Some Unix basics... *Based on some student requests and questions from weeks 1 and 2.*

Writing and Running a shell script

Running a shell script is important. Let's look at some really basic points about Unix shells, file permissions, etc.

1. For consistency's sake, let's make sure the prompt in the terminal window is $. If your prompt is different enter:
   a. `PS1 = "$ "  ;  export PS1`
2. The below example is one of a couple of says of making a script - a set of commands that the operating system can run on its own. Here we'll create a statement that tells the OS to run file (the .sh file) on its own. The `#!` is called *shebang*. The #!/bin/sh tells the OS to feed the contents of the .sh file into its own interpreting program, coincidentally called sh (for shell) that resides in the bin/ folder (or directory)[1]. [Don't type the a., b., etc., nor the $. They're there just to guide. The $ is used below to show you're entering data into the terminal.]
   a. `$ echo '#!/bin/sh' > firstscript.sh`
   b. `$ echo 'echo Hello, World' >> firstscript.sh`
   c. `$ chmod 755 firstscript.sh`
   d. `$ ./firstscript.sh`

File Permissions

Before a file can be used, we set the permissions. Files and folders have three user types and three types of permissions for each of those types. The user types are *owner* (you), *group* (people you share with, such as a "personnelOffice", "techGroup") and the rest of the *world* (or *others*). The permissions are into read (r), write (w), execute (x). Note: there's a shorthand for "all users" (a) for Unix; This isn't the same as the "a" we'll see when writing files that we can use "a" for *append*.

There are two ways of seeing a file's permissions: symbolic and numeric notations.

Symbolic notation:	**Numeric notation:**	**Description:**
`----------`	`0000`	no permissions
`-rwx------`	`0700`	read, write, & execute only for owner
`-rwxrwx---`	`0770`	r, w, x for *o*wner and *g*roup
`-rwxrwxrwx`	`0777`	r, w, x for owner, group, world
`---x--x--x`	`0111`	execute for all
`--w--w--w-`	`0222`	write for all
`-r--r--r--`	`0444`	read for all
`-rw-r--r--`	`0644`	r and w for owner; read for all
`-rwxrw-rw-`	`0755`	r, w, x for owner; rw for group, others

The numeric notation means r = 4 (binary 100), w = 2 (binary 010), and x = 1 (binary 001). So, read (4) + write (2) + execute (1) = 7. We want to write, edit and run the script, so the owner gets 7; everyone else might be allowed to read (4) and write (1) = 5, hence 0755 above.

Changing file permissions

The command chmod changes the [file permission] mode of the file or directory. The syntax is the command (chmod), the permissions, and then the target of the chmod, the file name.

---

1. There are lots of interpreters in the bin/ folder - these include bash (for Bourne again shell, in which case #!/bin/bash instead of /sh), perl5, python, and others.

Some Unix basics... *Based on some student requests and questions from weeks 1 and 2.*

```
chmod 755 firstscript.sh      OR    chmod a+rw firstscript.sh
```

The a means "a" followed by the permissions to be added, r and w.

Sometimes applications, such as Apache web server, will create a file for you but the ownership of the file is not you, it's "apache". Unless you are part of the "apache" group, you may have to change the ownership of the file and do so in your script or program.

Executing a script

The usual syntax is to use ./ to let the OS know the file name that follows is to be executed:

```
$ ./firstscript.sh
```

Standard output, standard input, standard error

Data are accepted from the default stdin (standard input, by default the keyboard), output to the standard output (stdout, the monitor); unless there is an error, then an error message might be recorded in a log file (the stderr, standard error). We can redirect the output and input. The "echo" command echoes or prints the data on the standard output. By using the >> or redirect command, we redirect the output from the standard output to the named target, such as a file name:

```
echo 'echo Hello, World' >> firstscript.sh
```

This command "echos" a string of data, indicated and delimited by the single quote marks, redirecting it to be saved in the firstscript.sh file. Note that we can use << to stream data the other way.

When we read and write to files, we are redirecting the output to a file and redirecting input from the keyboard to a file.

