FIRAS ABUZAID
CS 145
October 1, 2013

Unix Help Session

Introduction

This overview is designed to give you an initial introduction to the Unix workbench and its various command-line tools, which will be very useful in CS145. In particular, you will learn how to:

1. Connect and log in to the remote cluster machines, such as `myth.stanford.edu` or `corn.stanford.edu`
2. Navigate and manipulate the Unix filesystem
3. Use various Unix tools and applications to complete your CS145 assignments and project

This is *not* a comprehensive overview of Unix — the Unix command-line has an enormous feature set, the vast majority of which isn't applicable to this class. For those interested, however, there are several excellent resources online that provide a much more thorough and detailed overview of the Unix workbench, which you can find at the end of this overview.

Connecting to a Remote Machine

Similar to the various computer clusters that Stanford provides around campus for academic use, there are also Unix workstations that you can log into remotely and use for academic purposes. Specifically, we will be using the `corn.stanford.edu` cluster machines for this class. Your projects will be graded on the `corn` cluster, so you need to make sure your code runs on `corn`.

Login Instructions for Mac OS X and Linux

First, open your Terminal application — for Mac OS X users, this should be located in your `Applications` folder under `Utilities`. For Linux users, this location will vary depending on which distribution of Linux you are using. Then, to open a remote connection, we'll use the Secure Shell (`ssh`) command. For instance, to connect to the `corn.stanford.edu` cluster, type the following into your Terminal:

```
ssh <your sunet-id>@corn.stanford.edu
```

This gives you a basic shell, but you will not be able to run any programs that require a graphical interface. If you want to use graphical interfaces and have X11, you can issue your `ssh` command as:

```
ssh -X <your sunet-id>@corn.stanford.edu
```

which will enable X-forwarding for your remote session. You will be prompted to input your password (the same password associated with Axess, Coursework, and your other online Stanford services). You may also be asked to add the remote machine to your list of known hosts — enter "yes" if you receive this prompt. Once you have successfully entered your password, you should receive a message in your Terminal welcoming you to the `corn` cluster.

Login Instructions for Windows

For Windows users, there is not native support for the Secure Shell, but there are many third-party `ssh` clients that provide the same functionality. Here's a list of some of the more popular clients:

1. MobaXterm
2. PuTTY
3. SecureCRT

I recommend you start first with MobaXterm; to set it up, follow the installation and configuration instructions found here.

Navigating and Manipulating the Unix Filesystem

Hopefully, you are now logged into a `corn` machine! On login, you will be placed in your Home directory on AFS, Stanford's filesystem. (For more information on AFS, see `https://itservices.stanford.edu/service/afs`.) I'll speak more about files and directories in a little bit. You are interfacing with the operating system through what's called a **shell**. The shell is a textual command-line interface that allows users to type in various commands to read, write, and execute files and programs.

In order to work through the demo, you will need to first issue the command:

```
cp -r /usr/class/cs145/unix-help-session/ .
```

into your shell. This will copy a demo directory into your AFS home, which contains a few sample files and directories to help you get acquainted with Unix.

Unix Directory Tree and the Working Directory

The Unix filesystem is organized hierarchically; there is a root directory, denoted by a forward slash (`/`), and all files on the system are children of this root. Children of the root may themselves have children, leading to a tree structure. Each element in the filesystem has an **absolute path** — a path relative to root. For example, a file called `bar.txt` stored in directory `foo/` which is a child of root

would have the absolute path `/foo/bar.txt`. Within your shell, you are always located in some directory within the filesystem, denoted as the **working directory**. If you're not sure what your current working directory is, issue the command `pwd` (which stands for "print working directory"), and your shell will output your current absolute path.

For example, upon login, your current working directory will be your Home directory on AFS, so typing `pwd` right after login will print the Home directory's absolute path. For me, `pwd` prints the following:

```
/afs/ir/users/f/a/fabuzaid
```

Your output should be similar, with your own SUNet ID at the end of the absolute path.

