# Scripting in BASH

## Writing a script

(More details)

Put the following into a file (e.g. script.sh)

```
#!/bin/bash
echo "Hello World"
# other fun
```

Now, run this command to make it executable

```
chmod u+x script
```

Now run the script like this:

```
./script.sh
```

There is a long-standing convention to leave off the .sh suffix from scripts (e.g. just naming the file script instead of script.sh)  So you will often see a script invoked as:

```
./script
```

just like any other program.

## Variables

Set a variable:

```
export myVar="This is a variable"
```

Read a variable, printing to standard out:

```
echo $myVar
```

```
Increment a Variable
```

```
export =1
```

## Arguments.

Suppose we run our script like this:

```
./script arg1 arg2 arg3
```

And inside the script, we run this command:

```
echo "argument 3 is: $3"
```

This will print:

```
argument 3 is: arg3
```

## Quoting

### Quotes are used to group text

Suppose we run

```
./script "this is a test" "of" "arguments"
```

The quotes group the words "this is a test" into a single argument, so that when we run this script

```
echo "arg1: $1"
echo "arg2: $2"
echo "arg3: $3"
```

will print

```
arg1: this is a test
arg2: of
arg3: arguments
```

### Quotes change the interpretation of escape sequences

In the example from the previous section,
echo `"arg1: $1"`
prints

```
arg1: this is a test
```

but

```
echo 'arg1: $1'
```

prints

arg1: $1

without expanding the $1 with the value of the variable.

Escapes (e.g. `"\\, \n, \r"`) are also useful.

## Conditionals

(More details) (UPDATED link at end of lab)

If  the strings in variables $var1 and $var2 are equal, then print “true”, else print false

```
if test "$var1" = "$var2”; then
```

```
	echo "true"
else
	echo "false"
fi


# If file “$var1” exists…
if test -e "$var1"; then
	echo "true"
else
	echo "false"
fi


# Using if with a command (not needed for this lab, I suppose)
if ./myCommand; then
	# …
else
	# …
fi
```

## Loops

Print out the numbers from one to ten.

```
for i in seq `1 10`; do
	echo $i;
done;
```

Keep looping as long as command succeeds.

```
while command
	echo "looping"
done;
```

Keep looping as long as the command fails.

```
while !command
	echo "looping"
done;
```

Keep looping as long as test is “true”

```
while test …
	echo "looping"
done;
```

## Arithmetic

For integer arithmetic, the built-in $(( )) is quite handy.

(The $ at the start of the lines indicates a command prompt.  You can also use these in a script.)

```
$ echo $((1+1))
2
$ export x=1
$ echo $x
1
$ export x=$(($x+1))
$ echo $x
2
```

For floating-point arithmetic, I usually use bc. I don't anticipate needing to do floating-point arithmetic in scripts in this class.

## Useful commands

`sleep 0.5` will sleep for half a second

`time command args1 arg2 arg3`

will measure how long it takes

`command arg1 arg2 arg3` to run

## Writing Shell Scripts

One of the main resources linked from above:

http://linuxcommand.org/writing_shell_scripts.php

For if, I like this one better

http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html

Intro to variables (was linked from above, not sure if it is useful for this class)

http://www.tldp.org/LDP/abs/html/parameter-substitution.html