Shell Scripting

Command Line Shell

- Accessed by using a command line interface
  - Terminal (UNIX/Linux/macOS)
  - Command Prompt (Windows OS)
- Result displayed to the user

Graphical Shell

- Manipulate programs based on (GUI)
- Window OS or Ubuntu OS are good examples

Reasons for Creating Shell Scripts

- Interpreted (non-compiled) language
- Much more portable than compiled language
- Automation to simplify repetitive tasks
- Repeated tasks can be done much faster
- Customize scripts to specific needs
- Make new flexible and configurable tools
- Efficiently execute multiple commands in sequence
- Consistency ensure tasks performed same each time
- Very useful in creating your own commands

What is Shell Scripting?

```
$ echo "Hello World"
Hello World
```

File myhello.sh

```
#!/bin/sh
echo "Hello World!"
```

- Make the file executable

```
$ ./myhello
Hello World!
```

Scripting Files

- File extension .sh (python: .py, C: .c, C++: .cpp)
- First line feeds Command Interpreter
  - `#!` "She-bang", designates a file type
- `#!/bin/sh`
- File must be executable
- `ls -l` gives us information

```
drwxr-x--- 2 mary users 4096 Dec 28 04:09 tmp
-rw-r--r-- 1 mary users 969  Dec 21 02:32 foo
-rwxr-xr-x 1 mary users 345  Sep  1 04:12 somefile
```

File Information

- File type, Permissions, Date, Size, Name
  - Types -, l, c, b, p, s, d
  - User, Group, Other : r (4), w (2), x (1)

```
-rw-r--r-- myfile.sh
chmod 555 myfile.sh   \# (read/execute permission)
-r-xr-xr-x myfile.sh  \# (write protected)
-rw-r--r-- myfile.sh
chmod +rx myfile.sh   \# (read/execute permission)
-rwxr-xr-x myfile.sh
-rw-r--r-- myfile.sh
chmod u+rx myfile.sh  \# (owner permission)
```

Scripting Concepts

- # one line comments only (no multi-line option)
- No whitespace between operands and operator (x=y)

Variables

- Configuration variables : OS Setup
- Environment variables : Your Session `printenv`
- Shell variables : Created at prompt or in scripts

Shell Variables

- Define and manipulate in a program
- Set variable with simple assignment
- Variable value referenced by adding a prefix $
- Variables are handled differently depending on the syntax
- Start with a letter, can contain numbers and underscores
- No declaring variables of type int, char, etc.
- Convention: Use UPPERCASE characters for variables

```
$ VAR1=im1.nii.gz
$ echo $VAR1                im1.nii.gz
$ echo VAR1                 VAR1
$ ls $VAR1                  im1.nii.gz
$ let NUMBER=34
```

{} Braces

- Add string after a variable name, put variable in {}

```
$ V=im1
$ echo $V_new
$ echo ${V}_new
im1_new
```

Backquotes

- Used for evaluating enclosed commands

```
$ V=`ls sub[13]*`
$ echo $V
sub1_t1.nii.gz sub1_t2.nii.gz sub3_pd.nii.gz
```

Single Quotes and Backslash

- To avoid substitutions : \ with special character or '

```
$ VAR1=im1.nii.gz
$ echo $VAR1                im1.nii.gz
$ echo \$VAR1               $VAR1
$ echo '$VAR1'              $VAR1
```

Double Quotes

- Enclosed variables are substituted with their values

```
$ V=Hello World
$ echo $V                   Hello
$ V="Hello World"
$ echo $V                   Hello World
$ echo "*"                  *
$ echo "$V"                 Hello World
```

echo Command

- prints to the screen
- Reference variables prefixed with a $
- Type of quote determine the output

```
$ echo Hello All!          Hello All!
x = 12
echo “variable x = $x”              variable x = 12
echo ‘variable x = $x’              variable x = $x
```

read Command

- to prompt for user input

```
#!/bin/bash
echo “What is your name?”
read NAME
echo “Hello $NAME.”
```

if Command

```
if [ EXPRESSION ] ; then
  COMMANDS ;
else
  COMMANDS2 ;
fi

if [ $a = 2 ] ; then
  b="y-axis";
fi

if [ “$FROM_ADDRESS” = “Wendy” ]
then echo “You have mail from Wendy”
else echo “Wendy has not sent you any mail”
fi
```

Basic Shell Scripting

```
$ mkdir a_folder
$ cd a_folder
$ echo "Hello" Hello
$ for N in John Mary; do for> echo $N for> done John Mary
```

```
#!/bin/sh
mkdir a_folder
cd a_folder
echo "Hello"
for N in John Mary;
do
  echo $N
done
```

Example

```
#!/bin/bash
echo "This is my amazing script!"
echo "Your home dir is: `pwd`"
clear
echo "Today's date is `date`"
echo "You are " `whoami`
echo "These users are currently connected:" `ls -la`
echo "This is `uname -s` on a `uname -m` processor."
echo "This is the uptime information:" `uptime`
echo "That's all folks!"
```

Arithmetic

- add +, subtract -, multiply *, divide /, modulus %

```
let A=4+65
let RAISE=4+$NUMBER
let TOTAL=$GRAND+$NUMBER
OTHER=4+19
```

- expr Command

```
X=$(expr 3 + 4)
X=`expr 3 + 4`
X=$((3 + 4))
```

Note that both ways deal only with integers.

Arithmetic bc Command

```
A=2;
A=`echo "3 * $A + 1" | bc -l`;
echo $A
7
B=`echo "$A > 45" | bc -l`;
echo $B
0
X=10
Y=3
DIFFERENCE=$(echo "$X - $Y" | bc)
# " stops * being used as a wildmark for filenames
PRODUCT=$(echo "$X * $Y" | bc)
echo "Difference: $DIFFERENCE"
echo "Product: $PRODUCT"
Difference: 7
Product: 30
VAR=500
RESULT=$(echo "$VAR % 7" | bc)
3
```

Questions?