C Programming under Linux

P2T Course, Semester 1, 2007–8

Linux Part 1: Introduction and Getting Started

Dr Graeme A Stewart

Room 481, Department of Physics and Astronomy

University of Glasgow

g.stewart@physics.gla.ac.uk

Summary

- A Shorter History of Linux and Unix
- The Linux Kernel
- User Space Programs
- Kernel Boot Process
- Command Line Survival
- Files and Directories
- Logout

`http://moodle2.gla.ac.uk/physci/moodle/`

`$Id: linux-lecture-01.tex,v 1.19 2007/10/11 15:45:41 graeme Exp $`

In The Beginning...

- The origins of Linux are in an operating system called `UNICS` c. 1969.
- This system developed into `UNIX` through the 70s – in parallel with a new programming language written for its operating system, `C`.
- Unix was developed initially on hugely expensive computers costing millions of $, £ or Euros.
- Unix became very popular in academic institutions, where a pioneering spirit of cooperation existed – thus Richard Stallman was able to establish the GNU (Gnu's Not Unix) software project in 1983, which developed *free* programs mainly for unix machines.
- However as computer hardware got cheaper and better even lowly PCs were capable of running unix-like systems – if they were 386s. (And Linux still runs usefully on such humble machines!)

Linux is Born

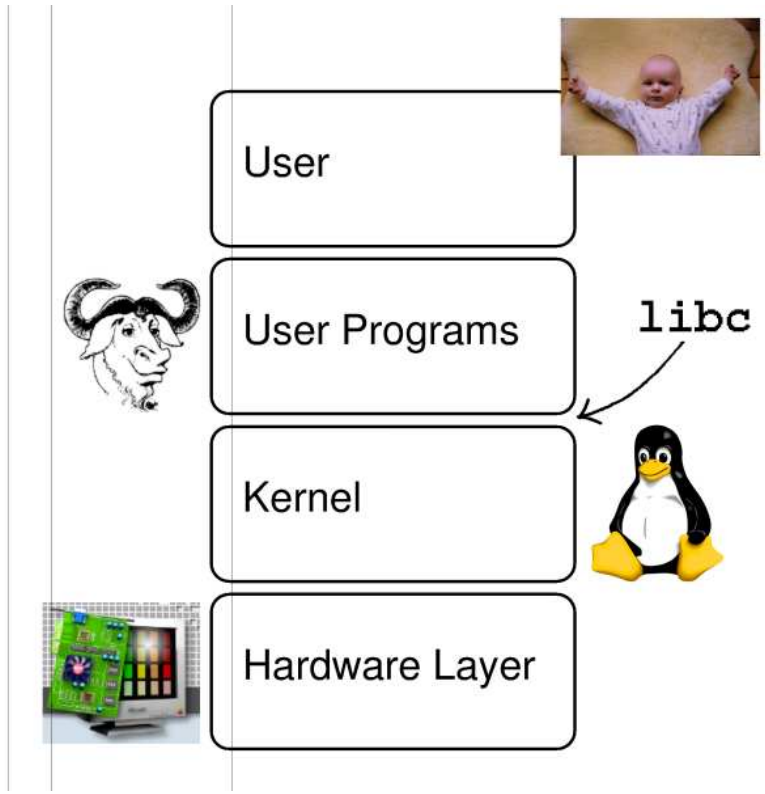
- In 1991, on `comp.os.minix`, a discussion group for a limited unix clone called minix, a Finnish student called Linus Torvalds announced he had created a new unix clone called Linux – this was version 0.01!
- He invited other people to join in the effort to make Linux better – calling on the same spirit of cooperation that had made the GNU software tools possible.
- Linux was a hit – soon Linux had hundreds of developers and within a few years it had reached a 1.0 release.
- 16 years later, and at version 2.6.22, Linux is one of the most successful software projects ever, having produced a fast, secure, stable operating system. Anyone can view the source code of Linux and modify it if they want.

So what is Linux?

