

Computing and Statistical Data Analysis (PH4515, UofL PG Lectures)

Glen Cowan

Physics Department

Royal Holloway, University of London

Egham, Surrey TW20 0EX

01784 443452

`g.cowan@rhul.ac.uk`

`www.pp.rhul.ac.uk/~cowan/stat_course.html`

Outline

1st two to three weeks will be a crash course in C++

- Quick overview of the important stuff

- Use UNIX (Linux) environment

- Intro to tools like ROOT, gmake, debugger

From week 3, statistical data analysis

- Probability, random variables, Monte Carlo methods

- Statistical tests

- Parameter estimation

Data analysis exercises will use C++ tools

Coursework, exams, etc.

For C++ part

Computer based exercises -- see course web site.

For data analysis part

More exercises, many computer based

For PH4515 students

Written exam at end of year (70% of mark),
no questions on C++, only statistical data analysis.

For PhD students

No material from this course in exam

C++ Outline

Approximately by lecture for 1st two to three weeks:

- 1 Introduction to C++ and UNIX environment
- 2 Variables, types, expressions, loops
- 3 Type casting, functions
- 4 Files and streams
- 5 Arrays, strings, pointers
- 6 Classes, intro to Object Oriented Programming
- 7 Memory allocation, operator overloading, templates
- 8 Inheritance, STL, **gmake** , **ddd**

Some resources (computing part)

There are many web based resources, e.g.,

`www.doc.ic.ac.uk/~wjk/C++Intro` (Rob Miller, IC course)

`www.cplusplus.com` (online reference)

`www.icce.rug.nl/documents/cplusplus` (F. Brokken)

See links on course site or google for “C++ tutorial”, etc.

There are thousands of books – see e.g.

W. Savitch, *Problem Solving with C++*, 4th edition
(lots of detail – very thick).

B. Stroustrup, *The C++ Programming Language*
(the classic – even thicker).

Lippman, Lajoie (& Moo), *C++ Primer*, A-W, 1998.

Introduction to UNIX/Linux

We will learn C++ using the Linux operating system
Open source, quasi-free version of UNIX

UNIX and C developed ~1970 at Bell Labs

Short, cryptic commands: `cd`, `ls`, `grep`, ...

Other operating systems in 1970s, 80s 'better', (e.g. VMS)
but, fast 'RISC processors' in early 1990s needed a cheap
solution → we got UNIX

In 1991, Linus Torvalds writes a free, open source version
of UNIX called Linux.

We currently use the distribution from CERN



Basic UNIX

UNIX tasks divide neatly into:

- interaction between operating system and computer (the kernel),
- interaction between operating system and user (the shell).

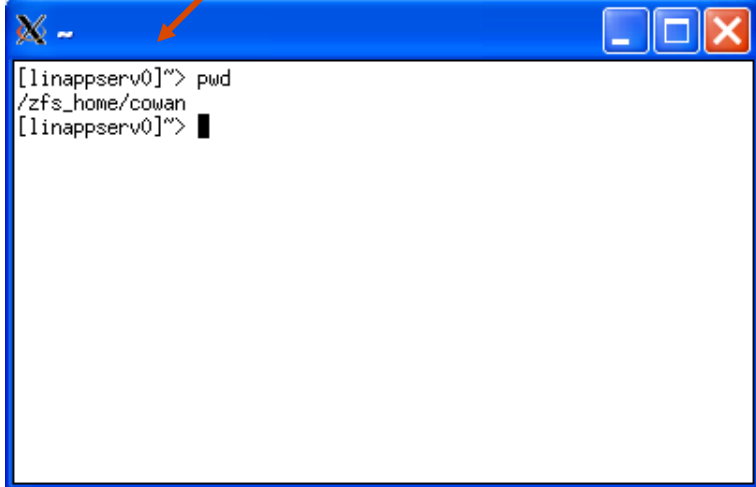
Several shells (i.e. command sets) available: sh, csh, tcsh, bash, ...

Shell commands typed at a prompt, here `[linappserv0]~>` often set to indicate name of computer:

Command `pwd` to “print working directory”, i.e., show the directory (folder) you’re sitting in.

Commands are case sensitive.

`PWD` will not work .

