Computing lecture 1

Getting started

I. Your computing environment

- operating systems
- networks
- windowing systems

II. Data analysis tools

- PAW

III. Programming

- languages
- compiling, linking

IV. More utilities

- document preparation
- manipulating plots

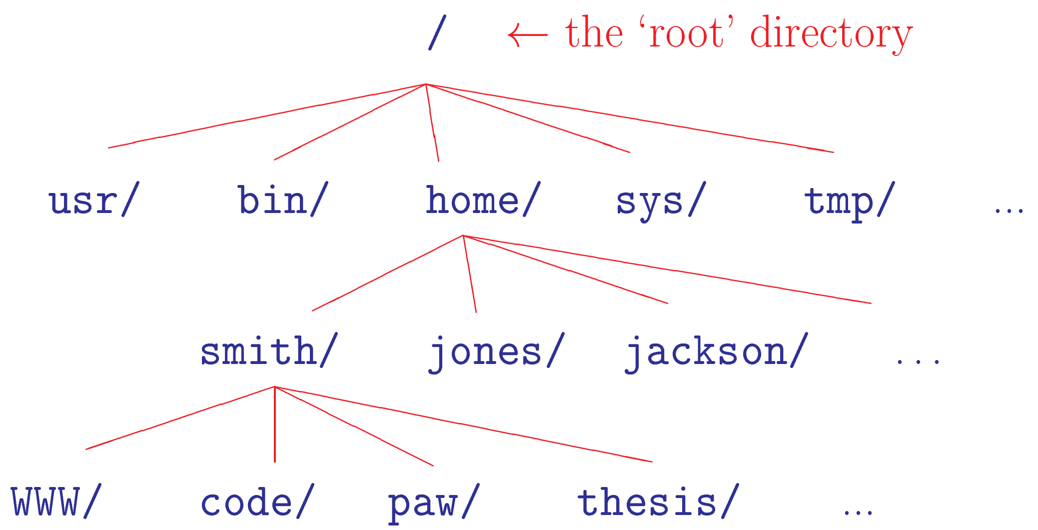
⋮

Operating systems

- Currently most widely used operating system in HEP is Unix:

    many books, online tutorials (see course web site).

- Several shells (i.e. command sets) available: sh, csh, tcsh, bash, . . .
- Shell/environment variables, shell scripts, redirection of i/o . . .
- Tree-like structure for files and directories:



- A complete file name specifies the entire 'path':

    `/home/smith/thesis/chapter1.tex`

- A tilde points to the home directory:

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

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

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

    `~smith/thesis` ← current directory

    `../code` ← same as `~smith/code`

A few Unix commands (case sensitive!)

`pwd`	Show present working directory
`ls`	List files in present working directory
`ls -l`	List files of present working directory with details
`man ls`	Show manual page for `ls`. Works for all commands.
`cd`	Change present working directory to home directory
`mkdir` *foo*	Create subdirectory *foo*
`cd` *foo*	Change to subdirectory *foo* (go down in tree)
`cd ..`	Go up one directory in tree
`rmdir` *foo*	Remove subdirectory *foo* (must be empty)
`xemacs` *foo* &	Edit file *foo* with XEmacs (& to run in background)
`more` *foo*	Display file *foo* (space for next page)
`rm` *foo*	Delete file *foo*
`cp` *foo bar*	Copy file *foo* to file *bar*, e.g. `cp` ˜*smith/foo* `./` copies Smith's file *foo* to my current directory
`mv` *foo bar*	Rename file *foo* to *bar*
`lpr` *foo*	Print file *foo*. Use `-P` to specify print queue, e.g. `lpr -P`*laser1 foo* (site dependent)
`ps`	Show existing processes
`kill 345`	Kill process number 345 (`kill -9` as last resort)
`./`*foo*	Run the executable program *foo* in current directory
ctrl-`c`	Terminate currently executing program

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

Networks

- Accounts usually at home lab and also SLAC, CERN, RAL, . . .

`ftp flora.slac.stanford.edu`	file transfer
`telnet hpplus.cern.ch`	remote login
`ssh csfsun.rl.ac.uk`	'secure shell' login

- Many disks accesible world-wide via AFS (Andrew File System). Typical (long) AFS file name:

$$\texttt{/afs/}\underbrace{\texttt{cern.ch}}\texttt{/user/c/cowan/public/conference\_plot.ps}$$

↖ cell name

- Files (can be) visible world-wide; access determined by

  Access Control Lists (ACL), and

  whether you have a 'token' for the cell.

