Worlds of Linux and Provide

COMP 50

25 November 2013

This is the outline for a lab introducing Comp50 students to the Linux machines in Halligan, including such skills as remote connecting, commands for moving around in directories, secure file copying, and using the `provide` command.

In the beginning was the command line

Most students use computers that run Mac OSX or Windows operating systems. These computers have nice, graphical interfaces: they are controlled by moving the mouse and clicking on things. Unfortunately, you have to control them by moving the mouse and clicking on things. As you may have learned already, you can say more, quicker, by typing on the keyboard—as in DrRacket's *Interactions* window.

The Unix equivalent of the *Interactions* window is a *terminal*. To get one on the Halligan machines, chase menus along `Apps :: System Tools :: Terminal`. Or try right-clicking on the desktop, then click Open Terminal.

Here are some basic commands:

- `who` Who's logged in to this computer?
- `date` Self-explanatory
- `uptime` How long the computer has been on.
- `cal` The default behavior will give a calendar of the current month. If you want specific calendar information, you'll need to give the command more information, using *arguments*.

Files and Directories

Just as DrRacket has access to all your definitions and data, the Unix terminal has access to all the Unix definitions and data. This are stored in the *filesystem*. Each individual document or program is stored in a *file*. Files are organized into *directories*, which are sometimes called folders.

Just as DrRacket sees only one program at a time, your Unix terminal sees only one directory at a time. But the terminal includes a navigation system, similar to the Mac's Finder, to Windows Explorer, or Linux's file manager (Nautilus, Thunar, or something similar). The terminal is always "looking" at one directory, which is called the *current working directory*. To look at and change the current working directory, you must know a few survival skills:

- `cd` The change-directory command moves between folders. It can take an argument, which is the pathname of a directory. The default behavior changes into the home directory ().
- `ls` Lists out all of the files and directories contained inside of the current directory. It's equivalent to the teachpack function `directory-names`.

- `pwd` Print working directory; that is, it prints out the pathname of the directory you're in. The `setup` command from earlier made the terminal prompt show the working directory.
- `mkdir` Makes a new directory inside the current one. Takes one argument, which is the name of the new directory.
- `mv oldname newname` Stands for move: it can be used to rename files or to move them to dfferent directories.

There is one other super helpful command: `man`. This produces a manual page for any command, explaining its purpose and arguments. If you want to learn more about the `cal` command from earlier, try

```
man cal
```

If you don't know the name of the command, use the `-k` argument to specify a keyword, as in `man -k calendar`.[1]

A World of Computers

Future courses run exclusively on the Halligan servers. But to spend all one's time in Halligan is uncivilized. This section gives several problems which, when completed, will enable you to get remote access quickly and easily from your own computer. **If you do not have a Linux or Mac computer with you, skip to the next section**. You can come back to this later.

SSH

If you have a terminal somewhere, you can get a terminal anywhere (unless it is hidden behind a firewall). *From your own computer*, try the following problems

1. Run

   ```
   ssh-add -l
   ```

   (that's a lower-case ell, not a numeral one):

   a) If you get an error message saying "can't connect to an agent", give up.
   b) If you get a message saying "the agent has no identities", perfect. You're ready to move on.
   c) If you get a message like

   ```
   2048 73:21:d6:e3:b8:56:39:04:b3:c9:29:3d:f9:14:99:83 /home/nr/.ssh/id_rsa (RSA)
   ```

   then somebody has already configured SSH for you, and you're set up. Move way on.

2. Create a "key pair" by running `ssh-keygen`. It may work with no options, or you may want to consult http://en.wikipedia.org/wiki/Ssh-keygen.

   If all goes well, you'll be prompted for a "passphrase". **Use a very long, memorable passphrase.** Special characters and strange spellings are not necessary—what protects you is *length* and the ability to *remember* (not write down) your passphrase.

   Here's an example of a good passphrase:

   ```
   Tufts University is a place where unicorns eat the president's
   flowers
   ```

   Pick a different passphrase. It's yours—guard it as you would guard your password.

[1] Man pages are less useful than they once were. Ask Norman for his rant on this. Or just ask him to tell you to get off his lawn…

3. Now try

   ```
   ssh-add
   ```

   With luck the defaults will work, and you'll be asked for your passphrase. Once you've typed it in, the machine knows your identity, and you can get remote access.

4. Confirm your authentication by `ssh-add -l`. This should list a strange message like the one shown above.

5. If your UTLN is `nramse01`, copy your new SSH keys to the Halligan server:

   ```
   ssh-copy-id nramse01@linux.cs.tufts.edu
   ```

   Instead of `nramse01`, use your own CS login. You will need your CS password. But you will never need to copy your ID again.

6. Test the whole shebang by trying to connect remotely to the server:

   ```
   ssh nramse01@linux.cs.tufts.edu
   ```

Using SCP to copy files

SSH (Secure Shell) uses the same credentials as SCP (Secure Copy). You will use `scp` to copy file from your own computer to a Halligan computer, so that you can submit them using `provide`.

7. To copy files from your own computer to a Halligan server, use `scp`. To learn far more than you ever wanted to know, try `man scp`.

