# Tiny Introduction to Linux

Roland Philippsen

August 25, 2011

## 1 Basic Concepts

### 1.1 The Filesystem

The notion of filesystem is a powerful abstraction under UNIX. Virtually any resource appears as an entry in the filesystem.

The directory structure has only one "root" which is simply called "`/`", contrary to MS-based systems where you need to specify e.g. the hard drive "`C:\`" or similar. File and directory names can either be absolute (when they begin with a slash "`/`") or relative (when they don't). For relative names, the system uses the current working directory to determine where to start looking for the name. Directory levels are separated by slashes in the path. The current working directory is called "`.`" and the directory which is just above is called "`..`".

### 1.2 The Command Line

Commands are entered at the *prompt*, which should look something like "`root@lsa1sp2: #`". Your commands are interpreted by the *shell*, which is like a translation layer between the user and the operating system. The shell has quite some additional functionality, some of which is:

**Completion** You can type just the beginning of commands and file names, and then press `TAB`. The shell will try to determine what you intent to type and complete the word for you. The completion might not go until the end of the word if there is more than one possibility. Hitting `TAB` twice will give you a list of possible completions (if you're using the bash shell — for users of csh or tcsh the completion list is shown when you press `CTRL-D`).

**History** With the cursor keys `UP` and `DOWN`, you can browse through the commands you've typed so far. Typing "`history`" gives you a numbered list of past commands which you can then repeat using the form "`!123`" where 123 is the number of the command to be repeated. For the more advanced history features, refer to the manual pages of the shell you're using, probably "`man bash`".

**Line Editing** The `LEFT` and `RIGHT` cursors work as expected, and you can edit the current command line using the usual `BACKSPACE` and `DELETE` keys. Other movement and editing commands are:
`ALT-b` / `ALT-f` : backward / forward one word
`CTRL-a` / `CTRL-e` : beginning / end of line
`CTRL-k` : erase everything from the cursor to the end of line
`ALT-BACKSPACE` : erase one word backward
`ALT-d` : erase one word forward
`CTRL-y` : paste what you last erased at the cursor position

**Job Control** You can launch processes in the **background** by appending "`&`" to them. This will immediately give you back the prompt and is most useful for launching applications that create their own new window. But it can be confusing when applications which run in the background write something on their standard output or standard error file (it will all be echoed on the terminal). Similarly, you can **suspend** a running application by pressing `CTRL-z`, and if desired you can then continue it in the background by typing "`bg`" at the prompt. If you want the application to become your foreground process again, simply type "`fg`". Note that you can have several jobs in the fore- and background, to see a list you issue the command "`jobs`". You can use the numbers of that list to send specific jobs to the fore- or background, e.g. "`fg 2`" to put the second job in the foreground.

Most commands accept arguments and options. Arguments define what the command should act on (typically a file name), and options change the behavior of the command. Options are normally single letters preceded by a dash "`-`". For example, let's look at the command for listing directory contents:

| | |
|---|---|
| `ls` | prints all files and directories of the current working directory, |
| `ls -l` | does the same, but because of the "-l" option, it prints more information about each entry, |
| `ls /etc` | lists all files and directories that are in the directory "`/etc`". |

# 2 Some Useful Commands

Here is a very short description of a few commands. For more information, type "`man cmdname`" (replacing `cmdname` with the command you want to know more about).

## 2.1 Browsing

| | |
|---|---|
| `cd` | Change working directory. |
| `ls` | List the current directory entries. Useful options are:<br>"`-l`" print lots of information for each file,<br>"`-a`" show hidden files. |
| `pwd` | Print the current working directory. |
| `file` | Determine the type of a file. |
| `less` | Display files by screen fulls, the up and down cursors act as expected.<br>Press `SPACE` for the next screen, "`q`" to quit, "`h`" for help, "`g`" to jump to the beginning,<br>"`G`" to jump to the end, "`b`" to go back one screen. |
| `head` | Display the first 10 lines of a file, or specify the number of lines using "`-n`". |
| `tail` | Like "`head`", but displays the end of a file. |

## 2.2 Modifying / Copying Files and Directories

| | |
|---|---|
| `cp` | Copy a file to another file or directory, "`-r`" recursively copies directories. |
| `mv` | Move (rename) a file to another file or directory. |
| `rm` | Remove a file<br>**Warning** for people used to Windows:<br>files are *really erased*, not just put in a "Recycle Bin" |
| `mkdir` | Create a directory |
| `rmdir` | Remove a directory. This only works if the directory is empty.<br>If you're really absolutely sure of what you're doing, you can use<br>"`rm -rf YOUR-DIRECTORY-NAME`" to recursively erase a directory and everything underneath. |

