

Linux, X, Emacs, and Java¹

1 Lions and Tigers and Bears – Oh My!

This semester, we will be using the CS department's Linux workstations for all programming in CS230. You are not expected to have any prior experience with Linux coming into this course. As part of your first labs and assignments, you will become familiar with Linux and related tools (e.g., shells, X Windows, Emacs, and the Sun Java system). This handout provides some information to help you get started, and pointers to places where you can find more information.

There are currently 21 public CS department Linux machines distributed in two areas: the Linux Lab (the open computer area outside of rooms E131, E133, and E135, also known as the **micro-focus**) and the Hardware Lab (E125). Most of the machines are named after lions, tigers, and bears:

1. **The micro-focus:** caribou, druantia, grizzly, ibex, kudu, leopard, lion, lynx, moose, ocelot, polar, rhodes, santorini, tiger
2. **E125** (Hardware lab): baboon, chimp, gibbon, gorilla, lemur, orangutan, tamarin

You may use any of these workstations for your CS230 work. Note that the machines in E125 are dual boot (can be booted into both Windows XP and Linux); you may need to reboot these into Linux in order to use them.

If you want to stay current on the status of the Linux workstations, you may want to follow the **Vibrant Linux** conference on FirstClass, inside the **Computer Science** conference folder.

2 Puma accounts

In order to use a Linux workstation, you must have an account on the CS department fileserver, `cs.wellesley.edu` (also known as **puma**). To obtain such an account, you will need to fill out the form available from:

<http://cs.wellesley.edu/user-info/>

Select the link labeled **appropriate forms** on this page, and then select the option *registering for some other course or other reason* in the list that appears.

- If you have taken CS111 here, you already have a puma account, but it needs to be upgraded. Select *I already have an account, just list me in the class*. You will still need to fill out the information in the rest of the form. This is a chance to reset your account password if you have forgotten it or want to change it.
- If you do not have a puma account, select *I don't have an account, please create one for me* and fill out the rest of the information.

¹This handout was written by Lyn Turbak and revised by Ellen Hildreth

You will also need to enter a “magic word” as part of the form – ask Lyn or Stella for this word.

If at some point you forget your password, go to the above web page, click on the **appropriate forms** link, and select option *You need your password reset*.

Regardless of which Linux workstation you use, all files in your account are physically stored on the puma fileserver and not on the Linux workstation itself. For example, if you are user `gdome`, and you create a file named `~gdome/test.txt` while working on the workstation named `polar`, this file is actually stored on `puma` and not on `teddy`. You can later view, edit, or delete this file from any other CS department Linux machine.

3 Logging In to a Linux Console

The easiest way to do your work in CS230 is to log in directly to one of the Linux consoles in the micro-focus or E125. A console that is not in use displays a Linux login screen that has a redhat icon in the upper left corner and a small window to enter your username. To log in, type your `puma` username (the same as your “short” FirstClass username) and press the ENTER key. A second window will appear where you should type your password, followed by ENTER. If the log in is successful, a new screen will appear with three icons in the upper left corner (one will be labeled with the name of your home directory), and several icons along a menu bar at the bottom of the screen. At the far left end of the menu bar is a red hat icon that opens a menu, similar to the Start button on Windows machines. In this document, this menu will be referred to as the “red hat menu”.

Now you should open a window with a Linux shell. To do this, select the **System Tools>Terminal** option from the red hat menu. A window will appear containing a shell prompt that looks something like

```
[username@hostname current-directory]
```

where *username* is your username and *hostname* is the name of the machine into which you logged in. The *current-directory* is the directory that you are currently connected to, and is initially set to your username when you first log in.

A few things can go wrong when you are logging in:

- If you misspelled or used the wrong username or password, you will be prompted for your username and password again. If you cannot log into the console of a Linux workstation after repeated attempts, send an email message to both Lyn and Stella.
- If the screen is blank, type any key and the login screen should appear.
- If the screen displays a screensaver of some form, it has been “locked” by another student (see notes on locking etiquette, below) and you cannot use it. Try to log in to another machine.
- If the screen display looks like a window manager, then another student is logged into the machine and may still be using it.
- If you are in E125 and the screen display looks like Windows XP, then the machine has been booted in the wrong mode and needs to be booted into Linux mode.

4 The Linux Shell

The prompt that you see after selecting **System Tools>Terminal** from the red hat menu is part of a command-line interface to a Linux system known as a **shell**. At the shell prompt, you type a

Linux command to execute, followed by the ENTER key. Linux then executes this command, and upon finishing the execution, presents you with another shell prompt. This mode of interaction may be unfamiliar if you have only had experience with a point-and-click, drag-and-drop window system. Although many tasks are not as convenient in a text-based shell as in a graphical interface, some tasks, such as finding all files that satisfy a non-trivial specification or automating a sequence of tasks, are actually more convenient in the text-based interface.