- All computers run a core program, called the *Kernel*. This program controls access to the hardware, manages other programs and allocates system resources.
- *Linux* is a kernel program written by Linus Torvalds and many other people.
- It is a ‘unix-like’ kernel, and runs on a great variety of processors: x86, sparc, alpha, itanium, opteron, etc. It now supports almost every type of hardware which is found in these architectures: PCI buses, network cards, video cards, hard drives and controllers, etc.
- It is licensed under the *GNU Public License*, which ensures that no-one can remove users’ rights to see, modify and redistribute the kernel code.

Where Linux fits into a Computer

Linux, as with any kernel, fits in between the hardware and the programs running on the machine:



Above the Linux kernel user programs run – and above them there is (usually) a user!

GNU/Linux Distributions

On its own the kernel does nothing of great use for an end user. Many other programs are needed. Linux plus a minimal set of useful programs form an *operating system*. Various companies and teams have made *Linux Distributions*, which include a rather more extensive set of programs. Some of the most popular distributions are:

- Debian GNU/Linux
- Scientific Linux (rebuilt from RedHat Linux)
- SuSE
- Ubuntu (a popular and recommended distribution for home use)

When we speak of ‘Linux’ on this course this is usually shorthand for ‘the GNU/Linux operating system’. However, the contribution of the GNU and FSF (‘Free Software Foundation’) projects to making Linux useful (in fact, making Linux *possible*!) should not be underestimated.

Hierarchy of Programs

Programs which run in ‘user space’ – that is all programs which are not the kernel – also form a hierarchy:

- certain programs provide functionality for other programs
- programs have different privilege levels and permissions to get access to low level features of the operating system

Functionality Hierarchy

Try: `$ ps aux`

- Low level operating system programs: `init, kswapd, kapmd`
- Access and authentication services: `getty, login, sshd`
- Basic user enviroment (a *shell*): `bash, csh, ksh`
- Lower level user commands: `ls, cd, grep, gcc`
- Low level graphics server: `X`
- Graphical User Interface Layer: KDE suite (`kinit, konsole`; Gnome (`panel, sawfish`)
- End user applications: `mozilla, xemacs, kate, kspread`

Authorisation Hierarchy

1. `root`: The `root` user in unix is special, and programs running with root permissions can do things others cannot:
   - read or write any file
   - access any hardware device
   - perform privileged network operations
2. other programs: have restricted permissions, generally expressed through the file system.

The authorisation system in unix is, in fact, quite straightforward: `root` processes can do anything, other processes may have elevated privileges granted to them by `root`.

Getting Started

When a Linux machine is booted (on PC architecture) the BIOS passes control to the kernel (usually via a bootloader). The kernel then initialises the system's hardware. Here's an excerpt from a linux kernel boot:

```
Uniform Multi-Platform E-IDE driver Revision: 7.00beta4-2.4
ide: Assuming 33MHz system bus speed for PIO modes; override with idebus=xx
hda: IC25N020ATCS04-0, ATA DISK drive
hdc: HL-DT-STDVD-ROM GDR8081N, ATAPI CD/DVD-ROM drive
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
ide1 at 0x170-0x177,0x376 on irq 15
hda: attached ide-disk driver.
hda: host protected area => 1
hda: 39070080 sectors (20004 MB) w/1768KiB Cache, CHS=2584/240/63
Partition check:
/dev/hda: p1 p2 p3 p4 < p5 p6 p7 >
Initializing Cryptographic API
[...]
```

Logging in

Once the kernel has started, it begins to run userspace programs (in ‘protected mode’). The first program it runs is called `init`. This starts the system boot process, beginning daemons, services and other programs; configuring the network and performing other initialisations. Once the system is ready `init` will start a `getty`, which presents a login prompt:

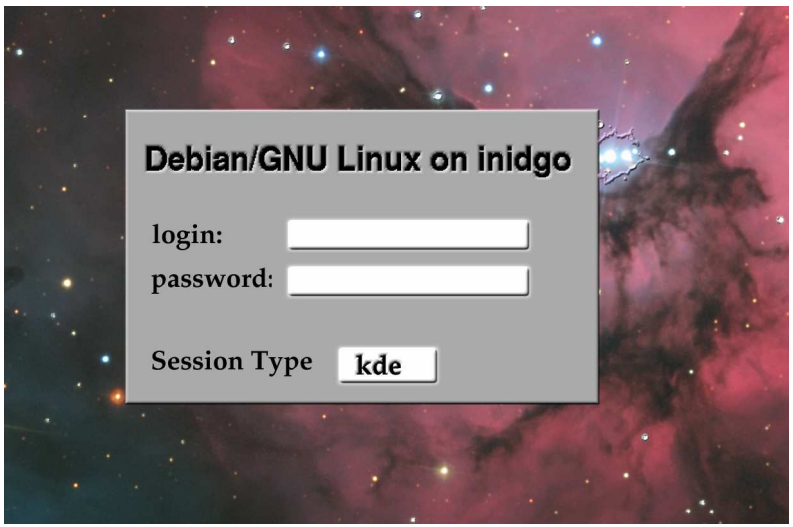
```
Debian GNU/Linux stable/woody vc/1

alpha login: graeme
Password: *********                    # These *s don't actually appear!
Welcome to Debian GNU/Linux
alpha$ _
```

If you login sucessfully you get a *shell prompt*:

```
alpha$
```

Graphical Login

Most Linux machines won't actually stay at the console login, they'll also start a *graphical login manager*:



However, when something goes wrong, or someone else is logged on at the graphical console, it's important to know that the plain console logins are always there. Pressing `CTRL-ALT-F1,2,3,4,5,6` will switch to *virtual console* 1–6, (even when a graphical session is running). `ALT-F1-6` is sufficient to change consoles and `ALT-F7` will switch back to X.

Command Line Survival++

```
alpha$ help?
bash: help?: command not found
```

- A lot of your interaction with Linux will be through the command line of the shell – it's important to know how to use it properly.
- At its simplest the shell will read a command that you type and give you back the results:

```
alpha$ date
Sun Oct  1 15:43:22 BST 2006
```

- We'll concentrate on the unix shell called `bash`, which is the standard shell in Linux distributions.

Let's now see the some of the core commands you'll need to work with the shell on Linux, and discuss some of its fundamental features.

Anatomy of a Shell Command I

Here's a typical command typed at the shell prompt:

```
$ ls -l foo.c
^  ^  ^  \-> "argument"
|   \  \-> "option"                    # Also called a "switch"
 \   \-> "command"
  \-> "prompt"
```

- The shell 'prompts' for input by displaying a standard text string. The prompt in `bash` can be customised, but should always end in a `$`.
- The first thing typed by the user is the command – in this case `ls`, which *lists* files.
- There can then follow options – these modify the way that the command works and are identified by a dash. In this example `-l` signals a 'long listing'.
- Then there follow *arguments* – more infomation passed to the command to process. Often arguments are files that the command will operate on; in this case `foo.c`.

Anatomy of a Shell Command II

What happens when we ask the shell to execute our command?

```
$ ls -l foo.c                    # Press RETURN
-rw-r--r--  1 graeme knigits  3392 2003-10-11 22:48 foo.c
$
```

- Once you've finished typing a command press RETURN ↵ (aka ENTER) and the shell will execute your command.
- In this case it 'long lists' the file `foo.c`, giving us detailed information about the file (we'll explain the details of just what `ls` is telling us later).
- After the shell has finished with that command it prints another prompt and will await further input.

Directories

Here's what you need to know to move around directories:

- `ls foo` list contents of a directory `foo`
- `cd foo` change directory to `foo`
- `pwd` print current working directory

And to create and destroy directories:

- `mkdir foo` make directory `foo`
- `rmdir foo` remove directory `foo`

N.B. A directory must be empty for `rmdir` to work.