Environmental variables

Most computer users leave their computers set to a default set of behaviors, found in a .bat, .ini, or .sh script. When the computer boots, part of the start-up routine is to locate a script that sets the individual user's computing environment. When you read about changing file paths or adding a path ... this is the type of file you're changing. Here's an example of a Unix environmental variable setup. [Note that some shells use "setevn", some use "export"; some like IBM use a setenv() function. On Mac, use 'export' to see the current environment settings:

```
GBs-iMac:~ gb$ export
declare -x Apple_PubSub_Socket_Render="/private/tmp/com.apple.launchd.2TIs9EDn1H/Render"
declare -x DISPLAY="/private/tmp/com.apple.launchd.1EJLQfhvSP/org.macosforge.xquartz:0"
declare -x HOME="/Users/gb"
declare -x LANG="en_US.UTF-8"
declare -x LOGNAME="gb"
declare -x OLDPWD
declare -x PATH="/Library/Frameworks/Python.framework/Versions/3.7/bin:/anaconda3/bin:/Library/
Frameworks/Python.framework/Versions/3.6/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:/opt/
X11/bin"
declare -x PWD="/Users/gb"
declare -x SHELL="/bin/bash"
declare -x SHLVL="1"
declare -x SSH_AUTH_SOCK="/private/tmp/com.apple.launchd.MFFVXAfqWT/Listeners"
declare -x TERM="xterm-256color"
declare -x TERM_PROGRAM="Apple_Terminal"
declare -x TERM_PROGRAM_VERSION="421"
declare -x TERM_SESSION_ID="11744767-ADE5-4BF2-89A0-A253BFE79B2B"
declare -x TMPDIR="/var/folders/5d/71r9310j2ws92zzdr2rf4rg00000gn/T/"
declare -x USER="gb"
declare -x XPC_FLAGS="0x0"
declare -x XPC_SERVICE_NAME="0"
```

Some Unix basics... *Based on some student requests and questions from weeks 1 and 2.*

We won't be changing the env scripts.

Unix syntax
Unix/linux and even Windows have this syntax for commands:

```
$ command -switches source target
```

The command appears first, followed by a space, optional "switches"; when there appropriate, such as copying a file, we need a source file name and a place to move it, the target. Examples:

`pwd`	print working directory
`ls`	list show (show directory contents)
switches for ls:	-l = long format, permissions, etc.
	-f = do not sort
	-F = appends * or / to show executable and directories
	-a = all files, including "hidden files", . and ..
	-R = recursive - list all files and then all files in sub-folders
	-t = sort by modification t
	-h = print info about files in human-readable terms (e.g., 5K)
	-G = show file names in color
examples:	ls -Fla
	ls -t

File hierarchy:
All operating systems organize files and folders according to an hierarchy. The lowest-level or root of the directory layout is / (a slash, called root). There are then a number of scripts/interpreters that are built into the OS and appear usually in hard-coded names: (bin, dev, etc, home, /usr [some versions of Unix home and usr might be the same; on the Mac there is also /Users], lib, mnt, opt, sbin, tmp and several others.

Since OS are multi-user systems, each individual user's files and applications are in subfolders usually under /usr. There is also /var folder for variables that usually hold log files, mail, and temp files, in /tmp. "tom_jones" is used for the currently logged in end-user name. Return to the home directory by entering `cd ~`

Example:

```
/
   bin/
   lib/
   usr/
      X11/
      bin/
      lib/
      local/
      share/
      sbin/
      standalone/
   Users/
      tom_jones/
         Desktop/
```

```
      Documents/
         myresume.txt
      Applications/
         Adobe/
   /include
   /lib
   /local
   /share
```

Copying, Moving, Deleting files:
Moving a file doesn't necessary delete the file. Note that OS are usually case sensitive: "myresume.txt" is not the same as "MyResume.txt". Avoid having spaces in file names (e.g., "my resume.txt").

Assuming that our current working directory is /Users/tom_jones/Documents, to move a single file from /Documents to an existing subfolder called "workfiles":

```
mv myresume.txt workfiles/myresume.txt
```

To move all the files with the .txt extension:

```
mv *.txt workfiles/
```

Copying the files is similar: cp for copy. Delete? del e.g., del myresume.txt deletes the file but only from the current working directory. Given this file structure:

home		*[this is the root of the file system]*
Documents	*Desktop*	*[these are the same level in hierarchy; siblings]*
myresume.txt		*[this file is in the Documents folder; child]*

to move the Documents/myresume.txt to the Desktop folder, tell the OS to move *up* the hierarchy and then back down to the Desktop folder:

```
mv myresume.txt ../Desktop
```

This command will move the .txt file back up the hierarchy (that's the two dots) and then back down the hierarchy into Desktop.

Strings, Escape Chars and Apostrophes/Quotes:

Recall that some computer languages, such as Java, require that Strings be delimited by double-quotes " ". Other scripting languages, such as JavaScript and python, allow both single quote marks ' ' and double quotes. But they must be used in pairs:

```
echo 'hello, tom'          echo "ni hao, fei"
```

it isn't uncommon to find strings being concatenated with both:

```
x = "Hello, " + 'Tom';
```

If the string includes a ', then "escape" the character (telling the computer to "escape" its usual interpretation of the character): e.g., echo "Welcome to Tom's House." [This might be okay because the single ' is not being matched with anything; it's safely wrapped in the " " of the string.]

Compare to echo `'Welcome to Tom's House'` will not work. The string is delimited from the starting "w" to the "m" in Tom ... leaving the final ' unpaired and out of balance. Use `'Welcome to Tom\'s House'`. The middle ' is now escaped so there is no problem with pairing.

Escape is common, too, for "new lines" and "tabs": `\n` for new line and `\t` for tab.

-end-