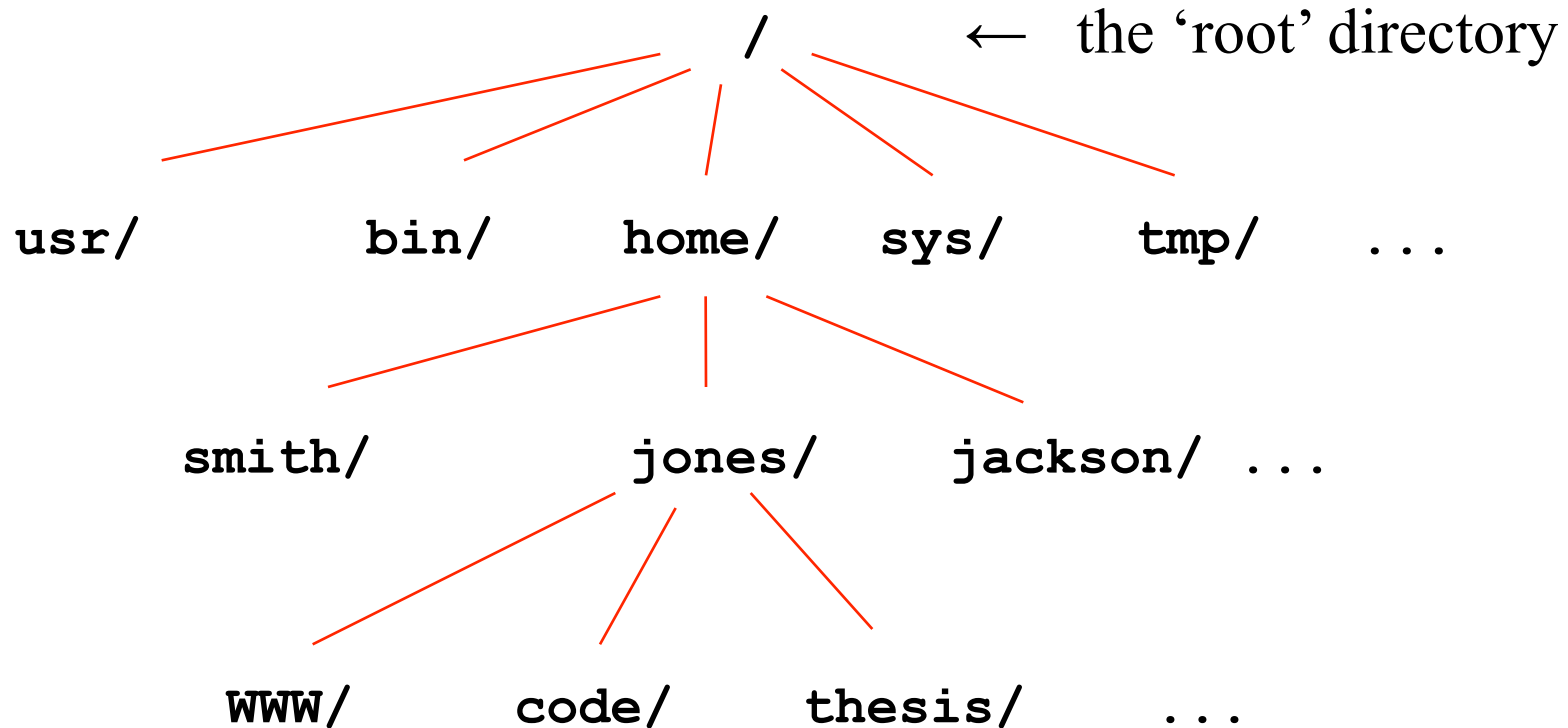


```
[linappserv0]~> pwd
/zfs_home/cowan
[linappserv0]~> █
```

A terminal window with a blue title bar and standard window controls. The prompt is `[linappserv0]~>`. The user has entered `pwd` and the output is `/zfs_home/cowan`. The prompt is now `[linappserv0]~>` with a cursor. An orange arrow points from the text above to the prompt in the terminal window.

UNIX file structure

Tree-like structure for files and directories (like folders):



File/directory names are case sensitive: **thesis** \neq **Thesis**

Simple UNIX file tricks

A complete file name specifies the entire ‘path’

```
/home/jones/thesis/chapter1.tex
```

A tilde points to the home directory:

```
~/thesis/chapter1.tex ← the logged in user (e.g. jones)
```

```
~smith/analysis/result.dat ← a different user
```

Single dot points to current directory, two dots for the one above:

```
/home/jones/thesis ← current directory
```

```
../code ← same as /home/jones/code
```

A few UNIX commands (case sensitive!)