## 2.3 Other Commands

| | |
|---|---|
| `apropos` | Find the manual pages containing a given keyword. |
| `man` | Print the manual page of a command. |
| `passwd` | Change your password. |
| `ps` | List running processes of the current terminal.<br>If you want to see all processes of the current user, use "`-a`".<br>If you're still not satisfied with the output, try "`-e`". |
| `kill` | Stop a process by passing its process-id (shown by `ps` as `PID`).<br>You need to specify the kind of `kill`, you should usually first try "`kill -TERM 1234`".<br>If that doesn't work, "`kill -9 1234` (where you need to replace `1234` with the actual PID). |
| `tar` | Create / expand / query archives. Typical invocations are<br>"`tar cfv arch.tar somedir/`" to put the contents of a directory into an archive, and<br>"`tar xfv arch.tar` to extract the files into the directory (creating it or overwriting existing files). |
| `bzip2` | Compress a file (other, less powerful, compression tools are `gzip`, `compress` and `zip`).<br>Uncompressing can be done using `bunzip2`, or `bzcat` in order to pipe it into another command. |
| `mount` | Make a device visible in the filesystem, typically to access CD-Roms or USB sticks.<br>Just typing "`mount`" without arguments lists the currently mounted file systems. |

Nowadays, these often are automatically mounted and opened in a file browser window.
If that doesn't work, try
"`mount -t iso9660 /dev/hdc /mnt/cdrom`" or
"`mount -t msdos /dev/sda1 /mnt/usbstick`" or variations thereof.
The "`dmesg`" command will usually print out the information needed to find out which device to use.
E.g. `/dev/hdc` works for CD-Rom drives that are secondary masters of the IDE bus.

`umount` Un-`mount` a device from a given point in the filesystem.

# 3 Some Examples

## 3.1 Showing Long Directory Listings

Most commands can have their in- and output redirected from and to other programs. This is done via so-called pipes, which are noted on the command line as "`|`" (vertical bar). Let's look at an example:

```
$ ls -l /usr/bin | less
```

Here, the "`ls`" command is used to print a detailed ("`-l`") listing of the directory "`/usr/bin`". This is where most of the system's programs reside, so this list is bound to be longer than one screen full. In order to read it comfortably, the output from "`ls`" is piped into the "`less`" command by connecting the two with a "`|`" symbol, which makes "`less`" display the output of "`ls`" by screen fulls. Instead of piping into `less`, you can often use `SHIFT-PageUp` and `SHIFT-PageDown` to scroll terminal output.

## 3.2 Looking for a File

A very powerful tool for finding files is the "`find`" command. Here is a very simple example of its use: Suppose you want to find a file that's called something with "bak" in the middle but you don't remember what it begins and ends with. But you remember that it was somewhere under your home directory "`/home/robot`". This is the "`find`" command and options to use in that case:

```
$ find /home/robot -name '*bak*'
```

The path specification can, as usual, be absolute or relative. For instance, if you want to search under the current directory, you use the shortcut name "`.`" which is always replaced by the current directory. For example

```
$ find . -type d
```

lists all sub-directories of the current directory.

Another common problem is finding the file which contains a certain keyword or pattern. That is a typical application of the "`grep`" command, which looks at files and prints out the lines which match the pattern. A very simple example: Suppose somewhere in "`/home/robor/documents`" there is a file which contains the keyword "`bugs`", but you don't remember which one. No problem, just type:

```
$ grep -r bugs *
```

The "`-r`" options tells grep to look into sub directories as well, and the asterisk "`*`" tells it to look at all files. (Actually, the "`*`" is expanded by the shell and passed as a list of file names to the grep command.)

## 3.3 Creating Compressed Archives

The archiving facility `tar` can create compressed archives using a command line argument "`z`", but this uses `gzip` compression and doesn't work on all Unixes. Another way, which illustrates how commands can be stringed together on the command line, is

```
$ tar cv somedir/ | bzip2 > somedir.tar.bz2
```

to create an archive of the directory `somedir`, compress it using `bzip2`, and writing the output to `somedir.tar.bz2`. To extract the resulting archive, use

```
$ bzcat somedir.tar.bz2 | tar xv
```