$Id: lab1u-gmake-rcs.mm,v 1.120 2014-03-27 17:49:47-07 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps012b-wm/Labs-cmps012m/lab1u-gmake-rcs
URL: http://www2.ucsc.edu/courses/cmps012b-wm/:/Labs-cmps012m/lab1u-gmake-rcs/

## 1. Overview

This lab will introduce you to Unix if you are not already familiar with it, and show you how to compile and execute a Java program. It also helps you with a first pass at understanding:

- **`AFS`**, the file system where your files are kept, and access control lists;
- **`RCS`**, the revision control system, which helps you keep backup copies of all of your work;
- **`gmake`**, which helps you with building executables from source code; and
- automatically setting the **`$PATH`** variable by means of dot files in your home directory.

## 2. Directories and ACLs

Change directory to your home directory and create a subdirectory called **`private`**. Then make a subdirectory of that called **`cmps012m`**, and a subdirectory of that called **`lab1`**. In the following, bolface is what you type, and plain face is what is typed at you:

```
bash-1$ cd
bash-2$ mkdir private
```

Make sure you protect the directory against other users:

```
bash-3$ fs setacl private $USER all -clear
bash-4$ fs listacl private
Access list for private is
Normal rights:
  foobar rlidwka
```

At this point, there should be only one ACL on your private directory, namely your own, where **`foobar`** is shown above should be shown your own username. The shell variable **`$USER`** is your own username. You may wish to type it in explicitly instead. If there are any other ACLs on your private directory, you must delete them. Note that you may also lock your home directory to all ACLs but your own. That means, however, that you may not have a web page.

You must understand ACLs in order to protect your files. Some special ACL usernames are: **`wwwadmin`** is the IC web server, which needs read permission on your directory **`public_html`**, if you have one, and list permission on your home directory. **`system:authuser`** is anybody with a IC account. **`system:anyuser`** is anybody anywhere in the world on any machine running AFS. Try **`ls /afs`**.

You should have a separate directory for each course and each assignment. So create a directory using the following:

```
bash-5$ cd $HOME/private
bash-6$ mkdir -p cmps012b/lab1
bash-7$ cd cmps012b/lab1
```

The shell variable **`$HOME`** may also be replaced by a tilde (**`~`**) as the name of your

home directory. When you use **`cd`** without an operand, it sets the current directory to your home directory. You may wish to use two separate course directories, one for your CMPS-012B work, and one for your CMPS-012M work, or you might want to use just one. You may, of course, call them anything you like. In order to organize your files, make a separate subdirectory for each course, and under that, a subdirectory for each project in each course.

### 3. `Hello.java`

Now begin work on lab 1. Create a file called **`hello.java`**, as shown in Figure 2 (page 8). It is a slightly more complicated version of the standard "Hello World" program used to introduce programming languages.

If you use the command **`ls -la`** in my directory, you might notice a subdirectory containing that code. If you like, you may copy it into your lab directory instead of copying it in. Files beginning with a dot are not listed with **`ls`**(1) unless the **`-a`** flag is given.

### 4. 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-8$ 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. The file **`/etc/shells`** lists all of the ones available: **`/bin/bash`**, **`/bin/csh`**, **`/bin/ksh`**, **`/bin/mksh`**, **`/bin/sh`**, **`/bin/tcsh`**, **`/bin/zsh`**. We will discuss only **`tcsh`** and **`bash`** in this document and ignore the others. Since we are using Linux, **`/bin/sh`** does not exist, and is a symbolic link to **`/bin/bash`**. Also, **`/bin/csh`** does not exist, and is a symbolic link to **`/bin/tcsh`**. To find out which shell is your default, use the command:

```
echo $SHELL
```

Create or modify three files: **`~/.cshrc`**, **`~/.bashrc`**, and **`~/.bash_profile`**. Modify your path for both shells as follows:

(a) For **`tcsh`**, add the following line to the end of your **`~/.cshrc`** file:

```
set path=($path /afs/cats.ucsc.edu/courses/cmps012b-wm/bin)
```

Then source it with the command:

```
source ~/.cshrc
```

For this command to work, you must be interacting with **`tcsh`**.

