CS 455 Lab 1: Using the Vocareum Environment and Linux

Spring 2019 [Bono]

Goals and Background

The goals of this lab are to familiarize yourself with the Vocareum cloud-based programming environment, the Linux shell, and compiling and running simple Java programs. You'll be using Vocareum this semester to get your assignment and lab code files, submit your assignment files, and do your code development (optionally).

Vocareum has a simple IDE (Integrated Development Environment) that you must use for this lab. For later assignments, you may want to switch to another IDE that runs on your local machine, such as Eclipse.

There are Windows computers available in the lab classrooms. However, you may bring your own laptop to the lab if you have one.

Resources:

- Horstmann, Ch. 1

General information

You will earn up to 4 "checkoff points" in this lab section by showing your lab T.A. that you have successfully completed the various tasks in the assignment. In general, it is better to get each point checked-off as you go, rather than waiting until the end of the lab. For on-campus students check-offs will not be done outside of lab.

Lab pairs

Labs will generally be done working in partnerships (2 people). This is a version of the programming technique called Pair Programming. The pair of you will sit at one workstation. One person will be the designated driver, doing the typing and mousing to use the tools, and the other will be the scribe, writing down the answers to lab questions. In subsequent labs you will switch off roles. This does *not* mean only one person is doing the work. In pair programming both members of the team are active participants in solving the problems, discussing as you go, but with each with a designated role.

However, for this lab, because it's primarily focused on familiarizing yourself with the programming environment, we recommend you *each* do all the exercises (we'll say more about that in Exercise 1) but you may still collaborate on a single answer and get checked off as a pair.

DEN Students

DEN students will generally not be working in pairs since they are working remotely and away from other students in the class. DEN students will need to submit their labs in Vocareum, and a TA will grade them later. Because of this their answers to the lab questions will go in a `README` file, rather than on paper.

FYI, information about how and when DEN students must submit their labs is at the end of this document.

Getting Started

Step 1 Account activation.
If you are officially enrolled in this course, you should already have a Vocareum account.

If you did not previously have an account on Vocareum, you should have received (or will receive) an email invitation to join Vocareum. Simply click on the link in the email and it will take you to the login page. If you don't have the email, please go to the login page, enter your USC email address and choose "forgot password". If you still do not see an email, please check your spam folder. If the email is not there (e.g., because you are not yet officially enrolled in the course), please talk to your lab TA to get access to Vocareum. [From the Vocareum help pages.]

Step 2 Connecting to Vocareum
Once you log in, you can select "My Classes" and it will take you to a page with your class and assignments. When your teacher or TA publishes the course assignments, they become visible to you on the class page. [From the Vocareum help pages.]

Step 3 Getting to the lab assignment
You should see `Lab 1` in a list of assignments on the left side of the window. If you click it it takes you to a top level page for the lab. Click "My Work" (blue button) to get to the workspace for this lab. You will have a separate workspace for each lab or assignment that will start out populated with any starter files necessary for that lab. For some labs or assignments you will also create additional files as part of completing the assignment.

Exercise 1 (1 checkoff point)

If you followed the directions above you should be in your work area (aka, workspace) for this lab on Vocareum. The browser window will show a list of files on the left, an editor "window" at the top, and a terminal "window" at the bottom.

In this part of the lab you are going to play around a little bit with the terminal window to familiarize yourselves with some commands and the file system. This terminal window is running a Linux (Unix) shell. The Linux shell is a command-based interface to control the operating system of a computer; in this case it's a virtual machine based in the cloud.

Some of the things you'll do here using the command line you could also do using the GUI interface at the left of the screen. However, it's useful to understand some command-line basics. For example, you're going to be using the command line interface for compiling and running your programs (at least for now). In addition you'll be learning more about working with the shell as the semester progresses. Knowing Linux commands is also useful for more advanced shell programming, which you may need to do in this or other classes. Such programs, called shell scripts, are useful to automate routine chores on Linux. For example, when you submit your lab or we autograde your work, a shell script runs to compile your program and/or check whether its output is correct. There is also a shell script that does this in batch for all the programs submitted by students.