Paths I

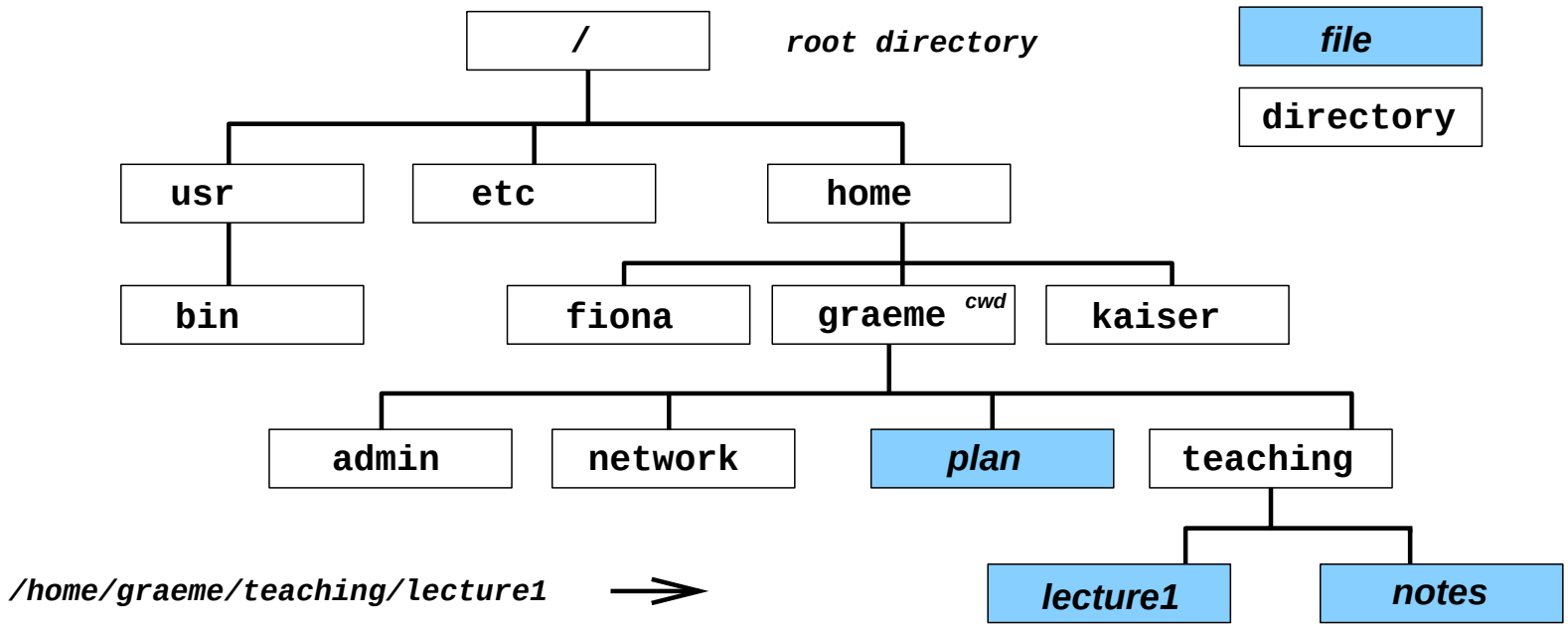
All of the directory commands above can take *path arguments*. If no argument is present they, generally, operate on a sensible default.

- `ls` list the current working directory
- `ls foo` list contents of a directory `foo`, (which is a subdirectory of the current working directory) [1]
- `ls foo/bar` list the contents of directory `bar`, which is a subdirectory of `foo` [1] (N.B. The directory separator in unix is '`/`')
- `ls /foo` list the contents of directory `foo`, which is a subdirectory of the *root directory*

[1] Any of the arguments to `ls` can be files, rather than directories, in which case just those files are listed.

Paths II

The root directory (/) lies at the top of the whole unix filesystem:



Any path which begins with '/' is called an *absolute path*: `/home/graeme/teaching/lecture1`

Any path which does not start with '/' is called a *relative path*: `teaching/lecture1` Relative paths are always accessed with respect to the *current working directory*.