There are a plethora of shell commands for tasks such as navigating through and modifying the file system, searching for files that match certain criteria, finding documentation, and invoking programs like text editors and compilers. For a quick introduction to some very basic commands, see Section 4 (Shell Commands) of Scott Anderson's article *Introduction to Unix and the X Window System*, which can be found online at:

`http://cs.wellesley.edu/user-info/handouts/unix-intro.pdf`

and is linked from the CS230 home page. For a more detailed introduction, read Chapter 4 (The Unix Shell) of Larry Greenfield's *The LINUX Users' Guide*. There are a few red-bound copies of this guide next to the Linux workstations in the Linux Lab. The guide is also available online at the following URLs, which are linked from the CS230 home page:

HTML version: `http://espc22.murdoch.edu.au/~stewart/guide/guide.html`

PDF version: `http://cs.wellesley.edu/user-info/handouts/linuxUsersGuide.pdf`

Once you have mastered the simple shell commands and are comfortable with X Windows, Emacs, and Java, as described below, you are encouraged to learn more powerful shell commands. A good starting point is *The LINUX User's Guide*, particularly the following chapters: Chapter 6 (*Working with Unix*); Chapter 7 (*Powerful Little Programs*); Chapter 9 (*I Gotta be Me!*); and Chapter 11 (*Funny Commands*). Also, the Linux `man` command can be used to find detailed documentation on any command. For example, executing `man ls` gives documentation on the file-listing command `ls`.

5 X Windows

When you first log in to a Linux machine, the X windows system is launched automatically. If for some reason, you are not running X on your machine, you can launch X via the shell command `startx`. This will change the display from a strictly text-based interface to a graphical windows interface similar to that on Macs and Windows machines.

The particular window manager we are using this semester is called Gnome. Using Gnome is fairly intuitive. As described above, at the bottom left corner of the screen is a red hat icon that serves a purpose similar to the Start button in Windows. Click on this icon for a menu of options. Some particularly important options are `Programming>Emacs` (to launch Emacs) and `Internet>Web Browser` (to launch Mozilla, a web browser).

6 Emacs

Emacs is an extensible, customizable, self-documenting text editor created by Richard Stallman. Many consider it to be one of the greatest programs of all time. It is one of the flagship programs of Stallman's Free Software Foundation and GNU project.

You will be doing most of your work this semester – writing, executing, and debugging Java programs – using Emacs. In fact, it is possible to do all your work in the course entirely within

Emacs. It is very important to become a proficient Emacs user because this will save you a lot of time during the semester.

There are two standard ways to launch Emacs:

- Execute `emacs &` from within a shell. The `&` will create a separate Emacs window. If you are working remotely and do not wish a window to be created, instead execute `emacs -nw` (the `-nw` means “no window”).
- Select the `Programming:Emacs` option from the red hat menu.

All Emacs documentation, including a tutorial and reference information, is available online. If you are unfamiliar with Emacs (or have used it before but are rusty), you should take the online Emacs tutorial. You can do this by typing the `Control` and `h` keys at the same time, followed by the `t` key.² This will load an interactive tutorial, whose directions you should follow. When you complete the tutorial, you will know how to do basic editing in Emacs.

The tutorial teaches you keystroke commands for basic Emacs functionality. If you prefer, most of this functionality can instead be accessed by using a combination of the mouse, menu items, and arrow keys. However, we strongly recommend that you learn the keystroke commands, as they will save you lots of time and make it easier for you to work remotely from PCs or Macs (see Sec. 10).

In addition to taking the tutorial, you should read Scott Anderson’s article *Introduction to The Emacs Editor*, which can be found online at:

<http://cs.wellesley.edu/user-info/handouts/emacs-intro.pdf>

Another useful introduction to Emacs is Chapter 8 (*Editing Files with Emacs*) of Larry Greenfield’s *The LINUX User’s Guide*. You will find links to this and several sites containing more detailed Emacs documentation on the CS230 home page.

A particularly useful link is the Emacs reference card you can find at

<http://cs.wellesley.edu/~cs230/emacs-refcard-letter.pdf>

You may want to print out a copy of this card and carry it with you for handy reference.

It turns out that Emacs even has its own hypertext information system. This system contains detailed documentation on Emacs itself, and is worth exploring to find out more about Emacs. In order to access this information system, type the `ESCAPE` key, followed by the `x` key, followed by the character sequence `info`.³ This will load up an editor buffer that contains a top-level menu of the system documentation. You can browse this system via mouse clicks, much as you browse web pages in a web browser.