- You get a token (expires after 25 hours) by

  `klog -cell` *cell name*

- Once you have a token, you can e.g. run editor locally and edit a file at a remote site (usually much faster).

- Useful commands:

`kpasswd`	Set AFS password
`tokens`	Show currently held tokens
`fs listquota`	Check AFS disk quota

N.B. Some sites use ARLA: same system, different command names.

X

- Windowing system, consisting of

X-server:	process running on local machine (workstation or X-terminal); talks to monitor, keyboard, mouse, ...
X-clients:	programs talking to X-server to request e.g. drawing windows, lines, etc. Can run on remote machines.

- Some important clients:

`xterm`	terminal emulator
`xman`	X-windows version of Unix manual
`xemacs`	X-windows version of emacs text editor
`paw`	CERN data analysis and display program

- Best with AFS or ssh. But if you need telnet e.g. to run on hpplus, specify hpplus as an allowed host to your local machine:

```
my_machine> xhost +hpplus.cern.ch
```

Log in to the remote machine:

```
my_machine> telnet hpplus.cern.ch
```

Set environment variable `DISPLAY` to tell X-client who to talk to; use internet address of your local machine plus `:0`

```
hpplus> setenv DISPLAY my_machine.ac.uk:0
```

Run X-client on hpplus; window appears on your local monitor:

```
hpplus> xterm &
```

Physics Analysis Workstation (PAW)

- PAW is an interactive program for plotting and manipulating

  vectors
  histograms
  n-tuples

- To run, type `paw`. Enter workstation type, default probably OK.

- Many commands (hierarchical structure), e.g.

  `help` (type Q to return to command mode)
  `help plot` (help on command 'plot')

- Customize set-up, define new commands, etc. with file

  `.pawlogon.kumac`

  in home directory (Unix systems).

- Online tutorial and manual via PAW home page:

  `http://wwwinfo.cern.ch/asd/paw/index.html`

- 'Advanced' features:

  mathematical functions and operations (SIGMA)
  FORTRAN interpreter (COMIS)
  fitting with MINUIT

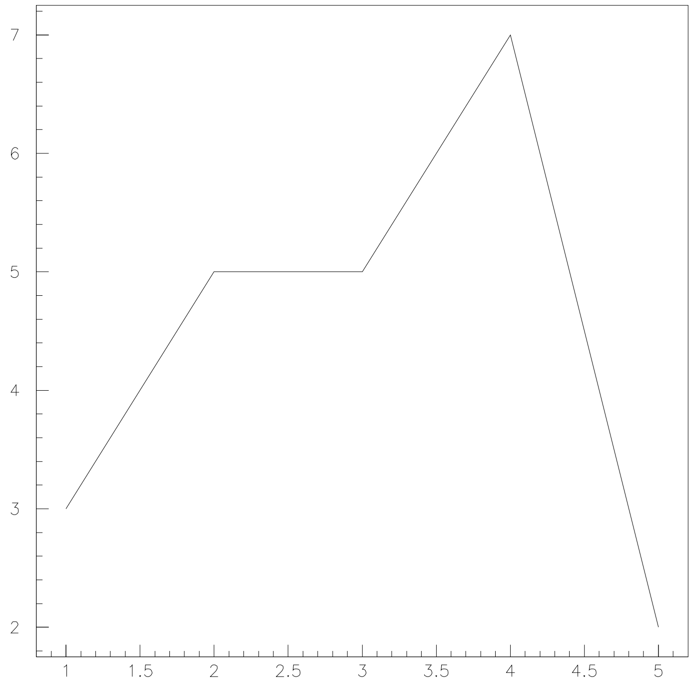
Using PAW

- Enter commands interactively or with command file (`*.kumac`)

  Example – file `test.kumac` contains

```
vec/create x(5) R 1 2 3 4 5
vec/create y(5) R 3 5 5 7 2
graph 5 x y
```

  Typing `exec test` produces



- To get an encapsulated PostScript file for printing, enter

  `pict/print` *filename*`.ps`

  (Needs `graphics/opt zfl1` e.g. in `.pawlogon.kumac`.)

Programming

- Many books on FORTRAN, C++ (see course web site).

