Shell Scripting

A script is a comand line program that contains a series of ocmands.

- The commands in the script are executed by an interpreter

```
#!/bin/bash
echo "Scripting is fun!"
```

- First, the sharp + exclamation mark (bang!) are refered as Shebang, inexact contraction of "sharp bang"
- The shebang is followed by the interpreter
- The script is passed as an argument to the interpreter.
- if no shebang is specified on the first line, the script will be executed using the current shell

```
chmod 755 script1.sh
./script1.sh
>> Scripting is fun!
```

```
# Excecutes in the brackground
./sleepy.sh &
ps -fp #pid
```

You don;t have to use a shell as the interpreter for your scripts.

```
#!/usr/bin/python
print "This is a python script."
```

```
chmod 755 hi.py
./hi.py
This is a Python script.
```

Variables

```
VARIABLE_NAME="value"
```

- Do not use spaces before or after the equals sign.

- variables are case sensitive
- By convention, variable are in uppercase
- to use the variable precede the name with a dollar sign

```
MY_VARIABLE="MY_VARIABLE"
echo "This is my variable content $MY_VARIABLE"
```

- To precede or follow the variable with additional data you have to use curly braces

```
"${MY_VARIABLE}"
```

- If you don't encapsulte the variable name in curly braces the shell will treat the additional text as part of the variable name
- Since a vriable with that name does no texist, nothing is put in its place.

Assign Commands to a Variable

To assign the command output to a variable, enclose the comand in parentheses an precede it with a dollar sign

```
#!/bin/bash
SERVER_NAME=$(hostname)
echo "You are running this script on ${SERVER_NAME}."
```

You can also enclose the command in back ticks. This is an older syntax replaced by the $().

Variable Names

- Can contain letters, digits and underscores
- Cannot start with a digit

Tests

To make decisions when the script needs to run, we will need to test for those conditions and have the script act accordingly.

To create a test, place a conditional expression between brackets. The syntax is `[ condition-to-test-for ]`.

This test checks to see if /etc/passwd esists. If it dows, it returns true. The comand exits with a status of 0. If file does not exist, it returns false and the command exits with a status of 1.

```
[ -e /etc/passwd ]
```

In bash shell, you can run the command **help test** to see te various types of tests yoy can perform

```
File operators:
-d FILE     True if file is a directory.
-e FILE     TRUE if file exists.
-f FILE     True if file exists and is a regular file.
-r      True if file is readable by you
-s      True if file exists and is not empty
-w FILE     True if the file is writable by you
-x FILE     True if the file is executable by you

String Operators:
-z STRING       True if string is empty
- STRING        True if string is not empty
STRING      True if string is not empty
STRING1 = STRING2       True if strings are equal
STRING1 != STRING2      True if the strings are not equal

Arithmetic operators
arg1 -eq arg2
arg1 -ne arg2
arg1 -lt arg2  True if arg1 is less than arg2
arg1 -le arg2  less or equal
arg1 -gt arg2       grater than
arg1 -ge    greater or equal
```

If Statement

- The if word is followd by a test.
- The following line contains the word then

if [condition-true] then command1 command2 ... fi

```
#!/bin/bash
MY_SHELL="bash"

if["$MY_SHELL"="bash"]
then
    echo "You seem to like the bash shell."
fi
```

It is a best practice t oenclose variables in quotes to prevent some unexpected effects whe nperforming conditinoal tests.

if/else

```
if [ condition-true]
then
```

```
    command1
    ...
else
    command2
    ...
fi
```

Else if

```
if[ condition-true ]
then
    ...
elif[ condition-true ]
then
    ...
else
    ...
fi
```

For

- start with the word 'for'
- Followd by a variable name
- followed by the word 'in'

```
for VARIABLE_NAME in ITEM_1 ITEM_2 ITEM_N
do
    command1
    command2
    ...
done
```

```
#!/bin/bash
for COLOR in red green blue
do
    echo "COLOR:$COLOR"
done
```

```
COLOR:red
COLOR:green
COLOR:blue
```

```
#!/bin/bash
COLORS="red green blue"

for COLOR in $COLORS
do
    echo "COLOR:$COLOR"
done
```

This shell script renames all of the files thatend in jpg by pretending today's date to the original file name

```
#!/bin/bash
PICTURES=$(ls *jpg)
DATE=$(date+%F)

for PICTURE in $PICTURES
do
    echo "Renaming ${PICTURE} to ${DATE}-${PICTURE}"
    mv ${PICTURE} ${DATE}-${PICTURE}
done
```

Positional Prameters

Positional paremeters are variables that contain the contatents of the command line. These variables are $0 trough $9. The script itself is stored in $0, the first parameter in $1, second in $2 and so on.

Comments

```
# Anything that followd the poung sign
# The only exception to this is the shebang on the first line
```

You can access all the positional parameters starting at $1 to the last one on the command line by using the special variabl $@.

```
#!/bin/bash

echo "Executing script: $0"

for USER in $@
do
    echo "Archiving user: $USER"

    # Lock the account
    passwd -l $USER

    # Create an archive of the home directory.
```

```
    tar cd /archives/${USER}.tar.gz /home/${USER}
done
```

User Input

- STD input use the `read` command.
- `read -p "PROMPT" VARIABLE_NAME`

```
#!/bin/bash

read -p "Enter a user name:" USER

echo "Archiving user: $USER"

# Lock the account
passwd -l $USER

# Create an archive of the home directory
tar cd /archives/${USER}.tar.gz /home/${USER}
```