Navigating the Directory Tree

You can change your working directory via the `cd` ("change directory") command. Earlier, I gave you a command to copy a demo directory into your AFS home. That directory is located at `<your-home-directory>/unix-help-session`. Let's `cd` there — execute the following command:

```
cd unix-help-session
```

Here `unix-help-session` is an **argument** for the `cd` command, specifying the directory we wish to change to. Notice that we did *not* specify the absolute path of the directory; had we used the absolute path, our command would have looked something like this:

```
cd /afs/ir/users/f/a/fabuzaid/unix-help-session
```

Instead, we used what's called a **relative path**, a directory path that's relative to the current working directory. Using a relative path, our command is *much* easier to execute. If we run `pwd` now, it should verify for us that we have a new current working directory. For me, `pwd` now prints this:

```
/afs/ir/users/f/a/fabuzaid/unix-help-session
```

So now that we're in a different current working directory, how do we see its contents? How can we view what files and directories it contains? We do so by executing the `ls` command, which lists the contents of a given directory. In this case, when we execute `ls` inside the `unix-help-session` directory, we get the following output:

```
java python sample-dir sql
```

We can also pass the name of a directory as an argument to the `ls` command — what happens if you execute `ls sample-dir`? You should get this:

```
dir1 file1 large-file
```

Let's try one more thing — we can add a `-a` **flag** to the `ls` command, which will give us a somewhat different output. If we execute `ls -a`, the output now becomes:

```
.  ..  java python sample-dir sql
```

We get the same four directories as before, but we also see two additional directories, "." and "..". What exactly do "." and ".." mean?

Relative Paths and Shortcuts

As we've seen, there are two types of paths: absolute paths (relative to the root directory) and relative paths (relative to the working directory). There are a few shortcuts that make life simpler when dealing with relative directories: ".", "..", and "~". The single period represents the current directory; the double period represents the current directory's **parent directory**; and the tilde represents your Home directory. For example, suppose we execute `cd sample-dir` and enter the `sample-dir` directory. Now, let's execute `ls ..` — we should get the following:

```
java python sample-dir sql
```

which is the same same output we saw earlier, when we inspected the contents of the `unix-help-session` directory.

Here's another example: suppose we execute `cd ~`, which should take us to our Home directory. If we then execute `pwd`, we should see a very familiar result:

```
/afs/ir/users/f/a/fabuzaid
```

Yep, `pwd` returns to us the absolute path of our Home directory, as we expected.

Creating and Removing Files and Directories

To create a new directory, we use the `mkdir` command and pass as an argument the name of our new directory. For example, if we executed

```
mkdir new-dir
```

then this would create a directory called `new-dir` inside our current working directory.

We could then `cd` into `new-dir`; a simple `ls` would show us that the directory is empty, which shouldn't be a surprise. So let's create some new files, using the `touch` command. We'll execute

```
touch new-file1 new-file2
```

which will create two new empty files inside the `new-dir` directory. (Note that we passed multiple arguments to `touch` in this example, which is perfectly normal in Unix.)

To remove files and directories, we use the `rm` command. If we want to remove `new-file1`, we simply execute

```
rm new-file1
```

To remove directories however, you need to include the `-r` flag with the command, which stands for "recursive". If we wished to delete the `new-dir` directory for example, we'd execute

```
rm -r new-dir
```

which would remove the directory and all its contents from the filesystem.

Copying and Moving

Once files are created, you will likely want to copy and move them. Moving files is accomplished via the `mv` command. Suppose we want to move `file1` (located inside `sample-dir`) into `dir1`. In that case, we'd issue the following command:

```
mv file1 dir1
```

If we now check the contents of `dir1` (using `ls dir1`), we'll see that `file1` is now located there. To move it back to the current directory, we'd execute:

```
mv dir1/file1 .
```

Another common use for the `mv` command is to rename files. For example, we can rename `file1` to `file0` like so:

```
mv file1 file0
```

If we execute `ls` now, we'll see that only one file – `file0` – is present.

What if you want to *copy* a file or directory, instead of moving it? Copying files can be accomplished via the `cp` command. To copy `file0` into `dir1` (instead of moving it there), we'd simply execute:

```
cp file0 dir1
```

Now, let's take a look at the command I gave earlier:

```
cp -r /usr/class/cs145/unix-help-session/ .
```

It should be clear what's going on here — we're copying a directory, `unix-help-session`, into our current working directory. And since it's a directory, we use the `-r` flag, just like in the case of `rm`.

Viewing and Editing Files

So far, we've learned how to create, delete, move, and copy files. But what about editing files and viewing a file's contents? Thankfully, Unix provides a wealth of tools that let you view and manipulate files in various ways.

Viewing Files

There's a litany of ways to view the contents of a file in Unix, but there are two commands that are particularly useful: `cat` and `less`.

To use `cat` (short for "concatenate"), we simply invoke the command with the file(s) that we want to view as arguments. For example, if I want to view the contents of `file1` in `sample-dir`, I'd simply execute:

```
cat file1
```

And I'd see the following output in my console:

```
I'm file number one
```

Let's try it with a different file — what if we execute `cat large-file`? It works, but there's a small problem of sorts: there's not enough space in the console to display the entire contents of `large-file`.