Some Special Directories

Notice in the last example how the three users of the system had their home directories in `/home/graeme`, `/home/fiona` and `/home/kaiser`.

These are examples of users' *home directories*. Under a Linux system a user stores (almost) all their files in their home directory.

As it's so important the shell assigns a special character to mean 'my home directory': `~`.

```
$ ls ~/bin/frobnicate
/home/graeme/bin/frobnicate
```

User foo's home directory is `~foo`:

```
$ ls ~foo/bar.c
/home/foo/bar.c
```

The absolute path to your home directory is also stored in a shell variable called `$HOME` – more about variables later.

Some More Special Directories

Every directory has two special directory entries "`.`" and "`..`". These are:

- "`.`" this directory
- "`..`" this directory's parent

```
$ pwd
/home/graeme/life/work
$ ls
research    teaching
$ ls -a
.         ..        research    teaching
$ ls -a ..
.         ..        play        sleep         work
$ cd ../play
$ pwd
/home/graeme/life/play
```

Paths can have an arbitary number of `..` and `.` path elements.

Hidden Files and Directories

Notice that the `-a` option (view *all*) to `ls` had to be used to see the special `.` and `..` directories.

In fact any file or directory which starts with a dot is *hidden*, and `ls` will not list it by default.

Programs which require, or enable, customisations can store these in such hidden files – consquently inside a user's home directory there will be many, many hidden files. Of particular note are:

- `.bash_profile, .bashrc` customisation files for `bash`.
- `.emacs` customisation files for the emacs editor.
- `.kde` a directory containing options and settings for the KDE graphical environment.

Files: Viewing

A lot of the basic file manipulation commands in unix are oriented towards the manipulation of *text* files.

- `cat foo` print file `foo`
- `cat foo bar` concatenate (print) files `foo` and `bar`
- `less foo` display the file `foo` through the pager

`less` is invaluable for viewing files. It takes more than one page of input and displays it page by page on the screen – `SPACE` moves down by 1 page and the `PAGE UP/DOWN` and arrow keys work as expected. Type `q` to quit.

Typing `/foo` will search forwards for the next occurence of `foo`.

Typing `?foo` will search backwards.

Type `h` in `less` for more options.

Files: Moving and Copying

To move and copy files in Linux use `mv` and `cp`:

- `mv foo bar` Move (rename) the file `foo` as `bar`. (`bar` is overwritten, if it exists.)
- `mv foo /tmp` Move the file `foo` into directory `/tmp` (ends up as `/tmp/foo`).
- `cp foo bar` Copy `foo` to `bar`. (Again, `bar` is overwritten.)
- `cp foo /tmp` Copy the file `foo` into directory `/tmp`

Note that `mv` and `cp` behave differently if the final argument is a file or a directory. If the last argument is a directory one can give multiple files to operate on:

```
$ ls
foo bar
$ mv foo bar /tmp
$ ls
$ ls /tmp
foo     bar
```

Files: Removing

The `rm` command will remove a file:

```
$ ls
foo        bar
$ rm foo
$ ls
bar
```

Note that unix does not blow whistles, ring bells, pop up alerts or phone your mother for confirmation when you ask for a file to be deleted. It just does it – generally unix does not patronise its users.

If you want to get confirmation then use the `-i` option:

```
$ rm -i foo bar
rm: remove ‘foo’? y
rm: remove ‘bar’? n
$ ls
bar
```

The `-r` option recurses through subdirectories and the `-f` options forces removal (e.g. of directories). (`rm -fr` is very useful, but potentially dangerous!)

Logging Out

When you have finished working with a shell you can logout with:

1. The command `exit`.
2. The command `logout` – but only if you are in a login shell. (Logins on virtual consoles are login shells, those in `X`, the unix windowing system, are not.)
3. Typing `CTRL-D` – as long as the shell is not set to `IGNOREEOF`.

If you are in a graphical session, there's usually a logout button or option:

Copyright

All these notes are Copyright (c) 2003-2007, Graeme Andrew Stewart.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is here:

http://www.physics.gla.ac.uk/p2t/fdl.txt