The Emacs command `M-x shell` creates a shell that runs inside an Emacs buffer. It is convenient to have a shell within Emacs, because then any shell command can be easily executed without leaving Emacs. This can be important if you are accessing a Linux machine remotely from PCs or Macs (see Sec. 10).

One minor drawback of running a shell under Emacs is that Emacs sometimes interprets or prints character sequences in a different way than a separate shell window would. For example, an Emacs shell will echo passwords that a normal shell would not. Also, the `ls` command in an Emacs shell may print a lot of annoying formatting characters; these can be removed by first executing `unalias ls` in the Emacs shell.

²In Emacs notation, this keystroke combination is usually written `C-h t` and pronounced “control-h t”.

³This keystroke combination, pronounced “meta-x info”, is usually notated as `M-x info`.

7 Java

We will use Java for all programming in this course. Our development environment is Sun's Java 2 Software Development Kit (SDK), Version 1.4.0. Using the Java SDK in Linux is different from using Java in CodeWarrior. The following subsections present the main things you need to know about using the Java SDK.

7.1 Editing Java Source Files

All editing of Java source files can be done with Emacs. As a simple example, you can use Emacs to create a file named `Test.java` with the following contents:

```
public class Test {
    public static void main (String [] args) {
        System.out.println("This is a test.");
    }
}
```

Because the filename ends with `.java`, Emacs “knows” that it is a Java source file and will enter a Java mode that facilitates the writing of Java code. For example, Java mode uses different colors to display different kinds of syntactic entities (e.g., keywords, types, method names, strings, etc.) and will match close braces with open braces. Java mode also “knows” the correct indentation for statements; typing the `TAB` key on any line will indent that line according to Java pretty-printing conventions.

7.2 Compiling Java Source Files

A Java compiler translates a Java source file (i.e., `.java` file) into one or more `.class` files that are suitable for execution on a Java Virtual Machine (JVM). In the Sun SDK, the `java` compiler is invoked via the `javac` command. For example, executing

```
javac Test.java
```

generates the class file `Test.class` for the sample program presented above. It can sometimes take `javac` a long time to compile a file. Unless there are errors, `javac` will not give any feedback about where it is in the compilation process. If you want more feedback, use the `-verbose` option. Here is an example of the information provide by this option:

```
[cs230@koala linux] javac -verbose Test.java
[parsing started Test.java]
[parsing completed 200ms]
[loading /usr/java/j2sdk1.4.0/jre/lib/rt.jar(java/lang/Object.class)]
[loading /usr/java/j2sdk1.4.0/jre/lib/rt.jar(java/lang/String.class)]
[checking Test]
[loading /usr/java/j2sdk1.4.0/jre/lib/rt.jar(java/lang/System.class)]
[loading /usr/java/j2sdk1.4.0/jre/lib/rt.jar(java/io/PrintStream.class)]
[loading /usr/java/j2sdk1.4.0/jre/lib/rt.jar(java/io/FilterOutputStream.class)]
[loading /usr/java/j2sdk1.4.0/jre/lib/rt.jar(java/io/OutputStream.class)]
[wrote Test.class]
[total 751ms]
```

In some Java development environments, such as CodeWarrior, a “project file” (`.mcp` file) is used to declare which source (`.java`) files are collected together to form a program. Unlike CodeWarrior,

the Java SDK has no notion of a project file. As a default, it is assumed that all the user source files for a program are stored in the same directory (it is possible to have source files spread out over several directories, but we will not consider that here) and that the user will compile the “main” source file with `javac`. The compiler determines all the files that the “main” source file ultimately depends on, and recursively compiles these as well.

For example, suppose that the contents of the three files `Pair.java`, `Swap.java`, and `SwapTest.java` are as shown in Fig. 1. Executing `javac SwapTest.java` will not only compile `SwapTest.java` (generating `SwapTest.class`), but will also compile `Swap.java` (generating `Swap.class`) and `Pair.java` (generating `Pair.class`), because the `SwapTest` class “depends on” both the `Swap` and `Pair` classes.

```
// The contents of Pair.java
public class Pair {

    public Object left, right;

    public Pair (Object left, Object right) {
        this.left = left;
        this.right = right;
    }

    public String toString () {
        return "<" + left.toString() + ", " + right.toString() + ">";
    }
}

// The contents of Swap.java
public class Swap {
    public static void swap (Pair p) {
        Object temp = p.left;
        p.left = p.right;
        p.right = temp;
    }
}

// The contents of SwapTest.java
public class SwapTest {
    public static void main (String [] args) {
        Pair p = new Pair ("foo","bar");
        System.out.println("Before swap: " + p.toString());
        Swap.swap(p);
        System.out.println("After swap: " + p.toString());
    }
}
```

Figure 1: The source code of three Java files.

