fi
```

```
if [ $1 -gt $2 ]
then
        echo “…”
fi
```

```
if (($1>$2));
then
        echo “…”
fi
```

ubuntu

String Comparison Operators

Operators	Description
`=`, `==`	Equals
`!=`	Not equals
`<`	Lesser in ASCII alphabetical order
`>`	Greater in ASCII alphabetical order
`-z`	String NULL, length 0
`-n`	Is not String NULL
`-l`	String length
`${variable}`	Is not String NULL

ubuntu

Control Flow Statements (Cont.)

- **for** Statement
  - Simple for loop
  - Range-based for loop
  - Array iteration for loops
  - C-Styled for loops
  - Infinite for loop

```bash
#!/bin/bash

for n in a b c;
do
   echo $n
done
```

```bash
#!/bin/bash

for n in {1..5..2};
do
    echo $n
done
```

```bash
#!/bin/bash

for n in {1..5};
do
    echo $n
done
```

```bash
#!/bin/bash

s=("football" "cricket" "hockey")
for n in ${s[@]};
do
    echo $n
done
```

```bash
#!/bin/bash

n=4
for (( ; ; ));
do
    if [ $n -eq 9 ];then
        break
    fi
    echo $n
    ((n=n+1))
done
```

```bash
#!/bin/bash

n=7
for (( i=1 ; i<=$n ; i++ ));
do
    echo $i
done
```

ubuntu

Control Flow Statements (Cont.)

- **`while`** Statement
  - Syntax

```
while [ condition ];
do
    # statements
    # commands
done
```

  - Example

```bash
#!/bin/bash

counter=1

while [ $counter -le 10 ];
do
        echo $counter
        counter=$((counter+1))
done
```

Control Flow Statements (Cont.)

- **`until`** Statement
  - Syntax

```
until [CONDITION]
do
  [COMMANDS]
done
```

  - Example

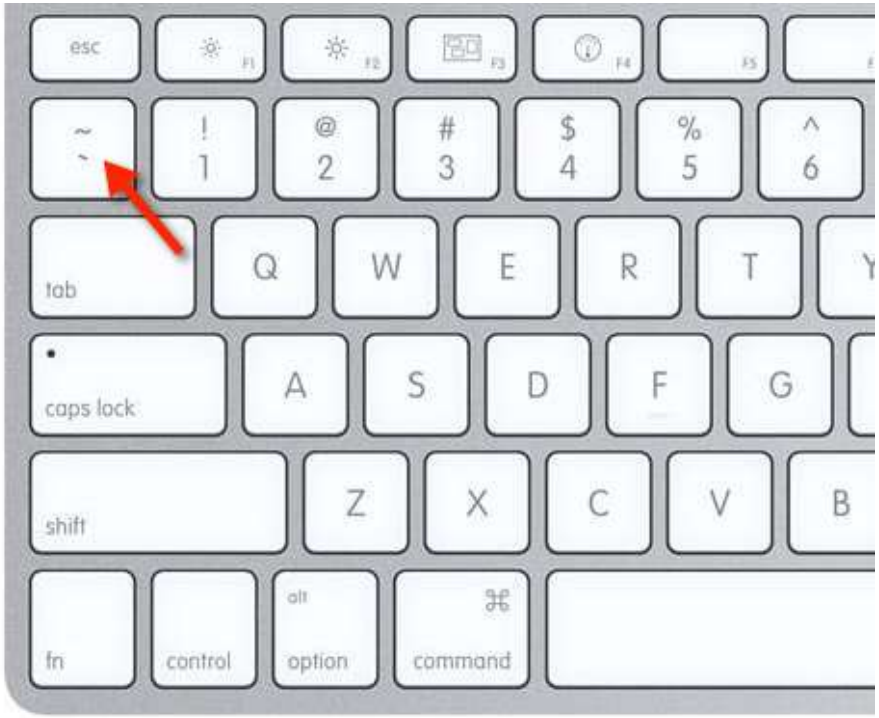
```
#!/bin/bash

counter=0

until [ $counter -gt 5 ]
do
  echo Counter: $counter
  ((counter++))
done
```

Backtick(`) symbol in Linux Shell Scripting

- Allows to assign the output of a shell command to a variable.
- Must surround the entire command line command with backtick characters:




```
#!/bin/bash
DATE= `date`
echo "You have accessed the script on: $DATE"
```

```
#!/bin/bash
# copy the /usr/bin directory listing to a log file
today=`date +%y%m%d`
ls /usr/bin -al > /tmp/log.$today
```

```
#!/bin/bash

echo "Hello $USER!"
echo "Today's date is : `date`"
```

Arguments

- Positional arguments

```
echo "Username: $1";
echo "Age: $2";
echo "Full Name: $3";
```

```
ubuntu@ubuntu-desktop:/tmp$ cat test_shell.sh
#!/bin/bash

echo "User name: $1"
echo "Age: $2"
echo "Full Name: $3"
ubuntu@ubuntu-desktop:/tmp$ bash test_shell.sh john 25 'John Smith'
User name: john
Age: 25
Full Name: John Smith
ubuntu@ubuntu-desktop:/tmp$
```

Arguments (Cont.)

- Flags
  - **$#**: Number of arguments
  - **$***: All positional arguments(as a single word)
  - **$@**: All positional arguments(as separate strings)
  - **$_**: Last argument of the previous command

```
ubuntu@ubuntu-desktop:/tmp$ cat test_shell.sh
#!/bin/bash

# Number of arguments
echo "$#"

# All arguments(as separate strings
echo "$@"

# All arguments(as a single word)
echo "$*"
ubuntu@ubuntu-desktop:/tmp$ bash test_shell.sh spring summer winter
3
spring summer winter
spring summer winter
ubuntu@ubuntu-desktop:/tmp$
```

Arguments (Cont.)

- Flags (Cont.)

```
ubuntu@ubuntu-desktop:/tmp$ cat test_shell.sh
#!/bin/bash

# All arguments(as separate strings
for argv in "$@"
do
    echo $argv
done

# All arguments(as a single word)
for argv in "$*"
do
    echo $argv
done
ubuntu@ubuntu-desktop:/tmp$ bash test_shell.sh apple lemon mango
apple
lemon
mango
apple lemon mango
ubuntu@ubuntu-desktop:/tmp$
```

Lab. Bash Shell Script