Refresher workshop in programming for polytechnic graduates

General Java Program Compilation Guide

Overview

Welcome to this refresher workshop! This document will serve as a self-guided explanation to developing and executing Java program. It will guide you on the following topics:

- Simple unix commands
- Java program editing
- Java program compilation

Section 1: UNIX Crash Course

The following are a few commonly-used commands. This list is by no means exhaustive and you are urged to explore on your own. Note that UNIX commands are case-sensitive.

In the examples below, bold words are commands which you are expected to enter. As I am `dcstanst`, my default prompt may look like this (yours will possibly be a little different):

```
dcstanst@tanst-d960:~[36]$
```

`tanst-d960` is the internal name for my machine; ~ indicates that you are currently in your home directory.

a. Directory commands

pwd	to **P**rint current **W**orking **D**irectory to show you which directory you are currently in

```
dcstanst@tanst-d960:~[xxx]$ pwd
/home
```

ls	to **L**i**S**t files in your current directory

```
dcstanst@tanst-d960:~[xxx]$ ls
c      doc
```

You may also use "`ls -F`" for more information (`-F` is one of the many options/flags available for the `ls` command. To see a complete list of the options, refer to the man pages, ie. "`man ls`".)

```
dcstanst@tanst-d960:~[xxx]$ ls -F
c/        doc/
```

The slash (/) beside the filename tells you that the file is a directory (folder). A normal file does not have a slash (/) beside its name when "`ls -F`" is used.

Note that the directories c/ and doc/ are created by the setup program in section A.3. If you did not run the setup program, you will not see any file at all.

You may also use the "`ls -l`" command (dash L) to display almost all the file information, include the size of the file and the date of modification. Try it now!

cd	to **C**hange **D**irectory from current directory to another

```
dcstanst@tanst-d960:~[xxx]$ cd c
dcstanst@tanst-d960:~/c[xxx]$ ls -F
ch1_1.c   ch2_1.c   ch2_2.c   ch2_3.c   ch2_4.c
```

Note that the prompt changes to ~/c to indicate that you are now in the `c` directory below your HOME directory.
Entering "`cd`" alone brings you back to your HOME directory, ie. the directory in which you started with when you first logged into the system.

```
dcstanst@tanst-d960:~/c[xxx]$ cd
dcstanst@tanst-d960:~[xxx]$
```

mkdir	to **M**a**K**e a sub**DIR**ectory in current directory

```
dcstanst@tanst-d960:~[xxx]$ mkdir another
dcstanst@tanst-d960:~[xxx]$ ls -F
another/   c/     doc/
```

rmdir	to **R**e**M**ove a sub**DIR**ectory in current directory -- note that a directory must be empty before it can be removed.

```
dcstanst@tanst-d960:~[xxx]$ rmdir another
dcstanst@tanst-d960:~[xxx]$ ls -F
c/     doc/
```

b. File commands

cp	to **C**o**P**y files

```
dcstanst@tanst-d960:~[xxx]$ cd doc
dcstanst@tanst-d960:~/doc[xxx]$ cp abridged.txt anotherfile
dcstanst@tanst-d960:~/doc[xxx]$ ls
abridged.txt   anotherfile    faq.txt        tutor
```

mv	to **M**o**V**e files from one directory to another; can also be used to rename files.

```
dcstanst@tanst-d960:~/doc[xxx]$ mv anotherfile afile
```

```
dcstanst@tanst-d960:~/doc[xxx]$ ls
abridged.txt   afile          faq.txt        tutor
```

rm	to **R**e**M**ove files. Be **careful** with this command -- files deleted cannot be restored (unless they have been backed up during the normal backup cycle).

```
dcstanst@tanst-d960:~/doc[xxx]$ rm afile
rm: remove `afile'? y
dcstanst@tanst-d960:~/doc[xxx]$ ls
abridged.txt   faq.txt        tutor
```

c. Command to display text files

cat	to string together or display (**CAT**enate) the contents of files onto the screen

```
dcstanst@tanst-d960:~/doc[xxx]$ cat abridged.txt
```

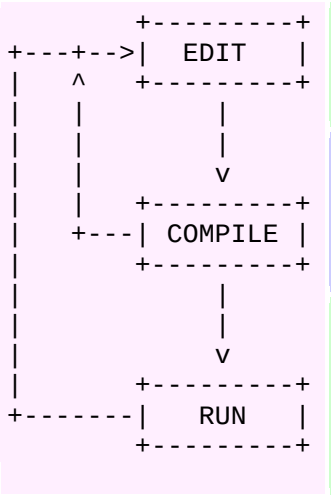
less	variant of "`cat`" (includes features to read each page leisurely)

```
dcstanst@tanst-d960:~/doc[xxx]$ less -e abridged.txt
```

	In "`less`", use <space> to move down one page, 'b' to move **B**ack up one page, and 'q' to **Q**uit from "`less`". You can also use the up/down arrow keys to move one line at a time.