7.3 Executing Java Applications

To execute a Java application, invoke the `java` command on the name of the class containing the desired `main()` method. For example, here is a transcript of invoking the two sample programs considered above:

```
[cs230@koala code] java Test
This is a test.
```

```
[cs230@koala code] java SwapTest
Before swap: <foo, bar>
After swap: <bar, foo>
```

Note that `java` is invoked on the name of the class and not on the name of the class file. For example, the invocation `java Test.class` yields an error:

```
java Test.class
Exception in thread "main" java.lang.NoClassDefFoundError: Test/class
```

7.4 Executing Java Applets

To execute a Java applet, invoke the `appletviewer` command on an `.html` file containing an applet tag referring to the class file for the desired applet. For example, suppose that `KnitWorld.class` is the class file for the CS111 applet that displays M.C. Escher's knitting patterns. Then the applet can be invoked via `appletviewer KnitWorld.html`, where `KnitWorld.html` is a file containing the following applet tag:

```
<applet code="KnitWorld.class" width=375 height=400>
</applet>
```

The above example assumes that `KnitWorld.html` and `KnitWorld.class` are in the same directory. If `KnitWorld.class` is in a different directory, the applet tag must contain an appropriate `codebase` attribute. For example, if `KnitWorld.class` is in a subdirectory named `Java Classes` (as is often the case in `CodeWarrior`), the appropriate applet tag in `KnitWorld.html` is as follows (the `%20` appearing in the `codebase` tag denotes a space character):

```
<applet code="KnitWorld.class" codebase="Java%20Classes" width=375 height=400>
</applet>
```

7.5 Debugging

Finding and fixing bugs (i.e., errors) in your programs is an essential part of the programming process, so it is important to hone your debugging skills in this course. Bugs can be classified into three categories:

1. **Compile-time Bugs:** These are errors in the syntax (i.e., form) of the program that are noticed and reported by the compiler. Common compile-time errors in Java include forgetting a semi-colon or brace, omitting the type of a parameter or the return type of a method, misspelling the name of a class or variable, and confusing an instance method or variable with a class method or variable. When `javac` encounters such an error, it prints an error message with the name of the file and a line number indicating the location of the error. Treat this line number as a hint rather than an absolute fact – often the real error is a few lines before or after the reported location. To locate the error, view the file in Emacs and use the `M-x goto-line` command to jump to the offending line. The compiler often reports several errors. Because a single mistake can often cascade into what the compiler thinks are many different errors, you should focus only on the first error it reports.
2. **Run-time Bugs:** These are errors that are not caught by the compiler but are found and reported by the Java Virtual Machine (JVM) when running the program. Common run-time

errors include attempting to invoke a method on the null pointer, accessing an array at an out-of-bounds index, or casting an object to an inappropriate type. Tracking down a run-time bug involves thinking carefully about the execution of the program – how does the program get into a state where the reported error occurs? To figure this out, it often helps to insert `System.out.println` statements at judiciously chosen spots in the program. Based on the information printed by these statements before the run-time error occurs, it is often possible to narrow down the source of the error. Graphical models like the Java Execution Model and Java Object Diagrams are also helpful aids in debugging. Expect to draw lots of diagrams on paper as part of the debugging process.

3. **Logical Bugs:** These are errors in the behavior of the program that are not reported by the Java compiler or the JVM but instead show up in testing the program. Common examples of logical bugs are a sorting method that does not correctly sort elements or an array-summing method that always returns 0. As with run-time bugs, good techniques for finding logical bugs are using `System.out.println` statements and drawing diagrams on paper. Another important technique is running the program on a variety of test cases to get a better sense of the kinds of conditions under which it misbehaves.

As a good overview to debugging, you should read the *Debugging* appendix from Allen Downey's *How to Think Like a Computer Scientist*, which can be found online at:

<http://cs.wellesley.edu/~cs230/debug.pdf>

This contains some good advice and concrete examples relevant to debugging.

7.6 Developing Programs

In CS111, most of your programming activity involved modifying existing programs. In contrast, in CS230 you will be writing many programs from scratch. This is a very different activity from modifying an existing program. In particular, just as essay writers experience the problem of the blank piece of paper, program writers experience the angst of the blank screen. Where do you start?

One important strategy in this regard is incremental program development. Start with a very simple program that does something related to the problem you are solving, and then modify it incrementally to solve more and more of the actual problem. The key is to have a working program at each intermediate step.

