

A very short intro to Linux

Laura Alisic

October 8, 2008

This document is a very short summary of some basic commands, enough to get you started for the problem sets in Ge161. For a nice and more thorough overview, have a look at the text Eun-Seo Choi wrote. It is located in:

```
/home/datalib/Resource_Class/Unix_2007/linux_intro_all.pdf
```

The best way to learn working with Linux (or any programming language) is doing it!

Introduction Linux is an operating system, and is a flavor of UNIX. Almost everyone in the Seismolab works on a Linux machine; some people work on MacOS which is also based on UNIX. The reason to use UNIX-like systems is the ease with which one can write software and run it. Programming in Windows will mostly give you nightmares! A UNIX/Linux/MacOS system is very easily maintained and operated using the terminal, which is a window that has a command line in it. If you enter commands in the terminal, the system will interpret them and execute them. The general syntax is as follows:

```
command -option1 -option2 ... <filename>
```

Below, you can find some of the most common commands and their usage, the ones that you will use very often when working in Linux. If you master these, you will be able to move around in your file system, manipulate files and directories, perform simple file editing, make an executable of a script and execute it from anywhere you want.

Navigating between files and directories

- `pwd`: current directory
- `ls`: list files in current directory
- `ls *.sh`: list all files in current directory that end with `.sh`, the `*` is a wildcard for any number of characters
- `ls output?.dat`: list files in current directory that have the form `output.dat`, the `?` is a wildcard for one character (i.e. `output1.dat`, `outputx.dat`, etc.).
- `cd`: move to other directory
- `cd ..`: move one directory up

Manipulating files and directories

- cp: copy files
- cp -r: copy directory with files in it (recursively)
- mv: rename and move files (technically equivalent)
- rm: remove files (BE CAREFUL!! NO UNDO)
- rm -r: remove directory recursively (BE CAREFUL!! NO UNDO)
- mkdir: make directory

Example: type on the command line after opening a terminal: (% indicates a new terminal line)

```
% pwd
% ls
% mkdir tutorial
% cd tutorial
% mv /home/datalib/Resource_Class/Unix_2007/linux_intro_all.pdf .
```

You have just determined where you are in the file system, what files and directories are present in your current directory. You created a directory `tutorial` in your home directory, and copied a linux manual into it. Remember the directory `/home/datalib`, it contains many datasets that are very useful for this class (or anything in geophysics really).

Some special characters in Linux

- `.` : current directory
- `..` : directory up one level
- `~` : equivalent to your home directory
- `;` : ending command line, separates commands
- `&` : after a command causes it to run in the background. This is equivalent to pressing Ctrl+z when a process is running which suspends it, and then entering bg which keeps it running in the background. To bring something back to the foreground, enter fg.

Editing files Editing files in Linux can very easily be done in the terminal. It takes a little practice, but you will see it works much faster than any editor with a graphical user interface (GUI), such as Word. Another reason to use a command line-based editor is that Word and similar programs put in formatting characters which are unreadable by the terminal and hence not usable to write scripts with. There are various programs available for command line editing, of which the best known are Vi and Emacs. Here we will use Vi.

```
% vi helloworld.sh
```

This opens a new file in your current directory, named `helloworld.sh`. If you already had a file with that name there, it would open that one for editing. Now you will see an empty screen. Vi works with two modes, 'move-around-mode' and 'insert-mode'. When you open a file, it opens in the 'move-around-mode' and you can't type any new text. Here is a list of common commands in Vi:

- `:i` : switches from 'move-around-mode' to 'insert-mode', now you can edit your file
- `esc` button: switches back to 'move-around-mode'
- `:w` : saves your file
- `:q` : quits your file and Vi, will complain if you haven't saved it first
- `:q!` : quits your file and Vi without saving

Now try to put this on the first line of your `helloworld.sh` file, save it and quit Vi:

```
echo 'hello world'
```

Creating executables Now you have a file with a command in it. You can try the command just on the command line to see what it does. You can also run the file as an executable, so that the computer reads the file and executes it for you. For a one-line command this seems like a hassle, but you will soon find out that many tasks in Linux take multiple lines and are far easier done using scripts like this. In order to run the script, you need to tell the computer that it is an executable. This is done by changing the file permissions:

```
% chmod +x helloworld.sh
```

Now if you type:

```
% ./helloworld.sh
```

the script executes the command it contains. Its output is directed to the standard output, being the terminal command line here. For more complicated programs, ones that e.g. produce lists with tables, you may want to redirect the output to a file:

```
% ./helloworld.sh > output.dat
```

If you open `output.dat` with e.g. Vi, you can see that it contains the output you first saw on the screen. A good thing to know is that you can interrupt any running script using `Ctrl+c` in that terminal.

Environment variables Another feature that can be very useful for scripts is being able to run them from any directory, not just the one that contains the actual script. You can achieve this by setting environment variables, in this case putting the directory that contains the script in your `PATH`. First find out if your terminal is of the 'bash' or 'csh' flavor:

```
% cd ~
```

```
% ls -a
```

This command lists all files in your home directory, also the 'hidden' ones that start with a dot. If you have files starting with `.bash` in that list, you have the 'bash' flavor. Similarly with `.csh`, you have the 'csh' variety. To set your environment variables in bash, add a line in `.bashrc`, e.g. at the bottom of the file:

```
% vi .bashrc
:i
export PATH=~/tutorial:$PATH
:wq
% source ~/.bashrc
```

To set your environment variables in csh, add a line in `.cshrc`:

```
% vi .cshrc
:i
setenv PATH ~/tutorial:${PATH}
:wq
% source ~/.cshrc
```

Now you can run any executable scripts you put in the directory `tutorial` from anywhere! A little more explanation on what you have just done: `.bashrc` or `.cshrc` is read in by the system every time you open a terminal. It can contain environment variables just like above, or it can contain 'aliases' which are command substitutions that you provide yourself (e.g. if you want to use `ls -a` every time you enter `ls`, you can specify that). This file is a very important one, so be careful when you edit it! Backing it up when you make many changes can't hurt.

Every time you make a change to this file, you will need to 'source' it, meaning that your current terminal has to re-read the file or it will not recognize that you made changes to it. A terminal window that you open after making changes to an rc file will have read in the modified file and should be fine.

Some other handy things

- To open a ps or pdf file in linux, use `gv` with file name.
- To find in which directory an executable script is located, use `which` with the script name.
- To print a file, use e.g.:

```
% lp -d 366sm_hp_2side file
```

- To find more information about a command, use `man` command. Also typing the command followed with `--help` or `-h` quite often works.

These days, Linux systems contain many GUI-based programs which are equivalent to what you are familiar with on Windows system. There is Open Office, which is similar to Windows Office. Also image editing software such as Gimp and Inkscape are extremely useful. Just go explore!