Section 2: Running a Java Program

The process of creating a working Java program involves the following steps:



1. Use an editor of your choice, type in the source code (filename must have extension `.java`, eg: `Helloworld.java`)
2. Compile your program to obtain the executable file. If there are compilation errors, identify them and re-edit the source code before you proceed.
3. Run your executable file by typing the name of the executable file.
   If there are run-time errors, you must identify them and re-edit the source code, and compile again.

Section 3: Creating your own Java Programs

Create your first Java program "HelloWorld.java"

There are a number of editors available in sunfire: *vim, vi, emacs, joe, pico, nano* etc. The more powerful one requires a longer time to learn. For this lab, you will use `vim` -- a powerful editor with many commands, but even with the knowledge of a few simple commands it is quite easy to use and very powerful. It is YOUR responsibility to pick an editor and master it, and in future labs we will assume that you are familiar with your editor and its various functions.

It is recommended that you create a directory to store all your Java programs. Placing all programs under your home directory can get messy real quick. Follow the following steps to create a new directory:

1. Enter "**cd**" (this will get you back to the home directory)
2. Enter "**mkdir java**"(creates a new directory java/ under home directory)
3. Enter "**ls –F**"
   You should see the new directory **java/** along with other files and directories.

Go to the **java/** directory. Enter "`vim HelloWorld.java`", then press the "`i`" key. You will see that the words "`-- INSERT --`" appear on the bottom left corner of your screen. You are now in `INSERT` mode. While you are in `INSERT` mode, you may use the arrow keys (Up, Down, Left, Right) to move around your program, as well as the Backspace key and Delete key to delete text. This is contrary to what I mentioned during the briefing session. You can use the arrow keys in Insert mode after you set up your `.vimrc` profile. The PageUp and PageDown keys do not work, so do not use them. Do not use the scroll bar as it does not always work as expected.

Notice that line numbers (1, 2, 3, etc.) are displayed on the left. This helps you to easily identify a number by its line. Line numbers are NOT part of the actual Java code that you write, but are provided by *vim* to assist you in coding. This is extremely useful when identifying the location of compilation errors.

To copy text, you may use the mouse to highlight blocks of text, then right-click on the mouse and choose Copy. To paste your text, you must use the cursor keys to move the cursor to the desired position, then right-click on the mouse and choose Paste.

Type in the following program:

```
import java.io.*;

public class HelloWorld {

   public static void main(String [] args) {

       System.out.println("Hello World!");

   }
}
```

Note that for simplicity, the above program has no documentation. A good program should include documentation, at least the identity of the author, the purpose of the program and other relevant information. Keep this in mind when you write your own programs.

When you are done, press <ESC> then ZZ. You may also press "`<ESC>:x<ENTER>`" (more clearly seen as pressing the following four keys one after another: `<ESC> : x <ENTER>`) to save your program and exit from the *vim* editor. `<ESC>` means press the Escape key, then press the colon key (shift-;), then press x (the x is a lowercase x), and finally `<ENTER>` means press the Enter key.

If you want to save your file without exiting from the *vim* editor, press "`<ESC>:w<ENTER>`", then press the "`i`" key again to go back into `INSERT` mode. It is a good habit to save your file periodically so that if the network or the system goes down for any reason, you will not lose your hard work.

When you startup *vim*, it begins in `COMMAND` mode. One way to go into `INSERT` mode is to press the "`i`" key. While we are in `INSERT` mode, we can type in our Java code. To switch back to `COMMAND` mode, we press the "`<ESC>`" key.

The following shows you a list of useful commands in *vim*:

- `<ESC>:wq<ENTER>` : Saves your program and exits from *vim*.
- `<ESC>ZZ` : (Note that the Zs are uppercase) Saves your program and exits from *vim*.
- `<ESC>:q!<ENTER>` : Exits from *vim* without saving your program.
- `<ESC>ZQ` : Exits from *vim* without saving your program.

If you would like to learn more about *vim* commands, we encourage you to go to http://tnerual.eriogerg.free.fr/vim.html

Compiling and running your program

```
dcstanst@tanst-d960:~/c[xxx]$ javac HelloWorld.java
```

There will be error messages if your program has errors. Go to Step A.7.1 to make the necessary corrections and re-compile. If there are no compilation errors, a class file helloworld.class will be created. Proceed with program execution as follows:

```
dcstanst@tanst-d960:~/c[xxx]$ java HelloWorld
```

If you feel comfortable with the above steps, you can try out some bigger java programs from the textbook. Do not be discouraged by the compilation errors (you are bound to get a few ☺), it is more important to understand those errors quickly (what they mean and how to fix them).