<code>pwd</code>	Show present working directory
<code>ls</code>	List files in present working directory
<code>ls -la</code>	List files of present working directory with details
<code>man ls</code>	Show manual page for ls . Works for all commands.
<code>man -k <i>keyword</i></code>	Searches man pages for info on “keyword”.
<code>cd</code>	Change present working directory to home directory.
<code>mkdir <i>foo</i></code>	Create subdirectory <i>foo</i>
<code>cd <i>foo</i></code>	Change to subdirectory <i>foo</i> (go down in tree)
<code>cd ..</code>	Go up one directory in tree
<code>rmdir <i>foo</i></code>	Remove subdirectory <i>foo</i> (must be empty)
<code>emacs <i>foo</i> &</code>	Edit file <i>foo</i> with emacs (& to run in background)
<code>more <i>foo</i></code>	Display file <i>foo</i> (space for next page)
<code>less <i>foo</i></code>	Similar to more <i>foo</i> , but able to back up (q to quit)
<code>rm <i>foo</i></code>	Delete file <i>foo</i>

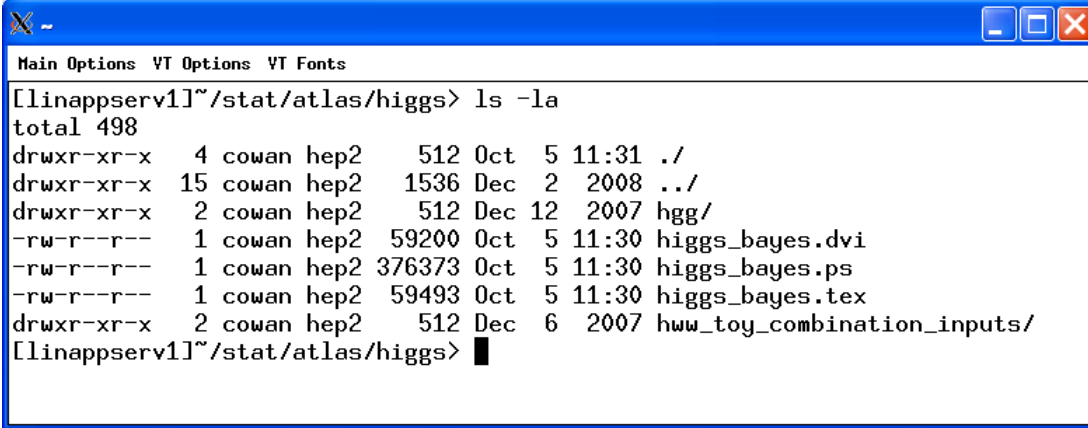
A few more UNIX commands

<code>cp foo bar</code>	Copy file foo to file bar, e.g., <code>cp ~smith/foo ./</code> copies Smith's file foo to my current directory
<code>mv foo bar</code>	Rename file foo to bar
<code>lpr foo</code>	Print file foo. Use <code>-P</code> to specify print queue, e.g., <code>lpr -Plj1 foo</code> (site dependent).
<code>ps</code>	Show existing processes
<code>kill 345</code>	Kill process 345 (<code>kill -9</code> as last resort)
<code>./foo</code>	Run executable program foo in current directory
<code>ctrl-c</code>	Terminate currently executing program
<code>chmod ug+x foo</code>	Change access mode so user and group have privilege to execute foo (Check with <code>ls -la</code>)

Better to read a book or online tutorial and use `man` pages

UNIX file access

If you type `ls -la`, you will see that each file and directory is characterized by a set of file access rights:



```
[[linappserv1]~/stat/atlas/higgs> ls -la
total 498
drwxr-xr-x  4 cowan hep2   512 Oct  5 11:31 ./
drwxr-xr-x 15 cowan hep2  1536 Dec  2  2008 ../
drwxr-xr-x  2 cowan hep2   512 Dec 12  2007 hgg/
-rw-r--r--  1 cowan hep2  59200 Oct  5 11:30 higgs_bayes.dvi
-rw-r--r--  1 cowan hep2 376373 Oct  5 11:30 higgs_bayes.ps
-rw-r--r--  1 cowan hep2  59493 Oct  5 11:30 higgs_bayes.tex
drwxr-xr-x  2 cowan hep2   512 Dec  6  2007 hww_toy_combination_inputs/
[[linappserv1]~/stat/atlas/higgs> █
```

Three groups of letters refer to: user (u), group (g) and other (o). The possible permissions are read (r), write (w), execute (x).

