Writing shell scripts

If you need help at any time, put your **red** sticky note on the back of your laptop. When you've finished the steps on the *front* of this page, put your **green** sticky note on the back of your laptop. Then, you can turn the page over and try out some of the more advanced tricks on the back while you wait for the rest of the group to be ready.

Creating a shell script

What makes the shell really powerful is your ability to take a complicated workflow, save it in a file, and then re-run those operations later just by running one command.

A shell script is just a file that contains commands, just as we would run them at the terminal. For example, to run the "Hello world" of shell scripts, create a new file called `hello.sh` that contains the following:

```
echo "Hello world"
```

Running a shell script

To run the shell script `hello.sh` that we have just created, we would run

```
bash hello.sh
```

We may prefer to run our scripts as `hello.sh` directly, without having to specify each time that they should be run using the Bash terminal. To do this, we'll add a line called a "shebang" at the top of our script that tells the operating system what program to use to run the script. To write a "shebang", we put the character sequence `#!` followed by the path to the interpreter at the top of our script.

Modify your `hello.sh` script to look like this:

```
#!/bin/bash

echo "Hello world"
```

There's one more step we need to take: we need to mark the script as an executable file with `chmod`. First, verify that the script is *not* marked as executable:

```
ls -l
```

Note the `-rw-r--r--` next to `hello.sh`: this indicates that the file may be **r**ead and **w**ritten by the user that owns it, **r**ead by members of the group that owns it, and **r**ead by all users of the system. (The user and group that own the file are also listed in the `ls -l` output.)

To make it executable, we will run

```
chmod a+x hello.sh
```

to give the e**x**ecutable permission to **a**ll users of the system. Now, the `ls -l` output should show `-rwxr-xr-x`. At this point we can run the script by either running

```
./hello.sh
```

while in the same directory, or by supplying the full path to `hello.sh` from anywhere else in the system.

Leave a script running for later

When working on a remote system, it's useful to be able to run a script and come back to it later.

Try saving the following script:

```
#!/bin/bash

for i in $(seq 100); do
  echo "On iteration $i"
  sleep 30
done
```

This script will run for a long time. Suppose you want to start it and come back to it later?

`screen` is a useful tool for this. Start a new screen session with

```
screen -S "mysession"
```

(passing any name you like to the session). Start your very long script running.

Then, *detach* from your current `screen` session with Ctrl+A+D.

Your script is still running; but you can go on and do other things, or log off and log back on. To resume your session, run

```
screen -ls
```

to list all your sessions. Then run

```
screen -R mysession
```

to re-attach to your running session, by name.