As you do the activities for this Exercise, we are going to ask some questions here, and you'll write down the answers on a piece of paper, in a lab notebook if you have one, or in a README file in your workspace (we provided you with a starter README file). Each lab partner should do the activities, but it's ok if just one person in the partnership is the "scribe" who writes down the answers to the questions for the pair. The questions to be answered by the scribe are in bold and labeled Question 1.1, 1.2, etc.

Remember, DEN students have to put their answers in the `README` file so they can submit it at the end.

When you got into your workspace you started out in your home directory (directory is the rough equivalent to a Windows or Mac folder). To see what directory you are in type the command

```
pwd
```

at the command line in bottom part of the Vocareum window. 'pwd' is short for Print Working Directory.

Question 1.1. What was the result of the command?

Note: if you are using a README file to record your answers, you can cut and paste the output from the shell window into the edit window.

Linux has a file system that's organized like a hierarchy or tree with one root. (This is similar to the folder structure on Windows.) The root of the tree is called "`/`". The results of pwd showed the absolute path from the root to your home directory.

To look at all the files in a directory you use the command `ls`. Try it now.

Question 1.2. What was the result of the command?

It will show the "starter" files we gave you for the lab. The home directory is also the one labeled "work" on the left side of the screen, and you can also see the files in that directory there.

Now you're going to make a new directory for our lab inside the current directory:

```
mkdir ex1
```

To change the current directory, you use the command `cd`. You don't have to use absolute pathnames when you are changing directories, but can just use a path relative to the current directory. Let's change our current directory to be the new directory we made:

```
cd ex1
```

Note that the dir also appeared on the left side of the Vocareum window, although it may not show up unless you collapse and expand the work folder. In Vocareum you can also create, rename, and delete files and folders from that interface.

Question 1.3. Predict what the results of the `pwd` command would be at this point. Verify your answer by trying it.

For any Linux command that takes a directory or filename as an argument we can identify that argument with a relative path instead of an absolute path. That is a path that's relative to your current directory. For example, if you're in your home directory, the following command will print out the contents of the "`ex1`" directory:

```
ls ex1
```

There are also shorthands for some particular paths (some relative). Here are a few of them:

.
: the current directory

..
: the directory one level up from the current directory

~
: your home directory

Question 1.4. Assuming you are still in the `ex1` directory, write down two different commands that will get you to your home directory from there (i.e., will make your home directory your current directory). You can test them out by going back to the `ex1` directory after trying each one.

These tricks are useful for navigating around in general, but there's actually another special shortcut for going to your home directory:

```
cd
```

(i.e., `cd` with no arguments).

Question 1.5. Make sure you are in your home directory. Write down the command to list what's in the directory one level up from your home directory. Try it.

To make a copy of a file, we use the command, `cp`, used as follows:

```
cp fromFile toFile
```

Similarly, `mv` (move) renames a file, and `rm` (only has one argument) removes a file.

Question 1.6. Assuming you don't know what directory you are starting from, write down a sequence of commands such that after they are done, the `ex1` directory in your workspace has a copy of the file `Hello.java` from your home directory. Try out your command sequence. Use `ls` to check if it worked.

To look at the contents of a text file we will often use a text editor. But if we want to look at a file quickly (especially if it's a small file), we can use the `cat` command (so called because we can also use it to concatenate files together). Or if we want a screenful at a time, we can use `more`. Try the following:

```
cat Hello.java
more Hello.java
```

Now, let's organize our work for this class a little better.

Question 1.7. Write down (and execute) the command(s) to make a directory in your home directory called `lab1`.

