BASIC INTRODUCTION TO THE SHELL

Mikel Egaña Aranguren (m.egana@stud.man.ac.uk -
www.sindominio.net/~pik)

February-2004

Contents

1 Introduction

This is just a tiny guide to basic tools in the UNIX typical terminal, I'm doing it on the go so I don't take any responsibility on the damage you can cause to the your computer using this guide, for example missusing the `rm` command ;-), or in any wrong information. Of course, there are much more commands than the ones described here.

You can find plenty of manuals in the internet, a good one that covers a lot is:
http://www.rute.sourceforge.net

The terminal, also known as *prompt*, *black screen*, *shell*, etc., is a especial program that deals with input (commands typed in the keyboard) and does things with that input. As you can imagine, there are plenty of different shells, and the one we are using is called BASH (Bourne Again Shell), because is derived from BSH (Bourne Shell) a typical UNIX shell. This is not really important, just historical background.

2 Basic commands

2.1 Before we see the commands...

First of all, know that in every UNIX system there is a really helpfull and massive manual that can be accessed just typing `man` and the command or program we would like to know about, for example if we want to know about the command `ls`:

```
me@myShell: man ls
```

Will give us all the information available about that command or program. To exit of the man page just type `q`.

Second, the UNIX directories:

- The directory currently I'm on: `./`
- The directory up of my current directory: `../`
- Directories within directories (`dirth` is within `dirw` which is within `dir1`): `/dir1/dirw/dirth`

If you are in `dirth`, `dirw` would be `../dirw/`.

2.2 Some basic commands

ls show the files. For example:

```
me@myShell: ls -l
```

Will show the files of the current directory in list format.

```
me@myShell: ls ../
```

Will show the files of the upper directory, or even the upper of the upper if we want (`ls ../../`).

```
me@myShell: ls -a
```

Will show *all* the files, including the *hidden* ones. The hidden files are files that start with dot (like `.MyHiddenFile`) and therefore they are not listed. Normally they are configuration files (it doesn't make any sense showing them in every `ls` you do).

rm rm files. Be carefull with what you remove, in UNIX there is no "bin" from where to get the deleted files. If you delete something, is forever.

```
me@myShell: rm myfile
```

cp copy files from an origin to a destiny.

```
me@myShell: cp origin destiny
```

Copy from the upper directory to the one I'm currently on:

```
me@myShell: cp ../file.txt ./
```

Copy a file from the current directory and change the name, so we have another copy of the same file but with another name:

```
me@myShell: cp file.txt othername.txt
```

mv move files from an origin to a destiny. The syntax is like `cp`, you can use it to change the name of a file:

```
me@myShell: mv file.txt othername.txt
```

cd change de directory.

mkdir create a directory.

rmdir remove a directory.

pwd it tells you in which directory you are (the whole path from the root directory, `/`).

2.3 A bit more interesting commmands

less is used to read the content of files:

```
me@myShell: less file.txt
```

Will show all the content of the file. Obviusly it can just deal with files that don't content binary data:

```
me@myShell: less file.java
```

Will show you source code of that java program, but

```
me@myShell: less file.class
```

Will show nothing.

nano, pico, joe little editors with just minimun functionality for fast edition of files. In our case use `pico`.

vim, emacs big powerfull editors with plenty of functionality for big projects (programming, essays, ..). If you have got time have a look, they are quite good, although a bit difficult to learn.

touch create a file.

locate search a file.

find search a file.

chmod change the permission of a file:

me@myShell: chmod a+x file

Will change the permissions so *everyone* (a) can *execute* (x) that file in your directory. Bad idea.

`./` execute some programs (other can be exectued just by typing their name).

See also: `tar`, `gzip`, `who`, `ssh`, `sftp`, `ln`, `tail`, `grep`, ...

3 Funny stuff: complex tasks in one line

The power of the shell is given by combining commands and regular expressions. As we know from the PERL lecture, regular expressions are expressions that match patterns. For example, imagine that in a directory you have two kinds of files, some with the file extension `.java` and some with the file extension `.html`, and you just want a list of the files with the `.java` extension. You type:

```
me@myShell: ls *.java
```

And this will give you just the files with that extension (for example, `rm *` would remove everything, as everything matches `*`). Imagine what you can do with all the available regular expressions in just one line!

Another usefull thing are *pipelines* and *redirections*: pipelines (the symbol is `|`) connect the output of a command with the input of another, so the output of a command becomes the input of another:

```
me@myShell: ls|less
```

Gives the output of `ls` to `less` instead of sending it to the standard output (The display), so `less` reads it as if it was a single file and shows the list of files in the display; if we are exploring a big directory it can be usefull.
Redirections(`>` and `>>`) are like pipelines but instead of directing the output to another command, they redirect it to a file, creating it and writing on it:

```
me@myShell: ls>myfile
```

Would create a file called `myfile` with the list of files on the current directory. `>>` is the same but instead of overwritting the original file adds new lines to it (if myfile already exists) with the output from `ls`.
Now we combine everything:

```
me@myShell: ls | grep .java > myfile
```

`grep` is a command to find patterns. So, in this line we execute `ls`, we give its output (the list of all the files) to `grep`, which tries to find the given pattern (in this case, `.java`) and we redirect its output to a file called myfile. So in just one line we have put all the files with a certain characterisitic (`.java`) in a file. The same in a Windows interface would be much slower. (`ls *.java > myfile` would give the same result, I just want you to note the use of `grep` and `|` and `>` in the same line).

4 Supreme fun: shell scripting

Surprise, the shell has got its own programming language, to make it even more powerfull. I have just learned a bit of it, I just want you to be aware that it exists.

It is called shell-script: you can combine command and standard stuff of a programming language (loops, if statements, etc..) in a single file, the shell script program. This programs end up with the `.sh` extension, they must be executable (remember, chmod +x) and they can be executed using `./`. The start of the file is always the following line:

```
#!/bin/sh
```

The path to the interpreter, like in perl.

Here are some small shell-scripts I have created; the first example is a shell-script that removes the cache of the web browser, called noCache.sh, so I can see properly the changes in my web page:

```
#!/bin/sh
rm ~/.mozilla/default/viimu6d4.slt/Cache/*
```

Another example is a shell script that creates a plain file with executable permissions and it gets a name as an argument: executing this script like this:

```
me@myShell: ./newPerl.sh exercise.pl
```

Will create and executable perl file called exercise, ready to write source code on it. Here is the source code of the script:

```
#!/bin/sh
touch $1
chmod +x $1
```

Note that the especial variable `$1` refers to the first argument given to the script. Obviusly there are much more interesting things that can be done with shell-script, just discovered it!