Chapter 1

A newcomers guide to Linux

This chapter was written by **Thomas Breslin**, **Michael Green**, **Björn Canbäck** and **Lokeshwaran Manoharan**.

1.1 Basics

In this section we will go through some of the most basic stuff one needs to know to be able to work on a Linux system. The hash sign, #, is a comment. If copying and pasting the commands it doesn't matter if you include the comment.

1.1.1 The file tree and how to navigate it

A typical hard disk contains thousands and thousands of files and if we were to put them all in the same place this would result in an enormous list. So how do we deal with this situation? Well we deal with it the same way we do in an office. We get a lot of folders and put files which belong together in the same folder. Then, since there are so many files we get a big rack and put the folders which belong together on the same same shelf in the rack. Then, since we have just about as many files as the national tax authorities we clear out few rooms and put the racks belonging together in the same room. Well, by now you get my point. On a hard disk files are stored in directories and in each directory there are sub directories and in those sub sub directories and so on. The top-level directory is called the root and is denoted /.

Navigation skills

Assuming we have logged in, the first thing we do is to check out our position in the file tree. We do this by saying `pwd` *print working directory*.

```
pwd # /home/student1
```

When I say `pwd` to this machine it answers you are in the directory `student1` which is a subdirectory of `home` which is a subdirectory of / (root). Check it out on your machines! Now let's look at which files are in this directory. We do this by saying `ls` *list directory contents*.

```
ls
# Output: bin  public_html
```

Some of this stuff may be files and other stuff are subdirectories. On some terminals, this is color encoded but if it isn't we have to tell `ls` to be more specific. We do that by saying `ls -l`.

```
ls -l
# Output:
drwxr-xr-x 2 student1 root 4096 Jan 26  2013 bin
drwxr-xr-x 2 student1 root 4096 Mar  6  2013 public_html
```

Puh, thats a lot of information. All we have to concentrate for the moment is the first column, if it begins with a `d` then it is a directory. Otherwise it is a file (to be correct, directories are also files). To make a new directory (which will be a subdirectory of the one you are currently working in) we say `mkdir DIRECTORY` where `DIRECTORY` is the name we want the new directory to have. The keyword `mkdir` is an abbreviation of *make directory*.

```
mkdir test
```

Lets do that and then check what happened by typing `ls -l`

```
ls -l
# Output
drwxr-xr-x 2 student1 root    4096 Jan 26  2013 bin
drwxr-xr-x 2 student1 root    4096 Mar  6  2013 public_html
drwxr-xr-x 2 student1 student 4096 Nov 14 16:27 test
```

Voila, there it is. To enter our newly created directory we say

```
cd test
```

The keyword `cd` obviously means *change directory*. Typing `ls` reveals that there are no files or subdirectories which of course is what we expect. Now check out what `pwd` says. Does it make sense? To change to the parent directory of the current one we type

```
cd ..
```

The two dots mean above directory. Now we are armed for the first exercise.

Exercise

Using `cd, ls, ls -l` and `pwd` play around in the file tree. But first do `pwd` from your home directory and write down the path. After swinging around the beaches for a while, type `cd ..` enough times to move to the root. try to find your away back home from there. If you could not find your way back to home directory, this little trick comes well in hand. Type `cd ~` (tilde) and you're home. The `~` sign is an alias for your home directory.

Summary

ls *list directory content*

ls -l *list directory content with more detail*

cd DIRNAME *change the directory to* `DIRNAME`

mkdir DIRNAME *make subdirectory* `DIRNAME`

1.1.2 Files

Now when we know where files are and how to organize them, we will look at what files are in more detail. A file is really just a piece of information stored somewhere on a computer's hard disk. In its rawest format, a file is a sequence of ones and zeros which is what computer uses to represent everything. Some files are programs which are loaded on to the computer's memory and run on its processor, making the computer to do something that you want it to do. Other files contain ones and zeros that represent text which we can read. To create a file that contains text, we need to take the input from the keyboard and translate it to binary code. We also need to tell the computer where to store this binary code.

Creating, moving and copying files

The program we will use is called `cat`, an abbreviation for *concatenate* which means merge or put together. When we type `cat` in the terminal, it starts executing. It waits for us to type something on the keyboard and press return. When we do the `cat`, it reads what we typed in and spits it back on to the terminal. To tell `cat` to stop we hold on to the `control` key (Ctrl) and press `d`. Try it!

```
cat
Hi. I'm Batman!
# Output: Hi. I'm Batman!
I will press control-d to stop this.
# Output: I will press Ctrl-d to stop this.
```

So whatever we give as input to `cat` it gives it back as output. There is a standard way of telling the computer to put the output of a program into a file. One just types the program name, followed by `> FILENAME` where `FILENAME` is the name of the file one wants the output to end up in, and `>` is the output director. So to create a file we do like this:

```
cat > somewords
I wear a mask. It's not to hide who I am, but to create what I am!
```

End with `return` and `Ctrl-d`. Now that we know how to redirect the output from a program to a file, let's try the opposite; redirecting input. Instead of letting `cat` take the input from the terminal we'll make it take the input from the file we just created.

```
cat somewords
# Output: I wear a mask. It's not to hide who I am, but to create what I am!
```

