HEP Computing
Part I
Intro to UNIX/LINUX
Marcella Bona

Lectures 1,2,3

© Adrian Bevan
Lecture 1

• Files and directories.

• Introduce a number of simple UNIX commands for manipulation of files and directories.

• communicating with remote machines
What is LINUX

- LINUX is the operating system (OS) kernel.
- Sitting on top of the LINUX OS are a lot of utilities that help you do stuff.
- You get a ‘LINUX distribution’ installed on your desktop/laptop. This is a sloppy way of saying you get the OS bundled with lots of useful utilities/applications.
- Use LINUX to mean anything from the OS to the distribution we are using.
- UNIX is an operating system that is very similar to LINUX (same command names, sometimes slightly different functionalities of commands etc).
  - There are usually enough subtle differences between LINUX and UNIX versions to keep you on your toes (e.g. Solaris and LINUX) when running applications on multiple platforms ...be mindful of this if you use other UNIX flavours.
  - Mac OS X is based on a UNIX distribution.
Accessing a machine

• You need a user account
  → you should all have one by now

• can then log in at the terminal
  (i.e. sit in front of a machine and type in your user name and password to log in to it).

• you can also log in remotely to a machine somewhere else
The command line

- A user interfaces with Linux by typing commands into a shell.
  - if you are familiar with Windows then think of a shell being something like the DOS prompt.
- The shell is a program that knows how to find commands to run and how to run them.
Some useful commands

(listed and more get introduced by example in the following pages)

ls <dir>
list the content of a directory
cd <dir>
change directory to the specified one
mkdir <–p> <dir>
make a new directory (–p makes missing parents)
cp <file> <newfile>
make a copy of a file
cmp <file name> <new file name>
rename a file
tail <file>
look at the end of a file
head <file>
look at the start of a file
cat <file>
show the file from start to finish on the terminal
more <file>
file viewer
less <file>
file viewer (more versatile than more)
sleep <nSeconds>
sleep for a number of seconds before continuing
gzip <file>
zip a file up
tar cvf somefile.tar <directory>
make a tar file (archive files or directory trees)
tar xvf somefile.tar
unpack a tar file
tar cvzf somefile.tgz <directory>
make a tar file and zip it up in one go
tar xzvf somefile.tgz
unpack a zipped tar file
Files and directories

• Files and directories are stored on a file system.
• the root file system starts at /
• sitting in this are many different directories
e.g. you can see what is there using ls

somehost ~ > ls /
afs/    bin/    data/   home/    lost+found/  opt/  root/  u01@  u04@  usr/
afscache/ boot/  dev/  initrd/  misc/      pippo/ sbin/  u02@  u05@  var/
bfactory@  cern@   etc/    lib/    mnt/      proc/   tmp/   u03@  u06@

• Directories are organised in a hierarchical structure
  → a / separates one directory level from the next [c.f. \ in Windows]
• e.g. /usr/bin is the subdirectory bin that is
  in the sub directory usr found in /
• You have a home directory that can be access via ~username
e.g. ~smith or ~/ for short
Manipulating files and directories

- make a new directory with the `mkdir` command
  ```bash
  mkdir test
  ```
- list the files in a directory using the `ls` command
  ```bash
  ls test
  ```
- change into the directory you just made using the `cd` command
  ```bash
  cd test
  ```
- can now make another directory
  ```bash
  mkdir test2
  ```
- now if you list the files in the directory `test` (type `ls`) you’ll see `test2`
- which you just made. You can then rename the new directory with
  ```bash
  mv test2 test3
  ```
  and using `ls` again you can see that the new directory is now called `test3`
mkdir test
ls test
cd test
mkdir test2
mv test2 test3

(sub)directories:

```bash
./
./
./test
./
./test
./test/test2
./
./test
./test/test3
```

(directory renamed)
Manipulating directories and files (II)

• What if you get lost and want to go home?

  cd
  cd ~/
  cd $HOME

  All are equivalent and will take you to your home directory (the one you appear at when you log in)

• How can you tell which directory you are currently in?

  pwd

• Is there an easy way to go to the previous directory?

  cd –

• How do I get to one directory down the tree?

  cd ../
Looking at the content of a file

- You already have a file called ~/.bash_profile which is one of your login scripts. How can you look at this file? If you just ls in your home directory it is not there ...
  
  ```bash
  ls -a
  ```

  will show the hidden files (names starting with a ‘.’)