- FORTRAN ('structured'), e.g. file `hello.f`,

```
program HELLO
write (*, *) 'hello world'
END
```

- To compile, link and run,

```
f77 -o hello hello.f
```

rename the executable program

```
./hello              ← type this to run program
hello world          ← output appears on screen
```

- List extra files separated by spaces; new line with backslash,

```
f77 -o greetings greetings.f bonjour.f \
hola.f gruezi.f yo_dude.f
```

- List CERN libraries at end in backwards single quotes, e.g.

```
`cernlib`
`cernlib graflib mathlib kernlib packlib`
```

- C++ ('object oriented'). `g++` runs the GNU C++ compiler gcc,

```
g++ -o hello hello.cc
```

A FORTRAN example

```
      program MAKE_DATA

c   Author:  Glen Cowan
c   Date:    21 August, 1999
c   Test program to make a simple data set and write it to a file.

      implicit        NONE

c   Constants

      integer         num_points
      parameter       (num_points = 11)

c   Local variables

      character*80    outfile

      integer         i

      real            x
      real            x_max
      real            x_min
      real            y

c   Initialize some variables and open output file

      x_min = 0.
      x_max = 2.
      outfile = 'test_data.dat'
      open (unit = 20, file = outfile, form = 'formatted',
     & status = 'unknown', carriagecontrol = 'list')

c   Make the data and write to file

      do i = 1, num_points
        x = (x_max-x_min)*FLOAT(i-1)/FLOAT(num_points-1)  +  x_min
        y = x**3  -  2.*x**2  +  x
        write (20, *) x, y
      end do

      close (20)

      stop
      END
```

A C++ example

```
//   Test program make_data
//   Glen Cowan
//   Royal Holloway, University of London
//   28 September, 2001
//   Test program to make a simple data set and write it to a file.

#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <math.h>

void main(){

  int num_points = 11;
  double x, y;
  double x_min = 0;
  double x_max = 2;

  ofstream outfile("test_data.dat");
  outfile << setw(10);
  outfile << setiosflags (ios::fixed | ios::right) << setprecision(5);

  for (int i=0; i<num_points; i++){
    x = (x_max-x_min)*double(i)/double(num_points-1)  +  x_min;
    y = pow(x,3) - 2.*pow(x,2) + x;
    outfile << x << "    " << y << endl;
  }

}
```

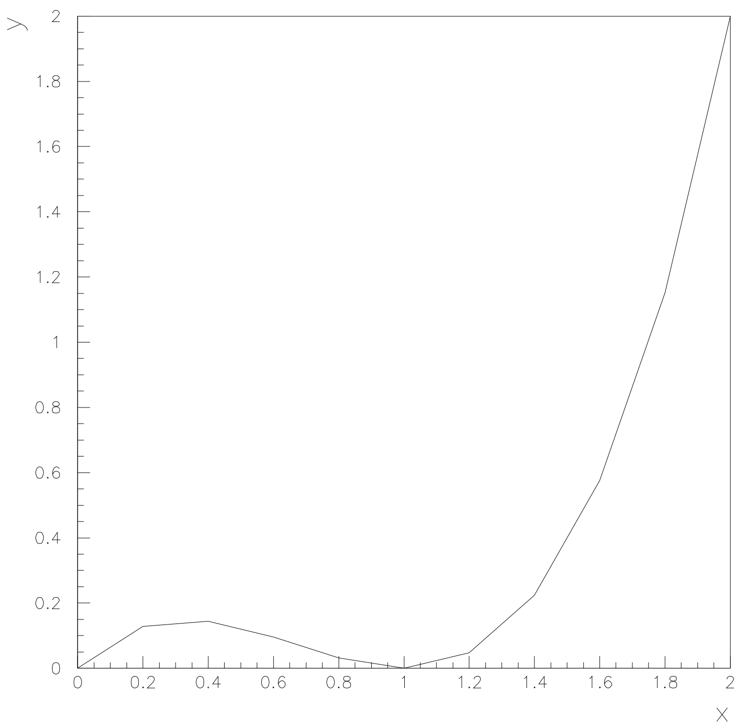
Looking at the output

- Running the program produces the file `test_data.dat`:

$0.0000000E+00$	$0.0000000E+00$
0.2000000	0.1280000
0.4000000	0.1440000
0.6000000	$9.6000016E-02$
0.8000000	$3.1999946E-02$
1.000000	$0.0000000E+00$
1.200000	$4.8000097E-02$
1.400000	0.2240001
1.600000	0.5759999
1.800000	1.152000
2.000000	2.000000

- With PAW, read in the file, create vectors and plot . . .

```
vec/read x,y test_data.dat   | create vectors x,y
null 0. 2. 0. 2.             | sets min/max values
graph $VLEN(x) x y 'L'       | $VLEN(x) = length of x
atit  'x' 'y'                | axis titles
```



Compiling and linking with `gmake`

- Often a program is in many files, e.g. `hello.cc` contains

```
#include <iostream.h>
#include "goodbye.h"     // contains prototype of function goodbye

void main(){
cout << "Hello world" << endl;
goodbye();
}
```

  Function `goodbye` is in `goodbye.cc`, prototype in `goodbye.h`.

