```
$Id: lab1u-shells-etc.mm,v 1.50 2015-01-08 18:54:56-08 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps012b-wm/Labs-cmps012m/lab1u-shells-etc
URL: http://www2.ucsc.edu/courses/cmps012b-wm/:/Labs-cmps012m/lab1u-shells-etc/
```

1. Overview

This lab will continue your introduction to Unix. We will look at environment variables, controlling the shell with **.bashrc**, and backing up files using the **RCS** utilities.

2. The $PATH variable

There are some commands given in this lab which are not generally available Unix commands. These are **cid** and **checksource**. You will notice that you get an error message when you use them :

```
bash-01$ cid hello.java
bash: cid: command not found
```

This is because they are not in your path. These commands, among many others, are in the directory **/afs/cats.ucsc.edu/courses/cmps012b-wm/bin/**. You should add this directory to your path. How this is done depends on which shell you are using. We will discuss only **bash** in this document and ignore the others.

The path to the course **bin** directory can be added to your path with the command :

```
bash-02$ export PATH=$PATH:/afs/cats.ucsc.edu/courses/cmps012b-wm/bin
```

This command should be added to your **.bashrc** file.

3. Standard output, standard error, and exit status

Normal output is written to the standard output, which is **stdout** in C, **System.out** in Java, and file descriptor 1 in the shell. Error messages and other out of band notes are written to the standard error, which is **stderr** in C, **System.err** in Java, and file descriptor 2 in the shell. When a program returns to its caller, it provides an exit status integer between 0 and 255. Success is indicated with 0, and any other number indicates failure.

Write a Java program called **hello.java** with the following specifications :

(a) If it is given no command line arguments (in **args**) it prints the message
Hello, World!
to the standard output and exits with status 0.

(b) If it is given any command line arguments, it prints the message
Usage: hello
to the standard error and exits with status code 1.

4. Putting Java in a jar

Java programs can be run from the command line using the **java** command directly, but the usual practice is to put class files in a jar file so that they can be executed like any other Unix command.

This is done via the **jar** command, which requires a **Manifest** file to tell the jar file exec which class is the main class.

```
bash-03$ echo Main-class: hello >Manifest
bash-04$ jar cvfm hello Manifest hello.class
bash-05$ chmod +x hello
```

The `echo` command redirects output to the `Manifest` file (note the capital M). Then the `hello` jar is created and the `x`-bit is turned on.

5. Running a jar file

A jar file can just be run using the name of the file, without having to specify that it is a Java program. It can be specified by explicit path name, as here, or by the name of the program being looked up in the `$PATH` variable.

```
bash-06$ ./hello
Hello, World!
bash-07$ echo $?
0
bash-08$ ./hello world
Usage: hello
bash-09$ echo $?
1
```

6. The script `checksource`

Use the script `checksource` to check on some basic formatting items. Edit your files so that it does not complain. If you run `checksource` without filename operands, it will print out a text-format manual page. To check up on `hello.java`, use the command:

```
bash-10$ checksource hello.java
```

Tab characters should ***only*** be used in a `Makefile`. Otherwise indentation should be done with a few spaces. If you edit on a M*cr*$*ft system, make sure to eliminate the carriage turn charcters from the file when copying to Unix. And do not make lines longer than 72 characters.

Carriage return characters can be eliminated with the following `vim` line mode command:

```
:g/^V^M/s///g
```

Do not actually type the circumflex (`^`) here. The `^`-notation means hold down the Control key. So `^V^M` means type the letters "vm" while holding down the Control key. Or use the `bash` script `elimcr` (see Figure 2) in the course `bin` directory.

7. `RCS`

It is a good idea to keep many backup copies of your work. `RCS` is a good utility to keep track of backup copies. If you don't have backups, you will have to depend on the IT department to recover yesterday's copy. Murphy's law says that the most important changes won't be in that copy. The corollary also says that you will lose your files so close to the due date that IC won't get your backups back to you in time. In this case, you will get a 0 on the assignment for not submitting anything.

Note that the code above has a magic string in it: `$Id$`. These track your development progress. For the initial checkin, do the following:

```
bash-11$ mkdir RCS
bash-12$ ci -zLT -s- -t- -m- -u hello.java
RCS/hello.java,v <-- hello.java
initial revision: 1.1
done
```

Now you have your initial version. Look at the **man** page for **ci** for all of the options. You will also want to read the **co** page. The options were: **-zLT** causes the time stamp to be in local time instead of UTC, **-s-** sets the state to –, **-t-** suppresses the descriptive text, **-m-** suppresses the log message, and **-u** checks in the file unlocked, thus not destroying the source.