- cd to your home directory [if you are in a subdir using cd will take you home]
- try to use the commands cat, tail, head, more and less to look at the file .bash_profile

  ```bash
  cat .bash_profile  # print the file to the screen
  head .bash_profile # print the first 10 lines
  head -20 .bash_profile # print first 20 lines
  tail .bash_profile  # print last 10 lines
  tail -30 .bash_profile # print last 30 lines
  more .bash_profile # use more to look at the file
  less .bash_profile # use less to look at the file
  ```
Looking at the content of a file (II)

cat .bash_profile
head .bash_profile
head -20 .bash_profile
tail .bash_profile
tail -30 .bash_profile
more .bash_profile
less .bash_profile

# print the file to the screen
# print the first 10 lines
# print first 20 lines
# print last 10 lines
# print last 30 lines
# use more to look at the file
# use less to look at the file

N.B. use ‘q’ in more or less to quit and return to the command prompt
→ in less you can use the up and down arrows
to move through the file
→ / followed by a string entered into less
will search for that string
→ ? followed by a string entered into less will search backwards for that string
1. try using the commands described on the last few pages to get familiar with them

2. play with the sleep command type
   
   > sleep 5

   then

   > sleep 10; mkdir iJustWokeUp

   (include the semi-colon, ‘;’ after sleep 10). You see that you can use sleep to delay execution of a command. ‘;’ is a command separator for the shell

• copy .bash_profile to the directory test that you made earlier

   > cp .bash_profile test/
   > ls -a test/  # use -a to see hidden files
   # (these start with a .)

   bfa ~/ > ls -a test
   ./  ../  .bash_profile

   current directory

   directory one up from the subdirectory test

   copy of your .tcshrc
Communicating between different machines

Some group machines may use telnet and ftp for communication (login and copy).

```plaintext
> telnet somemachine.somedomain
> telnet MyComputer.MyUni.ac.uk
> ftp somemachine.somedomain
> ftp MyComputer.MyUni.ac.uk
```

Almost all machines you will encounter will not use these programs. Instead you need to use ssh/scp to login or copy a file to a new machine.

```plaintext
> ssh <options> somemachine.somedomain
> ssh MyComputer.MyUni.ac.uk
> scp <options> <source> <target>
> scp test.eps MyComputer.MyUni.ac.uk:/./myEpsFiles/
```

where `<options>` are command line options that you can type in if you want to. [N.B. the angled brackets are not to be input but indicate that this is an option]

ftp is generally discouraged. It’s a good idea not to use it!
Logon and copy examples:

Example of using ssh to log into a machine

> ssh -l username hostname.MyUni.ac.uk
> ssh username@hostname.MyUni.ac.uk
> ssh hostname.MyUni.ac.uk

Example of using scp

> scp test.ps username@hostname:./public_html/

Equivalent forms of using the command. N.B. if you don’t specify the username it will be assumed that you want to use your for the connection.

> scp -r test-dir username@hostname:./public_html/

So why do/should you care?

→ most ... if not all ... of your work will be done at remote machines
Lecture 2

• Text Editing
• sed and awk
• Environment variables and aliases
• Archiving files
Text editing

As soon as you want to start to do analysis/write reports etc … you need to edit files. There is nothing like word available for editing code so you have to learn how to use a text editor.

Some UNIX Text Editors:

(x)emacs        nice gui/text based editor – most people use this
vi             very useful for sysadmin/minimal system work
pico, vim, …

EMACS:
to start emacs:

> emacs <somefile> &  to start a gui emacs session
> emacs –nw <somefile>  to start a text session

Useful resources can be found:

GNU’s online manual


man pages give a summary of information
emacs help → enter this by opening emacs and pressing F1 twice
Some emacs

Aside: On mac OS you need to replace Alt with Esc

Some of the emacs commands you should know:

- `[ctrl-x]+f` open a file
- `[ctrl-x]+i` insert a file
- `[ctrl-x]+s` save a file
- `[ctrl-x]+[ctrl-c]` close emacs
- `[alt-x]+ispell-buffer` run ispell from emacs
- `[alt-x]+spell-region` run ispell from emacs
- `[alt-x]+goto-line` go to line #
- `[ctrl-x]+(` start defn. of a macro
- `[ctrl-x]+)` close dfn of a macro
- `[ctrl-x]+e` execute a macro
- `[ctrl-x]+ u #` repeat next command # times
- `[alt-x]+query-replace` replace string with another one
Some more emacs

- [ctrl-space] mark the start of a region
- [ctrl-x]+r+k mark the end of a region and kill it
- [ctrl-x]+r+y paste a region