By default, everyone in your group will have read access to all of your files. To change this, use `chmod`, e.g.

```
chmod go-rwx hgg
```

prevents group and other from seeing the directory `hgg`.

Introduction to C++

Language C developed (from B) ~ 1970 at Bell Labs

Used to create parts of UNIX

C++ derived from C in early 1980s by Bjarne Stroustrup

“C with classes”, i.e., user-defined data types that allow “Object Oriented Programming”.

Java syntax based largely on C++ (head start if you know java)



C++ is case sensitive (**a** not same as **A**).

Currently most widely used programming language in High Energy Physics and many other science/engineering fields.

Recent switch after four decades of FORTRAN.



Compiling and running a simple C++ program

Using, e.g., emacs, create a file `HelloWorld.cc` containing:

```
// My first C++ program
#include <iostream>
using namespace std;
int main(){
    cout << "Hello World!" << endl;
    return 0;
}
```

We now need to compile the file (creates machine-readable code):

```
g++ -o HelloWorld HelloWorld.cc
```

↑
Invokes compiler (gcc)

← name of output file

← source code

Run the program:

```
./HelloWorld
Hello World!
```

← you type this

← computer shows this

Notes on compiling/linking

```
g++ -o HelloWorld HelloWorld.cc
```

is an abbreviated way of saying first

```
g++ -c HelloWorld.cc
```

Compiler (-c) produces `HelloWorld.o`. ('object files')

Then 'link' the object file(s) with

```
g++ -o HelloWorld HelloWorld.o
```

If the program contains more than one source file, list with spaces; use \ to continue to a new line:

```
g++ -o HelloWorld HelloWorld.cc Bonjour.cc \  
GuessGott.cc YoDude.cc
```

Writing programs in the Real World

Usually create a new directory for each new program.

For trivial programs, type compile commands by hand.

For less trivial but still small projects, create a file (a ‘script’) to contain the commands needed to build the program:

```
#!/bin/sh
# File build.sh to build HelloWorld
g++ -o HelloWorld HelloWorld.cc Bonjour.cc \
GuessGott.cc YoDude.cc
```

To use, must first have ‘execute access’ for the file:

```
chmod ug+x build.sh      ← do this only once
./build.sh                ← executes the script
```


A closer look at HelloWorld.cc

`// My first C++ program` is a comment (preferred style)

The older ‘C style’ comments are also allowed (cannot be nested):

```
/*  
    These lines  
    here are comments  
*/
```

```
/* and so are these */
```

You should include enough comments in your code to make it understandable by someone else (or by yourself, later).

Each file should start with comments indicating author’s name, main purpose of the code, required input, etc.

More HelloWorld.cc – include statements

`#include <iostream>` is a compiler directive.

Compiler directives start with `#`. These statements are not executed at run time but rather provide information to the compiler.

`#include <iostream>` tells the compiler that the code will use library routines whose definitions can be found in a file called `iostream`, usually located somewhere under `/usr/include`

Old style was `#include <iostream.h>`

`iostream` contains functions that perform i/o operations to communicate with keyboard and monitor.

In this case, we are using the `iostream` object `cout` to send text to the monitor. We will include it in almost all programs.

More HelloWorld.cc

`using namespace std;` More later. For now, just do it.

A C++ program is made up of functions. Every program contains exactly one function called `main`:

```
int main() {  
    // body of program goes here  
  
    return 0;  
}
```

Functions “return” a value of a given type; `main` returns `int` (integer).

The `()` are for arguments. Here `main` takes no arguments.

The body of a function is enclosed in curly braces: `{ }`

`return 0;` means `main` returns a value of 0.

Finishing up HelloWorld.cc

The ‘meat’ of HelloWorld is contained in the line

```
cout << "Hello World!" << endl;
```

Like all statements, it ends with a semi-colon.

`cout` is an “output stream object”.

You send strings (sequences of characters) to `cout` with `<<`

We will see it also works for numerical quantities (automatic conversion to strings), e.g., `cout << "x = " << x << endl;`

Sending `endl` to `cout` indicates a new line. (Try omitting this.)

Old style was `"Hello World!\n"`

Wrapping up lecture 1

We have seen just enough UNIX to get started. Try out the commands from the lecture and have a look at the online tutorials.

We have seen how to compile and run the simplest possible C++ program. Log in, enter the code into a file and get it to run.

If you can't get it to work, shout for help.

Try entering the compile commands into a short script and build the program in this way.

Later we will see a more elegant (read: cryptic) way of building larger programs with a utility called gmake.

Next lecture: variables, types, expressions