If one wants to *copy* a file we use the copy command: `cp`:

```
cp somewords CopyOfsomewords
```

Alternatively you could use `cat`:

```
cat somewords > CopyOfsomewords
```

Take a look:

```
ls -l
```

Exercise

The command `date` prints the current date. Put the output of date in a file called today. Then use `cat` to look at the content of that file. If you found creating a file difficult then you will certainly find removing one easy. All you do is `rm FILENAME` where `rm` means *remove*. Moving a file to some other place is not a troublesome task either. All you need to know is where to put it. The command below shows an example of moving a file.

```
mkdir stuff
cd stuff
cat > ToDo
Go Shopping
Catch Joker
Make Dinner. # Return and then Ctrl-d
cd ..
mkdir MustDoThings
mv stuff/ToDo MustDoThings
ls MustDoThings
```

Here we use `mv` like this `mv PATH/FILENAME OTHERPATH/`. We can also use `mv` to rename files. The `mv` command can also be used to rename directories.

```
mv DIRECTORY1 DIRECTORY2
```

Exercise

Use `mkdir` to make some subdirectories and sub subdirectories in your home directory. then put a file in one of them. Move that file around and verify it by using `pwd` and `ls -l` commands. `cp` can be used in the same way as `mv` just that it copies instead of moving. When you feel you're done, remove the files (in this exercise) from the directories you created using `rm` and remove the directories (in this exercise) using `rmdir`. Note that the directories can be removed only if they are empty. If you want to play it rough, you can actually say `rm -rf` and the directory will be removed even if it contains files.

Warning: Don't use `rm -rf` if you don't know exactly what you are doing.

The Path

As you might have noticed I used the word path without really explaining exactly what it is. The path to a file is the sequence of directories one has to traverse from root to that file. Giving a full path in front of the file name is

a good thing because it enables us to access the file no matter where in the file tree we are. For example

```
cat /home/student1/MustDoThings/ToDo
```

lets me look at the file no matter where in the file tree we are in. Note that the full path always begins with `/`. However, if I'm standing in the directory `/home/student1` I might find it a bit cumbersome to type the full path, so I can just type

```
cat MustDoThings/ToDo
```

This kind of path is called relative path (relative to the path where you are standing in).

Finding files

So now we know how to create, copy, delete and move files. However a file is no good to us if we can't find it. In most Linux system we use the command `find` to search for our files.

```
cd ~/test
find .. -name 'ToDo'
# Output: ../MustDoThings/ToDo
```

Let me walk you through this one. The first parameter (..) is the place in the file tree where we would like to originate our search from (here parent directory). We can use either a relative or absolute path. The second parameter tells find to search for a file named similar to the third argument which in this case is a pattern. A pattern is enclosed in single quotes (`'`). In this example we searched for a file named ToDo.

Exercise

Create a new directory and in that a new subdirectory. Create a file, using `cat`, in the most newly created directory. Go to home and then search for the file you just created. Can you find it? Go to different positions and search for the file and give different arguments as well. Does it make sense? If you find it easy then read about `find` in the `man` pages by doing

```
man find
```

and try some of its complex options. One handy option is that you can search for file when you only partly remember the name. For example you can search using 'To*' to find ToDo meaning that it searches for all the files starting with 'To'.

Summary

cat printing input to output

COMMAND > FILENAME put the output of the `COMMAND` into a file named `FILENAME`

cp FILENAME1 FILENAME2 makes a copy of `FILENAME1` called `FILENAME2`

mv FILENAME1 FILENAME2 renames `FILENAME1` to `FILENAME2`

mv FILENAME1 PATH/FILENAME2 moves the file `FILENAME1` to the specified `PATH` and names it to `FILENAME2`

rm FILENAME removes `FILENAME`

rmdir DIRNAME *remove subdirectory* `DIRNAME`

find PATH -name PATTERN search, starting from `PATH` for a file that has a name that matches `PATTERN`.

1.1.3 Miscellaneous tricks and commands

This section is an add on to the two previous sections to enhance your skills as a Linux user.

Tab expansion

During the time you use Linux you will encounter hundreds of commands and it isn't easy to remember them all. Don't worry, tab expansion comes to the rescue! Just type the first few letters of the command and press tab twice. This way you get all the commands starting with those letters. It also works for paths. Say for instance you want to go to `/export/home/linus` then typing `cd /e` followed by tab(s) will list all the possible subdirectories. The expansion also works with filenames, try for instance:

```
cd # cd is the same as cd ~ (go to home directory)
cat some
```

followed by tab or double tab.

Looking at files again

There are two other commands to look at the contents of a text file. They are:

```
more FILENAME
less FILENAME
```

You exit them by pressing `q`.

Exercise

Create a long file with many lines. Then look at it in `more` and `less`. By pressing `h` inside `less` you get the online help. When you are tired of these commands you will understand the geek joke "less is more".

Exercise

Do `man ls` and try some of its options (`-l` is just one of them).