There are MANY more commands available; these are just the ones that I use most often.
Emacs examples

• Open an emacs session and start typing into the file;
  ```
  emacs -nw test.txt
  ```

• When you have some text, save the file
  ```
  [ctrl-x]+s
  ```

• then close the file using
  ```
  [ctrl-x]+[ctrl-c]
  ```

• you should now have a file test.txt – you can see it is there by using the `ls` command
  ```
  ls -l test.txt
  ```

• now you can try using `less` or `more` to view the file:
  ```
  less test.txt
  ```
if you now open a new file called test2.txt, you can insert the original file using

\texttt{[ctrl-x]+i followed by test.txt}

if you want to play about with the file some more you can do so. Then save test2.txt using

\texttt{[ctrl-x]+s}.

Try using the \texttt{[alt-x] query-replace} command to change all of the letters ‘a’ for ‘X’ in what you have written

you are prompted for the strings to match and can either approve the change on a one by one basis or do all at once using !

• you should play around for a while with the various commands listed on the previous page to get used to things a bit.
The file is empty until you save it

Enter a few lines of text into the terminal

[ctrl-x]+s
Make a new file called `somefile.txt`, and edit your original

```bash
emacs -nw somefile.txt
```

Move the cursor to the end of `somefile.txt` and try to add the content of another file to this one:

- `[ctrl-x]+i` followed by `test.txt`
- `[ctrl-x]+s`

The file is not updated until you save it.
sed and awk

These are command line text editing utilities to help process information involving replacement of strings/extracting columns of text from files etc.

Some useful examples:

- `sed -e 's/A/B/' <filename>`
  - substitute A for B in 1st instance on line in the whole file

- `sed -e 's/A/B/g' <filename>`
  - substitute A for B in whole file

- `awk '{print $1}' <filename>`
  - print the first column of file [space separator used by default]

- `awk -F=' ' '{print $1}' <filename>`
  - use the ‘=’ character as a separator

- `awk '{s+=$1}END{print “sum = “ s}' <filename>`
  - add up the numbers in the first column

→ try to use sed and awk on test.txt and test2.txt.

Look at the GNU manuals for gawk & search on google for awk/sed

O'Reilly “sed & awk” is a good book to read
Some simple examples

`sed 's/the/THE/g' test.txt`  replace ‘the’ with ‘THE’

`sed 's/the/THE/' test.txt`  replace the first ‘the’ per line with ‘THE’

`awk '{print $1}' test.txt`  print the first word on each line

`echo "hello world" | sed 's/world/universe/'`  substitute `world` for `universe` in print out

`sed` and `awk` can do a lot more than shown here ... see extra material for a few ideas
Some more UNIX command line examples

grep <string> test.txt > myStringFile
    search for <string> in the file and
    redirect the output to a file called myStringFile

./myBin >& myBin.log &
tail –f myBin.log
    run the binary – writing to a log file (passing
    stdout and stderr to that log file) and then follow
    the tail of the log file as it gets written out

cat file2 >> file1
    append the content of file2 at the end of file1

export MYVAR=val; ./myBin >& mylog & tail –f mylog
    do several things on one line

ls /usr/local/bin/?v*
    e.g. pattern a search for binaries with
    a single character followed
    by a ‘v’ and finishing with anything else.
Some more commands

check how much free space you have left on an afs file system

rm <aFile>

to remove the file aFile. Deleting a file on LINUX/UNIX means that it has gone!

now you’ve seen the rm command, you know how to delete things
→ you might want to use the following until you get more confident
   rm -i <aFile>

• Note that the work areas you have and use are generally not backed up.

• if you have something that is important (e.g. thesis) you should ask how to make back up copies on a regular basis (if you don’t know how to do it) so that you don’t loose everything if you accidentally delete the original or a hardware failure looses the disk etc.

• At outside labs there is usually a copy of your home area backed up once a day: check if this exists and if the backup it suits your needs.
Environment Variables

There are many so-called environment variables set in a shell. To list these types:

```bash
printenv
tcsh/sh
env
tcsh/sh
```

The most important one is:

**PATH**

set the search path for commands so the system knows where to look for them. This is the search path that a shell uses to find a command. If a command is not in one of the directories listed in $PATH then you have to refer to it using the full path name e.g. `/usr/bin/perl` instead of `perl`.