Hint: when you are done with this task, doing "ls" of the home directory should show:

```
ex1  Hello.java  lab1  README
```

Since ex1 is part of lab1, we might want it to be inside that directory. You can move your already-created `ex1` directory into this new directory using the `mv` command as follows:

```
cd                              make sure we are in home dir
mv ex1 lab1
```

Once you do this, you should see the lab file you copied earlier listed by doing:

```
ls lab1/ex1
```

Note: when you use `mv` or `cp` with an existing directory as the second argument, it moves or copies the first argument *into* the destination directory. (The first argument can't be a directory for this to work with `cp`.) Here are some other examples of this:

```
cd lab1                         starting lab1 as current dir...
cp ~/Hello.java ex1             does same thing we did in Qn 1.6
cp ex1/Hello.java .
```

Question 1.8. What does the *last* "cp" command above do?

Question 1.9. Assuming you are starting from your home directory write down and execute a single command that will get you into your `ex1` directory.

Once you are in the `ex1` directory, do the following command:

```
cd ../..
```

Question 1.10. What directory did you just change to?

Question 1.11. "cd" into the `ex1` directory. Using relative path names as seen in the previous examples, write down and execute a single command to make a directory called `foo` that's also inside the `lab1` directory (but that is not a subdirectory of `ex1`. By "a single command", we mean you will still be in the same directory (`ex1`) after making the new directory. (When you are done a listing of the `lab1` directory should show both `ex1` and `foo`.)

Question 1.12. Write down and execute a single command to get into the new "foo" directory.

Question 1.13. Write down and execute a single command to list the files in the "`ex1`" directory (i.e., without changing directories).

While "rm" removes a file, you need to use "rmdir" to remove a directory. Use this command to remove the "foo" directory.

Exercise 2 (1 checkoff point)

1. Make sure you are in the `ex1` directory you created in the previous exercise. Make sure the file `Hello.java` appears there.

   You should also be able to see Hello.java in the ex1 directory in the list of files/folders at the left of she screen. Click `Hello.java` to load it into the edit window. We're not going to change it for now, just view it. This is the traditional first program, that prints "Hello world!"

2. Compile the program from the command line using the command:

   `javac Hello.java`

3. The program should successfully compile. No output will go to the terminal window: it will just show a prompt for the next command. The compiled version of the program gets put in a file called `Hello.class`. You can see that it got created by doing the `ls` or looking at the panel that list your files at on the left.

   To run the program type:

   `java Hello`

   at the Linux command line.

Exercise 3 (1 checkoff point)

A common beginner mistake is to leave out a semi-colon. Edit out the semi-colon on the statement that prints out "Hello world!". Vocareum will automatically eventually autosave the file, but you can expedite this by clicking on the "Save pending" button (above the edit window).

You are going to compile again, but before you do so we'll show you a shell trick to avoid retyping commands in the shell: you can use the arrow keys to sequence between previous commands. Up arrow goes back in the sequence, and down arrow to goes forward again in the sequence.

Try using the up and down arrows to bring back different commands to the command line to see what happens. End up with the compile command you previously typed (Hint: it uses `javac`) and hit "return" (or "enter").

Note the error message produced (in the terminal window), and where it finds the error. When the compiler generates errors it does not produce a new `.class` file, so if you tried to run the program right now, it would work, because the old version of Hello.class has not been updated: it's the compiled version from the last time we successfully compiled.

Question 3.1. Copy and paste the text of the compiler error message into the `README` file.

`Hello.java` should still be in the edit window. Keep the missing semi-colon, and add another mistake: take out the second `"` (i.e., the double-quote after `Hello world!`). Compile again. Note the error messages given for this new mistake, but you don't have to put them in the README file.

There are three error messages even though you only had two mistakes in the program. This is because the compiler can't always guess correctly what you intended, and sometimes this causes cascading errors: errors that are indirectly caused by some real error, but that will go away once you fix your actual error. Because of this, sometimes it is best to just fix one error at at a time, recompile, and see if the other messages go away. Even more important, you should compile your code often, i.e., don't type large amounts of code before trying to compile the code, especially while you are still learning the Java language syntax.