Unfortunately, the file is now read-only, so you may want to make locking non-strict:

```
bash-13$ rcs -U hello.java
RCS file: RCS/hello.java,v
done
bash-14$ chmod u+w hello.java
bash-15$ ls -la hello.java
-rw-r--r-- 1 foobar user 465 Sep 14 18:42 hello.java
```

Oops, you forgot to put your name and username at the top of the file. Edit the comment on the first line to reflect your name and username. Every file you submit must have a comment on the first line with your name and username in it. Add in your name and username. Now check in another copy to make a backup.

```
bash-16$ ci -zLT -s- -t- -m- -u hello.java
RCS/hello.java,v <-- hello.java
new revision: 1.2; previous revision: 1.1
done
```

Use **cat** to look at the new version of your file.

There are some alternatives to **RCS**: **SCCS** (very old). **CVS** (more flexible but more complicated). **SVN** (some people like using this). There are also some others.

8. Recovering lost files

If you are keeping files in an **RCS** subdirectory, you may recover them using the **co** command. For example

```
bash-17$ co -r1.9 hello.java
```

will recover version 1.9 of the file **hello.java** from the archive.

To see what versions of **hello.java** you have in the archive, use the command

```
bash-18$ rlog hello.java
```

If you want to see the differences between, say, versions 1.7 and 1.11, use the command

```
bash-19$ rcsdiff -r1.7 -r1.11 hello.java
```

Whenever you create a new file, either the first or last line should be a comment with the **Id** code in it, as in

```
// $Id$
```

After doing this, a check-in will automatically edit it to something like

```
// $Id: hello.java,v 1.1 2015-01-06 18:02:07-08 - - $
```

which shows the name of the file, the veersion, and the date and time of check-in. The "-08" at the end of the time indicates the number of hours west of Greenwich Mean Time (GMT), aka Universal Time Coördinated (UTC).

9. The script `cid`

An alternative to using **ci** (see below) directly is the program **cid**. It works just like **ci**, but automatically creates the **RCS** subdirectory and does the correct locking. To fetch back a deleted file, use the **co** command. You will find that the **cid** command is much simpler to use, since it automatically sets up the **RCS** subdirectory and appropriate file locking.

In order to find where that script is, you can do the following:

```
bash-20$ cd /afs/cats.ucsc.edu/courses/cmps012b-wm
bash-21$ find * -name cid -follow 2>/dev/null
```

This says find all files whose name is **cid**, even if you have to follow symbolic links. Without the redirection **2>/dev/null**, you will get lots of error messages because of directories that you don't have permission to access. With this redirection, error messages will be sent to **/dev/null**, the bit bucket. Try it both ways, with and without redirecting **stderr**.

10. Lab exercises

Each of the following items specifies something that must be done for credit in this lab.

(1) Write the program **hello.java** as described in section 3. Note that neither the class name nor the file name have upper-case letters. Then check it into an RCS subdirectory:

```
bash-22$ cid hello.java
```

Then edit the file and insert the following line as the first line in the file:

```
// $Id$
```

Then check it in again:

```
bash-23$ cid hello.java
```

Now add your name and username immediately after the RCS Id line, and check it in again. Notice that it now has version 1.3 of the file, or later, if you have edited it several more times.

Submit: **hello.java**

(2) Write a shell script for **bash** called **mkhello**. Make the first two lines of the file as follows:

```
#!/bin/bash
# $Id$
```

and insert your name and username as the third line. After that add the necessary **bash** commands to do the following (in the order specified here):

(i) Use the **cid** command to check **hello.java** into the RCS subdirectory.

(ii) Compile **hello.java** into **hello.class**.

(iii) Create the **Manifest** file.

(iv) Put the **Manifest** and the class file into a jar file called **hello**.

(v) Turn on the **x**-bit to make it executable.

(vi) Remove the **Manifest** and **hello.class** files.

See section 4 for details of some of these commands. Check the script **mkhello** into the RCS subdirectory.

Submit: **mkhello**

(3) Write another shell script called **testhello**. Make sure the hashbang (**#!**) is the first line and the RCS Id is the second line. This script should create 6 files:

(a) When **hello** is run without arguments, it creates **test1.out**, **test1.err**, and **test1.status**.

(b) When **hello** is run with the argument **world** it creates **test2.out**, **test2.err**, and **test2.status**.

The files with suffix **.out** should capture standard output; the files with suffix **.err** should capture standard error; and the files with suffix **status** should have the exit status values from **bash**'s variable **$?**.