So, instead of using `cat`, we can use another command, called `less`. Now, if we execute `less large-file`, we see that, although there is still not enough space in the console, we can easily scroll up and down using the `j` and `k` keys on our keyboard. `less` provides a much richer set of functionalities overall, and is much more useful for viewing larger files.

Editing Files

Unix also provides a whole host of applications for editing files as well. There are three in particular that I recommend using for this class: `vim`, `emacs`, and `gedit`. Of the three, `gedit` is the easiest to use, and, in terms of simplicity, it very much resembles the text-editing features found in Eclipse, XCode, and Visual Studio. In contrast, `emacs` and `vim` have a much steeper learning curve, and it definitely takes time to grow accustomed to using either of them. However, there are lots of rich resources available online for learning to use either `vim` or `emacs`, and both offer lots of unique and advanced features. For more information, check out the CS107 guide to `emacs` here, and check out the CS107 guide to `vim` here.

Running Programs

In CS145, we'll be working with Java, Python, and SQL — luckily, Unix also provides extensive support for working in all three environments.

Compiling and Running Java Code

In Unix, you can compile Java code by using the `javac` command. For example, in the `java` directory inside `unix-help-session`, we have a file called `hello.java`. If we execute `javac hello.java`, this will generate a new class file, `hello.class`. If we then use the `java` command (e.g. `java hello`) we can then execute our Java program, which should give us the following output:

```
Hello, world!
```

If you prefer working in an IDE when developing in Java, the `corn` and `myth` machines also offer the Eclipse IDE. Simply type `eclipse` into the command-line, and Eclipse will open as a result. (Note: Be sure to enable X-forwarding! Otherwise this will not work!)

Running Python Code

For Python, we don't need to worry about a compilation step — we can simply invoke the `python` command in the command-line and pass the filename of the Python script we wish to run. In the `python` folder, there's a sample script already present, called `hello_world.py`. Executing `python hello_world.py` should give you the same output as before — `Hello, World!`

If you invoke the `python` command without any arguments, then you'll instead open the Python interpreter, which will let you write and execute Python code in a Read-Eval-Print loop.

SQLite

For database work in this class, we'll be using SQL, which actually comes in several different flavors — MySQL, PostrgeSQL, and SQLite. On the `corn` machines, I recommend using SQLite, as it's the easiest of three to use right away.

To start a SQLite database, simply invoke the command `sqlite3` in the command-line. This will open up a SQLite console, much like the Read-Eval-Print loop we saw in the Python interpreter. You can also initialize the SQLite console with a database of your own, by passing the filename of the database as an argument. In the `sql` folder, we've given you a sample database called `sample.db` that you can use for experimentation as you get accustomed to SQL and SQLite.

Links

Finally, as part of your course project in CS145, you will be asked to parse a large dataset, consisting of many files. You won't be allowed to modify the dataset, however; you will effectively have read-only access to the data. As such, if you want to be able to quickly access the dataset, copying it in its entirety into your directory is not a very wise choice — your AFS space is limited, after all!

Fortunately, we can use **symbolic links** to give quick and easy access to our dataset. By creating a symbolic link, the data won't be physically located in your directory, but you will be able to access the data as if it was. For example, we've created a sample dataset of 100 files at `/usr/class/cs145/sample/data`. You can link to this dataset using the `ln` command, like so:

```
ln -s <source path> <link path>
```

(Note: Don't forget the `-s` flag, which stands for "symbolic".) Notice now that all of the data is easily accessible to you, and you can treat the dataset as if it's setting inside your directory. And all of this is achieved without having to allocate more space on the filesystem.

Conclusion

Hopefully, you now have a much deeper familiarity with the Unix workbench — you have all the tools and knowledge you need to complete the assignments and programming project in CS145. If you still feel uncomfortable working with Unix, or if you have any generic Unix-related questions, please don't hesitate to ask for help during office hours or on Piazza.

Resources

If you'd like to learn even more about the Unix environment, check out these handy resources:

1. CS107 Basic Unix Tutorial

   <https://courseware.stanford.edu/pg/pages/view/365347/cs107-guide-on-using-unix>

2. CS107 Guide to Unix Tools and Pipelines

   <https://courseware.stanford.edu/pg/pages/view/365336/cs107-guide-to-unix-dev-tipstricks>

3. Unix Programming Tools

   <http://cslibrary.stanford.edu/107/UnixProgrammingTools.pdf>

4. Unix Command Summary

   <https://www.stanford.edu/group/farmshare/cgi-bin/wiki/index.php/UNIX_Command_Summary>