For example, if asked to write an array sorting program, you might first write a trivial sort method that leaves the array unchanged and embed it in a program that simply prints out the contents of some test arrays before and after calls to the sort method. Next you might modify sort to find the smallest element of the array and swap it with the first element. Because you have already wrapped the sort method in testing code, it is easy to test if this modification to sort works as expected. Finally, you can modify sort to embed the swap-the-minimum-element code in a loop so that the whole array becomes sorted.

For some good advice on how to develop programs from scratch, read the *Program Development Plan* appendix from Allen Downey's *How to Think Like a Computer Scientist*. This can be found online at:

<http://cs.wellesley.edu/~cs230/develop.pdf>

7.7 Java Documentation

You will spend much time in this course studying Application Programming Interfaces (APIs) for Java classes and interfaces – both those that are part of the official Java SDK, and those that are specific to CS230. You can browse all APIs for version 1.4 of the Java 2 SDK at the following URL, which is linked from the CS230 home page:

`http://java.sun.com/j2se/1.4/docs/api/`

8 Printing

There are three standard ways to print your files from a Linux cluster machine:

- Within Emacs, select either the `File:Print Buffer` or `File:Postscript Print Buffer` menu options. (The latter gives nicer looking output.)
- Within a shell, execute `lpr filename`.
- Within a shell, execute `a2ps -1 filename`.

All of these options will print your document on printer `psci11`, which is one of the printers near the mini-focus consultant's desk. If you use `lpr` or `a2ps`, you can print to a different printer using the `-P` option. For instance, you can print to `psci1r` using `lpr -Ppsci1r`. Note there is no space between the `-P` and the `psci1r`.

If you experience printing problems, please report them to our Linux system administrators by posting to the `CS-SysAdmin FirstClass` conference.

9 Saving Work

In addition to saving work in your puma home directory, you should make backup copies of your work on a Zip disk or on your own personal computer. For instructions on how to use a Zip disk with the Linux machines, read Section 8 (*Using Removable Disks*) of Scott Anderson's *Introduction to Unix and the X Window System*.

10 Using Linux Machines Remotely

You do not have to be physically seated in front of one of the Linux workstations in order to use it. You can access the department's Linux machines remotely from any PC or Mac on the Internet. From the PC's, which currently run Windows XP, you can connect to Linux machines using the Putty program, which can be found inside the `Program Files>PUTTY` folder on the local disk. After launching the program, enter the name of one of the Linux workstations (listed at the beginning of this handout) in the `Host Name` field. From Macs, which currently run OS X, first launch the `Terminal` program in the `Applications` folder. In the terminal window, you can then invoke the `ssh` program. To connect to the machine named `polar`, for example, enter the following command:

```
ssh username@polar
```

Note that you may receive a warning indicating that the authenticity of the host cannot be established, and a question about whether you want to continue connecting. If you respond with yes, the `ssh` program should then connect you to the Linux machine. You can also connect from one Linux machine to another via the above `ssh` command.

There are two key advantages of connecting to a Linux machine remotely. First, you can access the Linux machines from any other machine on the Internet – a fact which is important when you don't wish to go to the Science Center. Second, you can still use the machines even when all consoles are actively being used (several people can be logged into the same Linux machine at once). This is important to know when the lab areas are crowded with people near a problem set deadline.

A disadvantage to accessing the Linux machines remotely is that these clients provide only a text-based interface, so you will not be able to use the graphical user interface familiar from the console.⁴ For this reason, it helps to be familiar with Emacs control- and meta- key commands!

11 Logging Out of a Linux Machine

After you are done using a Linux workstation, you need to logout. From the Red Hat window manager, logging out is a two-step process:

1. Select the menu sequence **Red Hat:Logout**
2. Select **OK** in the resulting pop-up window.

You know that you have succeeded in logging out when you see the Linux login prompt appear.

If you are logged in remotely, you can log out by executing `logout` in the shell created by your remote access program.

It is important not to accidentally leave yourself logged in to a Linux machine when you are done. If you do so, someone may accidentally or purposely read, modify, or delete your files. Also, you will be tying up an important resource.

If you want to leave a Linux console for a short break, you can “lock” your console by selecting **Red Hat:Lock Screen**. This will lock the screen in such a way that your password is required to unlock it. You should only lock machines for *short* breaks (as a rule of thumb, no more than 15 minutes). Otherwise, you will be tying up an important resource that someone else may need to use.

⁴Note: there are programs (such as Hummingbird's Exceed software) for both Macs and PCs that can display X windows from a remote Linux machine, but these are not standard on the public cluster machines at Wellesley.