

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

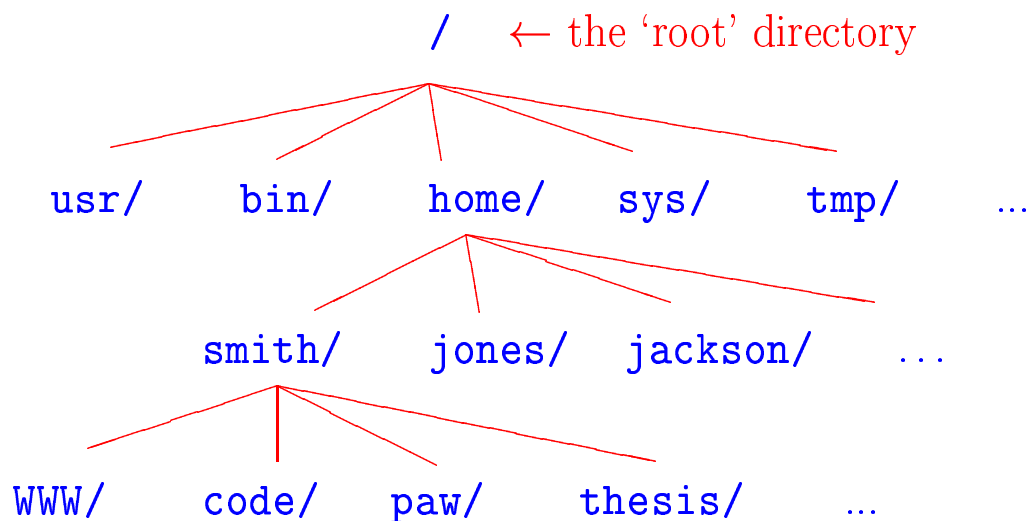
⋮

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`, `csch`, `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!)

<code>pwd</code>	Show present working directory
<code>ls</code>	List files in present working directory
<code>ls -l</code>	List files of present working directory with details
<code>man ls</code>	Show manual page for <code>ls</code> . Works for all commands.
<code>cd</code>	Change present working directory to home directory
<code>mkdir foo</code>	Create subdirectory <i>foo</i>
<code>cd foo</code>	Change to subdirectory <i>foo</i> (go down in tree)
<code>cd ..</code>	Go up one directory in tree
<code>rmdir foo</code>	Remove subdirectory <i>foo</i> (must be empty)
<code>xemacs foo &</code>	Edit file <i>foo</i> with XEmacs (& to run in background)
<code>more foo</code>	Display file <i>foo</i> (space for next page)
<code>rm foo</code>	Delete file <i>foo</i>
<code>cp foo bar</code>	Copy file <i>foo</i> to file <i>bar</i> , e.g. <code>cp ~smith/foo ./</code> copies Smith's file <i>foo</i> to my current directory
<code>mv foo bar</code>	Rename file <i>foo</i> to <i>bar</i>
<code>lpr foo</code>	Print file <i>foo</i> . Use <code>-P</code> to specify print queue, e.g. <code>lpr -Plaser1 foo</code> (site dependent)
<code>ps</code>	Show existing processes
<code>kill 345</code>	Kill process number 345 (<code>kill -9</code> as last resort)
<code>./foo</code>	Run the executable program <i>foo</i> in current directory
<code>ctrl-c</code>	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, ...

<code>ftp shire.slac.stanford.edu</code>	file transfer
<code>telnet hpplus.cern.ch</code>	remote login
<code>ssh csfsun.rl.ac.uk</code>	'secure shell' login

- Many disks accessible world-wide via AFS (Andrew File System).