(b) For **`bash`**, add the following line to the end of your **`~/.bashrc`** file:

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

Then source it with the command:

```
source ~/.bashrc
```

For this command to work, you must be interacting with **`bash`**. If your current shell is **`tcsh`**, type the command **`bash`** before running the above command.

Whenever you start **bash** as a subshell, it will automatically read the contents of **.bashrc**.

Since **bash** differentiates between a login shell and a subshell, You should also have the file **~/.bash_profile** source **~/.bashrc**. This is done by putting the following line in your **~/.bash_profile** file:

```
source ~/.bashrc
```

The sourcing command does not need to be typed in every time you log in. Shells source their start files automatically. Any command or alias you want executed automatically every time you log in should be placed in the appropriate startup file. The last two letters, "**rc**" mean "run commands".

If your shell is currently **tcsh** and you want to use bash, just use the command **bash** at the prompt. Or follow the instructions printed by the command **chsh** if you want to make this permanent.

When you log in using **tcsh**, the file **~/.cshrc** is automatically sourced, followed by **~/.login**, which you probably may ignore. When you start **bash** by typing in the command at the command line, the file **~/.bashrc** is automatically sourced. But if **bash** is your login shell, then at login, **~/.bash_profile** is automatically sourced.

Submit your files **.cshrc**, **.bashrc**, and **.bash_profile**, using the command:

```
submit cmps012b-wm.s14 lab1 .cshrc .bashrc .bash_profile
```

This command must be done in your home directory.

### 5. 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-9$ checksource hello.java
```

### 6. 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-10$ cd /afs/cats.ucsc.edu/courses/cmps012b-wm
bash-11$ 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**. This works if you use **bash** or **sh** as your shell, If you use **csh**, you should instead use

```
csh-% sh -c 'find * -name cid -follow 2>/dev/null'
```

which will pass the command to the Bourne shell to have it executed. There are many small differences between the shells. Inside of a **`Makefile`**, always use Bourne shell syntax, and never use the born-again shell syntax or **`csh`** syntax. Read **`http://www.faqs.org/faqs/unix-faq/shell/`**. You may find something of interest in **`http://www.faqs.org/faqs/unix-faq/shell/csh-whynot/`**.

## 7. Running the program

The output of running the program is shown in Figure 1. The figure was actually generated from the **`gmake`** process that created this document. A Unix pipe was used. If you don't want to type in the entire "hello" program, you may copy it from the hidden (or "dot") subdirectory. This directory will not show up if you are using a browser. Log into **`unix.ucsc.edu`** and find it with **`ls -la`**.

```
     1  Hello, Java World!
     2  java.class.path = /afs/cats.ucsc.edu/courses/cmps012b-wm/Labs-cmps
012m/lab1u-gmake-rcs/.files/hello
     3  sun.arch.data.model = 64
     4  sun.cpu.endian = little
     5  os.arch = amd64
     6  os.name = Linux
     7  os.version = 2.6.32-358.23.2.el6.x86_64
     8  java.runtime.name = OpenJDK Runtime Environment
     9  java.runtime.version = 1.7.0_45-mockbuild_2013_10_23_08_18-b00
    10  java.vm.name = OpenJDK 64-Bit Server VM
    11  java.vm.version = 24.45-b08
    12  java.home = /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.45.x86_64/jre
    13  java.version = 1.7.0_45
    14  free memory = 123081744 bytes = 117.3799 megabytes
    15  current time = 1395967782.787 sec = Thu Mar 27 17:49:42 PDT 2014
    16  EXIT STATUS = 0
```

**Figure 1.** **`.files/test.output`**

## 8. `Makefile`

Typing in all of these commands repeatedly is a hassle. To avoid that, we use a program called **`gmake`**. Create a file called **`Makefile`**, as shown in Figure 3 (page 9).

Make sure that indented commands in a **`Makefile`** are indented with a TAB character, not spaces. If you mistakenly use spaces to indent commands in a **`Makefile`**, you will get an error message. Proper capitalization is also important. Now you can make, check in, and submit your program as follows:

```
bash-12$ rm hello.class
bash-13$ gmake ci
bash-14$ gmake all
bash-15$ gmake submit
```

To submit files directly from the command line without using the `Makefile`, the following command may be used:

```
submit cmps012b-wm.s14 lab1 README Makefile hello.java
```

If you are doing pair programming, also submit the file `PARTNER`. The first argument to submit is the name of the class volume, the second argument is the name of the project, and the rest are names of files to be submitted.

### 9. Verifying the submit

To check on the files you submitted, look in the directory

```
/afs/cats.ucsc.edu/class/cmps012b-wm.s14/lab1/$USER
```

You won't actually be able to read the files, but you can verify the filenames. Check to make sure that you have submitted all the files that should be submitted.

In order to verify that your submit actually worked, create a new directory in your personal file space, or choose an existing test submit directory and use `rm` to delete any existing files in that. Then copy the files you submitted from your working directory into the new directory. Do not copy other files. Did everything work as expected?

You may use any system to which you have access as your development system. However, you ***must*** test your program on one of the Linux servers or lab workstations prior to submitting it. ***Graders will only use*** `unix.ucsc.edu` to do the grading.

### 10. Automating checkin and submit

You will note that the `Makefile` has two targets `ci` and `submit`. The purpose of `make` is to automate routine boring tasks. So try checking in everything all at once and then submitting it:

```
bash-16$ gmake ci
bash-17$ gmake submit
```

Note that when this is automated, you don't run the risk of forgetting a file when you submit.

### 11. WARNING

The time to figure out how to do it is ***NOT*** the day the first real assignment is due. Excuses sent late on the due date will be rejected and your assignment will score ***ZERO***. There is no excuse for not knowing how to use submit and/or the Unix workarounds if the script does not work. Assignments submitted via email will not be accepted. They only way an assignment will be submitted is via Unix on or before the due date. You may submit an assignment as many times as you want ***BEFORE*** the due date in order to ensure that something is submitted if there are last minute problems.

### 12. Miscellaneous

This section applies to ***all*** labs and programs submitted during the quarter. If you choose to do pair programming, read the requirements in the subdirectory of the syllabus.

The **README** file should contain your name and username, the name of the host you used to do the development, and any other necessary comments, such as the source of any code you did not write yourself. To find out the name of the host you are currently using, use the command **hostname**.

You must also follow some basic source code formatting requirements that are checked up on by the script **checksource**. You can locate it in the same manner as you did **cid** above. ***Reading assignment:*** **/afs/cats.ucsc.edu/courses/cmps012b-wm/Coding-style/**.

### 13. Prerequisites: Java and Unix

This course assumes you have had experience with Java and Unix. If not, you should read the first seven chapters of ***Java by Dissection*** [**http://www.lulu.com/javabydissection**] and begin working with Unix (you need a Unix reference book). This paragraph is just a slight roadmap to bring up your Java skills if you have had a prerequisite course that used C or C++, or used some operating system other than Unix.

UCSC has a subscription to Safari Books Online [**http://proquest.safaribooksonline.com/**], which is maintained by the O'Reilly publishers [**http://oreilly.com/**]. You may read their books online for free, provided you use a UCSC computer. Recommended readings: "*Learning the Unix Operating System*", "*Managing Projects with GNU Make*", "*Java in a Nutshell*".

### 14. Pair programming

If you are doing pair programming, carefully read over the summary in **cmps012b-wm/Syllabus/pair-programming**. Note especially carefully the format of the **PARTNER** file. Use the **partnercheck** script to check your **PARTNER** file before submitting it.

### 15. What to submit

Submit the files **.cshrc**, **.bashrc**, **.bash_profile**, **README**, **Makefile**, and **hello.java**.

There is a subdirectory called **.score** which contains instructions to the grader. Note that because it begins with a dot, it is "hidden", and so will not show up as part of the output of the **ls** command, unless the **-a** option is used. Similarly, there is another directory called **.files**, which is also hidden. Neither of these show up if you are using a browser.

Do ***not*** submit any of the files generated by the script in this directory. However, you may want to do a pre-grade in a private directory of your own in order to guess at what score you might receive. Look for a similar directory in future labs and programs.

### 16. More on rc (dot) files

Figure 4 (page 10) shows some commands that you might want to incorporate into your **~/.bashrc** file. They show how to customize the environment, make new convenience commands available, customize the prompt, etc. The **man**(1) pages for **bash**(1) and **tcsh**(1) will provide more information and details. This section is informational only and is not part of this lab.

If you are not sure which shell you prefer using, and your default is **`/bin/tcsh`**, you can switch to **`bash`** temporarily just by typing the command **`bash`** at the command prompt. To permanently make the change, use the command **`chsh`**.

My suggestion is that you make **`bash`** your default shell, unless you have a strong preference for **`tcsh`**. Shell scripts are almost exclusively written in **`bash`** or **`sh`**, and **`make`** always calls **`/bin/sh`** to process its commands. So if your interactive shell is **`bash`**, you only need to know one shell. And there are a few things that are difficult to do in **`tcsh`**.

Refer to Tom Christiansen's article "Csh Programming Considered Harmful":
**`http://www.faqs.org/faqs/unix-faq/shell/csh-whynot/`**

### 17. `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-18$ mkdir RCS
bash-19$ 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-20$ rcs -U hello.java
RCS file: RCS/hello.java,v
done
bash-21$ chmod u+w hello.java
bash-22$ 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-23$ 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.

### 18. Recovering lost files

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