How do you know if you can see a command? Use `which` to search for it. If the shell can find a command by looking through the locations in $PATH it returns the one it finds first. e.g.

```
somehost ~ > which perl
/usr/bin/perl
```
Using environment variables

- **Export a variable**: `export myVar=value`
  - variable name: `myVar`
  - value: `value`

- **Echo a variable**: `echo $PATH`
  - Command to print something to the screen
  - The value of the `PATH` environment variable is accessed by prefixing `PATH` with `$`

- **Unset a variable**: `unset myVar`
  - To delete/unset an environment variable
Sometimes you want to append to an existing variable, e.g. PATH. You can do this like

```
export PATH=<new path to add>:$PATH
```

• **command not found** means just that – the system can not find the command when searching the directories listed in $PATH

These examples are for sh. For the tcsh they differ slightly and I’ll leave it to you to find out how to manipulate the environment variables there.

A few other useful variables are:

- **EDITOR** set the default text editor
- **PRINTER** the default printer (what lpr will try to print to unless you tell it otherwise)
- **PAGER** e.g. set `more` or `less` as the default pager

E.g.

```
export EDITOR=emacs
export PAGER=less
export PRINTER=oa2
```
An alias is a command short cut that you can specify. These usually go into your login scripts such as .cshrc/.tcshrc etc e.g.

```bash
alias rm='rm -i'
alias ls='ls -F'
```

You can see the list of aliases by typing `alias` at the command line:

```bash
alias clean='rm *~'
alias l.='ls -d .* --color=ttty'
alias ll='ls -l --color=ttty'
alias ls='ls -l'
...
```

This can be useful so that you don’t have to type in the full command to ssh to an outside lab all the time, or to customize your setup as you like.
Examples

• List the environment variables already set for you.

• Check the variables EDITOR, PAGER and PRINTER. Are these defaults ok? if not change them: e.g. I have

  [somehost] ~ > env | grep EDITOR
  EDITOR=vi
  [somehost] ~ > env | grep PAGER
  PAGER=less
  [somehost] ~ > env | grep PRINTER
  PRINTER=oa1

  you probably want emacs
  Ask a local what the names of your printers are

• Could have just echo(ed) the variables instead.

• Make a directory called scripts and add this to your PATH. You’ll use this later on.
Compressing files and directories

• You can compress files to save space. The `gzip/gunzip` commands are used to zip/unzip files. Large savings in space can be had in compressing ascii files ... on the other hand sometimes binary files are packed so efficiently that you don’t get any gain from using these utilities to compress those files.

  • use `ls -l` to see how big a file is
    (`ls -lh` shows the file size in Kb/Mb/Gb)
  • e.g. compressing a single file
    `gzip thesis.ps`
    this command will write a compressed file called `thesis.ps.gz`
    that should be a lot smaller than the original `thesis.ps`
  • you can then unzip the file by using
    `gunzip thesis.ps.gz`
  • This can be quite handy if you have very limited disk space
    (e.g. at SLAC/CERN etc)
    • what about dealing with the content of a directory tree?
Archiving files

• If you have a lot of files that you want to store you can make an archive before moving/compressing them. The most common utility you will see for this is called `tar`. The name comes from the historical origin as a Tape ARchival program.

• Other programs exist such as `dd` & `cpio` etc.

• to make an archive of the directory test
  
  ```
  cd
  # return to your home directory
  tar cvf test.tar test/
  # make the archive file
  ```

• you should now have a file called `test.tar` in your home directory that can be compressed using `gzip`. To unpack the archive in a new directory
  
  ```
  mkdir new-test
  cd new-test
  cp ~/test.tar .
  tar xvf test.tar
  # unpack the archive in ~/new-test
  ```

  so `new-test` now contains a complete copy of the directory tree `test`
What did the options given to tar mean???

- **c** create an archive file
- **v** verbose (print out files added/extracted from the “tarball”)
- **f** file – the f option should be followed by the name of the file to create
- **x** extract files from the archive.

```
tar cvf test.tar test/
```

archive file to create
directory to archive

if you use the option ‘z’ when using tar, you can compress the archive at the same time as making it. e.g.
```
tar zcvf test.tgz test/
tar zxvf test.tgz
```

• So How are you supposed to know all of this for the different commands?
system commands usually come with man pages
  → these are instructions on how to use the command.
e.g. type the following to look at the ls man page
  > man ls

NAME
   ls - list directory contents

SYNOPSIS
   ls [OPTION]... [FILE]...

DESCRIPTION
   List information about the FILEs (the current directory by default). Sort entries alphabetically if none of -cftuSUX nor --sort.

   -a, --all
      do not hide entries starting with .