Typical (long) AFS file name:

`/afs/cern.ch/user/c/cowan/public/conference_plot.ps`


- 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:

<code>kpasswd</code>	Set AFS password
<code>tokens</code>	Show currently held tokens
<code>fs listquota</code>	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

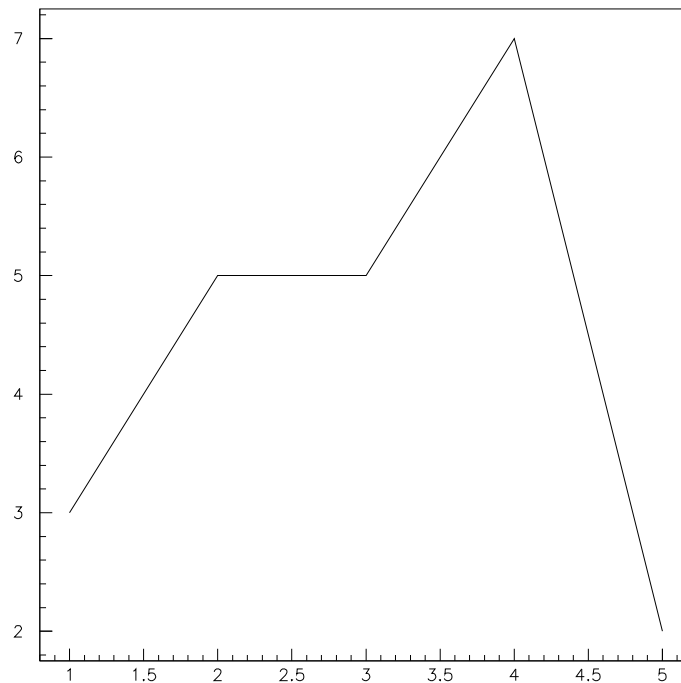
Using PAW

- 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.eps
```

(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
```

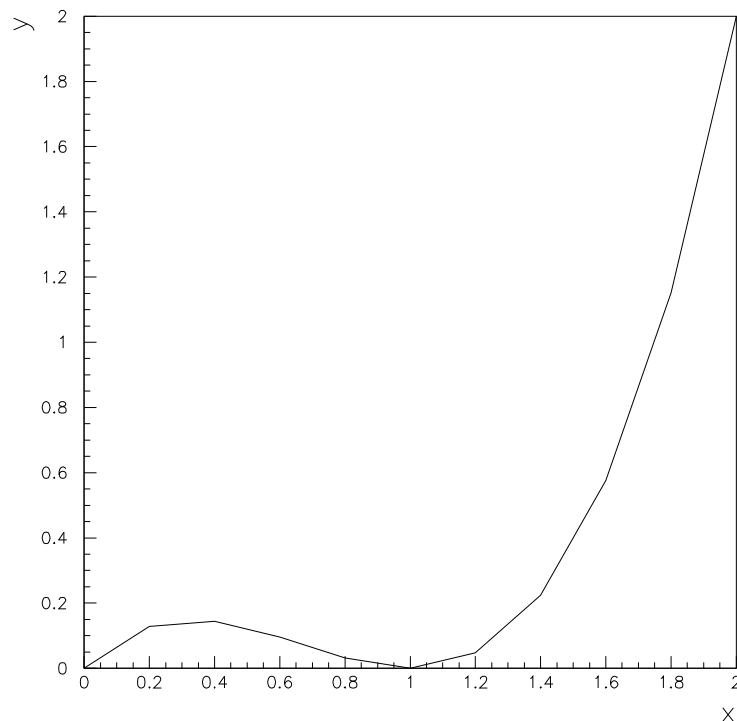
Looking at the output

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

```
0.000000E + 00  0.000000E + 00
0.200000      0.128000
0.400000      0.144000
0.600000      9.600016E - 02
0.800000      3.199946E - 02
1.000000      0.000000E + 00
1.200000      4.800097E - 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`), ...

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

Kopka and Daly, *A Guide to L^AT_EX 2_ε*, Addison-Wesley, 1995

- An almost minimal L^AT_EX source file (more samples on web):

```
%\documentstyle[12pt,epsfig]{article}    % use with version 2.09
\documentclass[a4paper,12pt]{article}    % use with version 2ε
\usepackage{epsfig}                    % use with version 2ε
\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, ...)

⋮