```
$Id: lab2u-unix-shells.mm,v 1.25 2013-09-18 18:02:16-07 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps012b-wm/Labs-cmps012m/lab2u-unix-shells
URL: http://www2.ucsc.edu/courses/cmps012b-wm/:/Labs-cmps012m/lab2u-unix-shells/
```

In this lab, you will become familiar with some features of Unix and its shells. Unix interaction is generally done with either the `bash` shell or the `tcsh` shell.

1. Reading assignment

Following are suggested readings to become familiar with Unix :

(a) ***Your Unix***, by Sumitabha Das, chapters 1–4 and 7. If you have a different Unix book, read the introductory chapters.

(b) ***Unix is a Four Letter Word... and Vi is a Two Letter Abbreviation***, available at :
`http://unix.t-a-y-l-o-r.com/`,
`/afs/cats.ucsc.edu/courses/cmps012b-wm/Tutorials/unix.t-a-y-l-o-r.com`,
`http://www2.ucsc.edu/courses/cmps012b-wm/://Tutorials/unix.t-a-y-l-o-r.com/`.

(c) `http://www2.ucsc.edu/courses/cmps012b-wm/://Tutorials/www.ee.surrey.ac.uk/unixtut/`
is another source for learning Unix.

(d) You should also learn one of the editors `vim` or `emacs`. To learn about `vim`, use the command `vimtutor`, which will take you through a tutorial. For emacs, type the command `emacs &` then click on *Help→Emacs Tutorial.* The non-GUI form of emacs can be started with `emacs -nw`.

(e) The UCSC library has subscribed to Safai Books Online
`http://proquest.safaribooksonline.com/`
so if you go to that URL using a UCSC computer, you can read them for free. Search for "Learning the Unix Operating System".

2. Symbolic links

In order to make references to my directory easier, you might want to establish symbolic links from your directories into mine. For example, the command

```
ln -s /afs/cats.ucsc.edu/courses/cmps012b-wm ~/12b
```

will create a symbolic link so that you can refer to anything in the course volume using the name `~/12b` instead of the whole path.

You could also `cd` into your lab2 directory and make a symbolic link (symlink) to point at my lab2 directory by using `cd` to get to your lab2 directory. Then type the command

```
ln -s ~/12b/Labs-cmps012m/lab2u-unix-shells/ lab2
```

which will allow you to refer to my lab2 files using the short name `lab2`. Example : One of the commands below could then be replaced by

```
grep '^ *Submit:' lab2/*.tt
```

3. Lab exercises

Following are the lab exercises you are to do. For each one submit a file as specified under each point.

(1) In your working directory for lab2, you made a symbolic link above pointing at mine. Type the following two commands :

```
ls -la lab2 >symlink.info
stat lab2 >>symlink.info
```

This will put the output of two commands into a file called `symlink.info`, which should show your symbolic link pointing at my directory.

Submit : `symlink.info`

(2) The `find`(1) command is a useful one for locating files when you know the name or part of it and don't remember which directory it is in. Find all of the files in the course volume that contain the word "`lib`" as part of the filename :

```
cd /afs/cats.ucsc.edu/courses/cmps012b-wm
find . -name '*lib*'
```

Note that many error messages are generated to `stderr` because you don't have permission to access some of my directories. Stop this by redirecting `stderr` :

```
find . -name '*lib*' 2>/dev/null
```

The file `/dev/null` is a pseudo-device which discards bytes written to it.[1] The above command works only with `bash`. Now redirect the output into the file you are to submit :

```
find . -name '*lib*' >~/files.found 2>/dev/null
```

Submit : `files.found`

(3) Another useful command, this time for searching files for a given string, is `grep`(1). Run the command

```
grep '^ *Submit:' *.tt
```

in this lab directory and redirect its output into a file called `grep.submit` in your directory.

Submit : `grep.submit`

(4) These commands can be combined using a pipe and `xargs`(1). Use a command to find all files anywhere in the course volume

```
/afs/cats.ucsc.edu/courses/cmps012b-wm
```

that match the wildcard `*.java` and pipe this into the command

```
xargs grep -li lib
```

The pipe is the stick character (`|`). Redirect `stdout` into a file called `java.libs`.

Submit : `java.libs`

(5) Write a program in Java that will print the message "`EXIT 1`" to the standard error and then exit with a status of 1.

Submit : `exit1.java`

(6) Write a shell script called `mkexit1` which compiles that program and puts it in a jar file. The first line of the script must be

```
#!/bin/sh -x
```

The second line should be an RCS Id string :

```
# $Id$
```

The third line should be your name and username, along with those of your partner, if any. Also put the RCS Id string in your java program from the previous part. Type the following commands at a terminal and note that they build a jar :

```
javac exit1.java
echo Main-class: exit1 >Manifest
jar cvfm exit1 Manifest exit1.class
yes | rm -i Manifest exit1.class
chmod +x exit1
```

Put all of them in the script file `mkexit1` Use `chmod +x mkexit1` to make it executable. Run the script by typing its name at the command line :

```
mkexit1
exit1 >/dev/null
echo $?
```

Note that you still see the error message. The last of these command shows you the exit status.

Submit : `mkexit1`

(7) Verify that your `private` directory is properly protected. Use either of the commands

```
fs la $HOME/private,
fs la ~/private,
```

and redirect its output into a file called `privacy`. Append to this file the output from the command `echo $HOME`.

1. It's the bit bucket. Empty it when full. :-)

Submit: `privacy`

(8) Make a copy of `exit1.java`, calling it `exit255.java`, and have it return an exit status of 255. Also change the message that it prints to "`EXIT 255`". Create a `Makefile` with a target `all`, which depends on the jar file `exit255`. The jar target `exit255` depends on `exit255.class`, which, in turn, depends on `exit255.java`. Add commands under the last two dependencies such that it builds the jar.

Submit: `exit255.java`

(9) To this same `Makefile`, add a target called `test`, which runs the jar `exit255`, redirecting its output into a file called `255.output`. It then appends the value of the exit status to that same file. Do not submit `255.output`, the grader will create it via `gmake test`.

Since the exit status is in the shell variable `$?`, to code that in a `Makefile`, you need to enter it as `$$?`. Also, each line of a `Makefile` is run as a separate process, so the line that echos the exit status must be on the same line as the program that is being run, separated by a semi-colon.

Submit: `Makefile`

(10) Check your quota and find all files larger than 1000 blocks in your file space. The command `fs lq ~` will tell you about your disk quota. The command `find ~ -size +1000` will print the names of all your files larger than 1000 blocks. Redirect the output of both of these commands into a file called `quota.size`.

Submit: `quota.size`

4. What to submit

You were instructed to submit many files. Verify each of them by looking in the submit directory. If you are in doubt as to the contents of a file you submitted, submit it again. You may submit any file as many times as you want, as long as you do it before the due date.

If you are doing pair programming, also submit `README` and `PARTNER`. Run `partnercheck` to verify that your `PARTNER` file is correct.

Look in the subdirectory `.score` for instructions to the graders.

Read the submit checklist file in the generic syllabus directory.

2. With apologies to Marcus Porcius Cato Maior (DXX–DCV AUC): ‹ *Praeterea, censeo Microflaccidem esse delendam.* ›