Question 3.2. Note the line number of the third error. Where in the program does it think this error is?

Once you have done this, take out the errors in the program and compile it *but don't run the new version yet*, until we show you another feature of the command line:

You can edit an old or newly typed command on the command line, using the left and right arrow keys to navigate on the command line and typing or "delete" to change parts of what's there. Once you have the exact command you want you hit "return" (or "enter") to initiate the command.

Try this now by using the up-arrow key to bring up the command

```
javac Hello.java
```

and edit the command line so it says

```
java Hello
```

instead, and then hit "enter".

Exercise 4 (1 checkoff point)

In this exercise you are going to type in a program from scratch, although you can look at `Hello.java` as a pattern to follow. You could cut and paste that program as a starting point, but we recommend you type the whole thing instead to get used to the pieces that go in every Java program; so it soon becomes automatic for you, and you don't have to be looking at one to write one.

Create a file in your home directory called **`Name.java`**. The program you write must generate output similar to the following:

```
My name is Joe Blow.
My major is underwater basket-weaving.
I am a PhD student.
```

Your program output must be formatted as shown (i.e., 3 lines of output), but it will display *your own* name and information instead of that for Joe Blow.

Hint: The name of the class containing the `main` method must match the prefix of the filename containing that class; for example, `Hello.java` contains a class called `Hello`.

If you get compile errors along the way, write down a summary of what the error message said and what caused the error. You can put this in your lab notebook or add it to your README file as you did with the error from `Hello.java` (DEN students: put it in the README). Keeping a journal of error messages and their causes can be useful so that when you get the same error (possibly much) later, you don't have to figure out what caused it the second time around.

Submitting work on Vocareum

Only DEN students are required to submit labs, but for this lab only, we recommend on-campus students submit too, just to try it out (we'll ignore lab submissions for the on-campus students). However, *all* students will be using Vocareum to submit programming assignments (these are distinct from labs).

To submit your lab (or assignment) just click the blue "Submit" button near the top of the Vocareum window. This will make a copy of your whole workspace for this assignment (including any subdirectories) in a location where we can grade it. When you do "Submit", a shell script runs that performs some simple checks, such as whether all the required files are there, and whether the necessary code compiles. These checks are primarily to give you an idea of whether you submitted the correct version of the assignment, with the correct file names. If these checks fail the assignment still gets submitted, and it doesn't automatically mean you get a zero on the assignment (especially in the case of labs where different exercises are largely independent of each other: you can get credit for one exercise without having to have done all exercises).

Once you submit, it will show your most recent submit at the left side of the screen ("LatestSubmission"), in case you want to double check which version you submitted. You can keep working on your assignment after you submit an assignment, and submit another version later. In fact, you can submit as many times as you want before the deadline, but we will only grade the latest submission.

Summary for checkoff (on-campus students)

Show your lab t.a. your answers for exercise 1, a run of `Hello`, and your answers from exercise 3, your error messages and causes from part 3, and a working `Name.java` program.

Checkoff for DEN students only

Make sure you put your name, USC NetID, course, and assignment (e.g., lab1) in all the files to be submitted (`README` and `Name.java`). For Java source code files, such as `Name.java` you will need to put this information in one or more *comments* at the beginning of the file, e.g.,

```
// Name: Tommy Trojan
// NetID: ttrojan
// CS 455, lab1

public class . . .
```

When you are all done with the lab, you can submit it as described above.

DEN students must submit this (and all labs) by 11:59pm on Sunday Pacific Time for the Sunday that occurs just after the lab sessions for that week (e.g., if labs are on Thur, 1/10 and Fri, 1/11, your final deadline is Sun, 1/13). The deadline is delayed to give DEN student working full time more flexibility in when they do the lab. However, if possible we recommend you try to do it during the weekdays (well before the deadline) so you have ample time to get help from the course staff during the week, if necessary.