Submit: **testhello**

(4) When programs are in nonstandard places, the **$PATH** environment variable addes places to look for programs and scripts. To ensure that **cid** and **checksource** are available, add the following to **~/.bashrc**:

```
export cmps012b=/afs/cats.ucsc.edu/courses/cmps012b-wm
export PATH=$PATH:$cmps012b/bin
```

Submit: **~/.bashrc**

(5) Using an alias is a useful way to avoid typing in long commands and pathnames. Add the following lines to **~/.bashrc**:

```
alias 0='cd $cmps012b'
alias 0a='cd $cmps012b/Assignments'
alias 0m='cd $cmps012b/Labs-cmps012m'
```

After sourcing **~/.bashrc**, you can use the commands **0**, **0a**, and **0m** to quickly navigate to three of my directories. Perhaps you wish to add a few more aliases. Note that there are no spaces before or after the equal sign.

Submit: **~/.bashrc**

(6) Another way to refer to files quickly is with a symbolic link. A symbolic link is just a file named somewhere convenient which points at another file. Type the following:

```
bash-24$ ln -s /afs/cats.ucsc.edu/courses/cmps012b-wm/ ~/12b
```

This creates the link **12b** in your home directory, so no you can get to the 12B directory with PROMPT "cd ~/12b" Symbolic links remain as files unless/until you delete them. The commands **ls**(1) and **stat**(1) can be used to find information about files. Create a file **links**:

```
bash-25$ ls -la ~/12b >links
bash-26$ stat ~/12b >>links
```

Note that "`>`" is a redirect to create a file or replace it, and "`>>`" appends to a file.

Submit: **`links`**

(7) Are you sure you submitted all required files? You may submit files more than once before the due date. In the directory containing the **`.tt`** version of this file, type the command:

```
bash-27$ grep Submit: *.tt
```

This will list all of the lines with the string "**`Submit:`**". Cut and paste the output of this command into a file called **`submits`**.

Submit: **`submits`**

11. Submit checklist

Carefully review the submit checklist:
`/afs/cats.ucsc.edu/courses/cmps012b-wm/Syllabus/submit-checklist/`
`http://www2.ucsc.edu/courses/cmps012b-wm/://Syllabus/submit-checklist/`

The subdirectory **`.score`** has instructions to the grader, along with a script that the grader will run.

```
 1  #!/bin/bash
 2  # $Id: elimcr,v 1.7 2015-01-07 16:37:32-08 - - $
 3  #
 4  # NAME
 5  #    elimcr - fix basic formatting in a text file
 6  #
 7  # SYNOPSIS
 8  #    elimcr filename...
 9  #
10  # DESCRIPTION
11  #    Backs up the original file.
12  #    Fixes basic formatting on a text file:
13  #    - Expands tabs using expand(1), except for Makefile.
14  #    - Deletes trailing carriage return characters.
15  #    - Adds a final newline if missing from the file.
16  #    Does not fix lines longer than 72 characters.
17  #
18
19  exim='/usr/bin/vim -E'
20
21  if [ $# -eq 0 ]
22  then
23     grep '^#' $0
24     exit 1
25  fi
26
27  for file in $*
28  do
29     if /bin/cp $file $file~~
30     then
31        { [[ $file != *Makefile ]] && echo '1,$!expand'
32          echo 'g/\r/s///'
33          echo w
34          echo q
35        } | $exim $file
36        echo $exim status: $?
37        ls -goa $file $file~~
38     fi
39  done
40
```

Figure 1. `bin/elimcr`

```
 1  #!/bin/bash
 2  # $Id: bashrc.example,v 1.1 2015-01-08 18:31:57-08 - - $
 3
 4  export cmps012b=/afs/cats.ucsc.edu/courses/cmps012b-wm
 5  export submit012b=/afs/cats.ucsc.edu/class/cmps012b-wm.s13
 6
 7  export EDITOR=vim
 8  export MANPAGER=more
 9  export MANWIDTH=72
10  export PATH=$PATH:$cmps012b/bin
11  export SHELL=/bin/bash
12  export VISUAL=vim
13
14  export PS1='\s-\!\$ '
15  set -o ignoreeof
16  set -o noclobber
17  set -o physical
18  unset HISTFILE
19
20  alias cp='cp -i'
21  alias grind='valgrind --leak-check=full --show-reachable=yes'
22  alias m='more'
23  alias mv='mv -i'
24  alias rm='rm -i'
25
26  alias 0='cd $cmps012b'
27  alias 0a='cd $cmps012b/Assignments'
28  alias 0m='cd $cmps012b/Labs-cmps012m'
29
30  alias la='ls -la'
31  alias lf='ls -Fa'
32  alias ll='ls -goa'
33  alias llh='ls -goah'
34  alias llr='ls -goaR'
35  alias lls='ls -goaSr'
36  alias llt='ls -goatr'
37  unalias ls 2>/dev/null
38
```

Figure 2. Example **.bashrc**