```
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

```
rlog hello.java
```

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

```
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 1.12 2014-03-24 17:32:36-07 - - $
```

which shows the name of the file, the veersion, and the date and time of check-in. The "-07" at the end of the time indicates the number of hours west of Greenwich Mean Time (GMT), aka Universal Time Coördinated (UTC).

```
 1  // $Id: hello.java,v 1.12 2014-03-24 17:32:36-07 - - $
 2  //
 3  // NAME
 4  //    hello - a Java version of the classical "Hello World" program.
 5  //
 6  // DESCRIPTION
 7  //    Introduces itself and its environment.
 8  //
 9
10  import java.util.ArrayList;
11  import java.util.List;
12  import static java.lang.System.*;
13  class hello {
14     static String[] properties = {
15        "java.class.path",
16        "sun.arch.data.model",
17        "sun.cpu.endian",
18        "os.arch",
19        "os.name",
20        "os.version",
21        "java.runtime.name",
22        "java.runtime.version",
23        "java.vm.name",
24        "java.vm.version",
25        "java.home",
26        "java.version",
27     };
28     public static void main (String[] args) {
29        out.printf ("Hello, Java World!%n");
30        for (String property: properties) {
31           String property_value = getProperty (property);
32           out.printf ("%s = %s%n", property, property_value);
33        }
34        long free_memory = Runtime.getRuntime().freeMemory();
35        out.printf ("free memory = %d bytes = %.4f megabytes%n",
36                    free_memory, free_memory / (double)(1<<20));
37        long now = currentTimeMillis();
38        out.printf ("current time = %.3f sec = %tc%n",
39                    now / 1e3, now);
40     }
41  }
```

**Figure 2.** `.files/hello.java`

```
 1  # $Id: Makefile,v 1.11 2014-03-24 16:59:53-07 - - $
 2
 3  JAVASRC    = hello.java
 4  SOURCES    = README Makefile ${JAVASRC}
 5  MAINCLASS  = hello
 6  CLASSES    = hello.class
 7  JARFILE    = hello
 8  JARCLASSES = ${CLASSES}
 9
10  all: ${JARFILE}
11
12  ${JARFILE}: ${CLASSES}
13          echo Main-class: ${MAINCLASS} >Manifest
14          jar cvfm ${JARFILE} Manifest ${JARCLASSES}
15          - rm -vf Manifest
16          chmod +x ${JARFILE}
17
18  %.class: %.java
19          javac $<
20
21  clean:
22          - rm -vf ${CLASSES} test.output
23
24  spotless: clean
25          - rm -vf ${JARFILE}
26
27  ci: ${SOURCES}
28          cid + ${SOURCES}
29          - checksource ${SOURCES}
30
31  test: ${JARFILE}
32          (${JARFILE} 2>&1; echo EXIT STATUS = $$?) >test.output
33          cat -nv test.output
34
35  again:
36          gmake --no-print-directory spotless ci all test
37
```

**Figure 3.** **`.files/Makefile`**

```
 1  #!/bin/bash
 2  # $Id: bashrc.sh,v 1.16 2014-03-24 17:34:25-07 - - $
 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 4.** `bashrc.sh`