- We could compile and link with

```
g++ -o hello hello.cc goodbye.cc
```

  which is really a short-cut for

```
g++ -c hello.cc        ← -c to compile (produces hello.o)
g++ -c goodbye.cc
g++ -o hello hello.o goodbye.o    ← link object files
```

- If e.g. `goodbye.cc` changed, we don't need to recompile `hello.cc`, but `g++ -o hello hello.cc goodbye.cc` does the whole lot; in large programs it's difficult to know what to recompile.

- With the Unix program `make` (GNU version `gmake`):

  user supplies a *makefile* (called `GNUmakefile`, `makefile`, or `Makefile`), specifying how files depend on each other.

- Type `gmake` (plus optional argument); `gmake` looks at file dates and figures out what to do.

A simple makefile

- Makefiles have several types of statements, most importantly *rules*:

    target : *dependencies* ...

    command ← commands preceded by tab character

    ⋮ ← new line (and tab) for each command

- A possible makefile for the previous example:

```
hello : hello.o goodbye.o
        g++ -o hello hello.o goodbye.o
hello.o : hello.cc goodbye.h
        g++ -c hello.cc
goodbye.o : goodbye.cc
        g++ -c goodbye.cc
```

- Type `gmake` *target* (if target name omitted, first one used), e.g.

    `gmake` ← makes executable program `hello`

    `gmake goodbye.o` ← compiles `goodbye.cc` only

- In practice, more complicated (see note on web):

```
# A simple makefile

objects = hello.o goodbye.o

hello : $(objects)
        g++ -o hello $(objects)
hello.o : hello.cc goodbye.h
goodbye.o : goodbye.cc

.PHONY : clean
clean :
        -rm hello $(objects)
```

Add comments, define variables, use implicit commands and dependencies, supply phony targets (e.g. **`clean`**), ...

Preparing documents with LaTeX

- Many online information sources and books; see e.g.

  Kopka and Daly, *A Guide to LaTeX2e*, Addison-Wesley, 1995

- An almost minimal LaTeX source file (more samples on web):

```
%\documentstyle[12pt,epsfig]{article}   % use with version 2.09
\documentclass[a4paper,12pt]{article}   % use with version 2e
\usepackage{epsfig}                     % use with version 2e
\begin{document}

The text of your document goes here.  \LaTeX is very good
at writing formulae as in equation (\ref{exp_dist}),

\begin{equation}
\label{exp_dist}
F(t) = \frac{1}{\tau} \int_{0}^{t} e^{-t'/\tau} \, dt' \, .
\end{equation}

\end{document}
```

- To see output, type (default file extensions may be omitted)

  `latex` *file*`.tex` ← produces *file*`.dvi`

  `dvips` *file*`.dvi` ← produces *file*`.ps`, print with `lpr`

  `gv` *file*`.ps` & ← GhostView (& to run in background)

- Also useful ...

  `ps2pdf` *file*`.ps` ← creates *file*`.pdf`

  `convert` *file*`.gif` *file*`.ps` (many formats possible)

- See example files on web to see how to

  include figures, create equations, lists, tables;

  include sections, subsections, bibliography ...

More stuff

- Things we'll see later in the course:

  Histogramming, n-tuples (HBOOK)

  Structure of analysis programs in HEP

  Monte Carlo (random numbers)

  Function minimization (MINUIT)

- Things you'll have to pick up on your own:

  A text editor: emacs, xemacs, vi, xedit, ...

  E-mail: pine, outlook, Netscape

  The web: HTML

  Other operating systems: Windows NT, ...

  and their utility programs: PowerPoint, Excel, Word, ...

  Debugging programs: ddd, dbx, ...

  Command languages: shell scripts, tcl, perl

  Source code management: CVS, SRT, ...

  Databases: Oracle, Objectivity, ...

  Batch queues: LSF (CERN, SLAC, ...)

  NQS (RAL, RH, UCL, ...)

  ⋮