   When you're copying between computers, you need to specify the account and computer that you're copying to, followed by a colon, followed by the path to the new location. For instance, if you were copying a PDF file to the Halligan servers from your home computer, the command might look like this:

   ```
   scp my.pdf utln@homework.cs.tufts.edu:./Desktop
   ```

   Try this now. If you don't have a PDF file handy, create one using a word processor.

   If your file name has spaces, you will need to write it in double quotes, like a string in DrRacket:

   ```
   scp "Learning Portfolio.pdf" utln@homework.cs.tufts.edu:./Desktop
   ```

Turning in work with `provide`

This problem explains what you have to do to submit your learning portfolio, as well as work in other CS courses.

8. The Handin server is good only for DrRacket. To turn in your learning portfolio, and in all other department courses, you will use the terminal with a command called `provide`.

   Here is an example, **which you must run from a lab machine or a server**:

   ```
   provide comp50 lab-provide my.pdf
   ```

   Try this now.

   `provide` copies all the files and salts them away in an undisclosed location. The word `comp50` identifies the course, and `lab-provide` identifies the assignment. You provide as many files as you like.

   If you make a mistake or need to change a file, just `provide` again. `provide` remembers your old submissions.

Here are the steps to submit this lab and your portfolio:

- Get your PDF to the `linux.cs.tufts.edu` server using `scp`. (Or walk to Halligan and do it on a lab machine.)
- Use `provide` to submit your work.

`provide` is available *only* on Halligan machines.

Unix is just like DrRacket, only more complicated

The rest of the lab introduces you to Unix and the command line. By the end of the lab, you should have some idea why it might be useful to know more than just where to click the mouse.

Most files on Unix are "text," and they obey these underlying principles:

- A text file is a list of lines
- A line is a list of fields
- A "program" is a like a function; its input is "standard input" and its output is "standard output"
- In BSL and ISL, one function can be applied to the results of another function. Unix works similarly, except two commands are separated by a "pipe" character (vertical bar or broken vertical bar).

Let's try it out:

9. Some data is so big that it is stored in compressed format. The USGS point-of-interest database is like that. Such files can be uncompressed with `zcat` or `gzcat`.

   Often you want to look only at the first few lines of a file—especially if the file contains a million lines.

   Try this with the USGS database

   ```
   zcat /comp/50/usgs.txt.gz | head
   ```

   ```
   zcat /comp/50/usgs.txt.gz | head -20
   ```

10. If you want to see more commands, use `less`. Inside `less` you can type the space bar or the letter `q`:

    ```
    zcat /comp/50/usgs.txt.gz | less
    ```

11. Unix has a variety of commands that act like `filter`. The most common is called `grep`. Find the first 15 points in Massachusetts:

    ```
    zcat /comp/50/usgs.txt.gz | grep -F '|MA|' | head -15
    ```

12. There is a super-duper command that can act like `filter`; it's called `awk`. Awk divides each line into "fields"; this command picks points out of a bounding box by looking at fields `$10` and `$11`:

    ```
    zcat /comp/50/usgs.txt.gz | awk -F"|" '$10 > 42.0 && $10 < 42.1 && $11 <
    -71.1 && $11 > -71.2' | head
    ```

13. How many points are in Massachusetts? We can count with `wc -l`, which behaves like the Racket function `length` (again, that's "dash-ell", not "dash-one"):

    ```
    zcat /comp/50/usgs.txt.gz | awk -F"|" '$4 == "MA"' | wc -l
    ```

    You should get 31,986. More than you want to see.

    The `-l` in `wc -l` tells it to count *lines*. `wc` also counts *words* and *characters*.

Exploring your own code

For the learning portfolio, we emailed you a `.zip` file containing all your work. That file can be "unzipped", which creates a bunch of new files in the current directory. Let's look at them:

14. Create a new directory and change to it:

    ```
    mkdir lab-provide
    cd lab-provide
    unzip ~/Downloads/mystuff.zip  # use the right pathname for your .zip file
    ```

15. Choose a .txt file that includes your work. How many lines are over 80 columns?

    ```
    awk 'length > 80' my.txt | wc -l
    ```

16. How many lines are over 90 columns?

    ```
    awk 'length > 90' my.txt | wc -l
    ```

17. What do those long lines look like?

    ```
    awk 'length > 90' my.txt
    ```

18. What do the long lines look like in *all* your files?

    ```
    awk 'length > 90' *.txt | less
    ```

    The star in `*.txt` makes it work on all files with that name.

19. Count the number of function definitions by

    ```
    grep -F '(define (' my.txt | wc -l
    ```

20. Count the number of tests by

    ```
    grep -E 'check-expect|check-error' my.txt | wc -l
    ```

21. What is the ratio of tests to definitions? The Unix command line is not very good at arithmetic, but you can ask Lua to do the arithmetic for you:

    ```
    lua -e "print($(grep -E 'check-expect|check-error' my.txt | wc -l) / $(grep -F '(defin
    ```

    A command is wrapped in `$(...)` in order to use its standard output as the result to another command.

22. For which assignment did you submit the most code?

    ```
    wc -l *.txt | sort -n
    ```

To submit the lab

Write up your usual findings—what you did and what you learned—but put them into a PDF file (text file if you're desperate) and submit them using

```
provide comp50 lab-provide lab-results.pdf
```