   -A, --almost-all
      do not list implied . and ..

   -b, --escape
      print octal escapes for nongraphic characters

If you are not sure of a command name you can always try using man –k followed by a search string to see if any matches might exist;
e.g. type man –k copy
Finding files

Sometimes you will lose track of where a particular file is. Either you are looking for something you have written, or something that you’ve been asked to find. There are two useful utils for tracking down files:

- **find**
  - set the path to start searching
  - `find . -name core`
    - name the file/string you are looking for

- **locate**
  - `locate core`
    - locate uses a database of files that is automatically updated on a LINUX machine so it is usually a lot quicker than using find to locate a file (unless you have made the file AFTER the db was last updated).
1. Try following the examples on using gzip and tar to compress and archive the dummy directory

2. If you’ve not already done so, download the example tgz files and unpack these in your home directory. Look at the content of ~/Lectures for each new examples file you add.

3. Run the command
   
   du –sh ~/Lectures

   to see how much disk is used in total and compare this with the sum of file sizes for partN_examples.tgz that you’ve unpacked using ls -lh

   [N.B.] most of the space is taken up by root files that are already compressed so the reduction in size is not great for part3_examples.tgz.

4. Look at the man pages for some of the commands you now know.
   HINT: the commands you know now do a lot more than you have learnt about so far. Details on the man page tell you what else they can do … sometimes the description isn’t that easy to follow!

5. If you’re happy with this – look at the extra material
Lecture 3

- Review the relationships between the most important concepts and commands covered in the lectures 1 and 2.

- Introduce web resources available.
It's a good idea that you make yourself familiar with the behavior of these Commands and know how to use them if you’ve not done so yet.

Most of what you will do with computers is, in essence, manipulating information in files. These commands cover a range of ways to look at the content of files and edit them.
Most of what you will do with computers is, in essence, manipulating information in files. These commands cover a range of ways to look at the content of files and edit them.
Most of what you will do with computers is, in essence, manipulating information in files. These commands cover a range of ways to look at the content of files and edit them.
Most of what you will do with computers is, in essence, manipulating information in files. These commands cover a range of ways to look at the content of files and edit them.
Most of what you will do with computers is, in essence, manipulating information in files. These commands cover a range of ways to look at the content of files and edit them.
Most of what you will do with computers is, in essence, manipulating information in files. These commands cover a range of ways to look at the content of files and edit them.
Most of what you will do with computers is, in essence, manipulating information in files. These commands cover a range of ways to look at the content of files and edit them.
Very Quickly you will see that a structured system for storing files is needed → you need sub-directories. These commands are useful to manipulate your directories.
Very Quickly you will see that a structured system for storing files is needed → you need sub-directories. These commands are useful to manipulate your directories.
Very Quickly you will see that a structured system for storing files is needed → you need sub-directories. These commands are useful to manipulate your directories.
Very Quickly you will see that a structured system for storing files is needed → you need sub-directories. These commands are useful to manipulate your directories.
Very Quickly you will see that a structured system for storing files is needed → you need sub-directories. These commands are useful to manipulate your directories.
Very Quickly you will see that a structured system for storing files is needed → you need sub-directories. These commands are useful to manipulate your directories.
When manipulating files there will come a time when you want to look for a string or pattern, and either extract or modify it. Basic use of these commands can save enormous amounts of time with repetitive tasks.
When manipulating files there will come a time when you want to look for a string or pattern, and either extract or modify it. Basic use of these commands can save enormous amounts of time with repetitive tasks.
When manipulating files there will come a time when you want to look for a string or pattern, and either extract or modify it. Basic use of these commands can save enormous amounts of time with repetitive tasks.
When manipulating files there will come a time when you want to look for a string or pattern, and either extract or modify it. Basic use of these commands can save enormous amounts of time with repetitive tasks.
Available Resources

• If you are stuck with a problem there are several resources available to you:
  – collaborators in your group or on your experiment.
  – books: library, colleague's bookshelf etc.
  – web resources:
    • www.google.co.uk is surprisingly good in helping you find useful technical websites.
    • An alternative is cetus-links (URL on next page)
Remember the link
Before moving on from this introduction, here is a biased opinion of what is worth knowing:

- **C, C++, some FORTRAN**
- **perl, tcsh and bash**
- **a debugger**

- coding your analysis
- automation when running analysis, job submission and automation of tedious repetitive tasks
- finding bugs

Along the way you may also learn a bit about design patterns and Object Orientated programming.

If you do, don’t worry about good design to begin with just learn the syntax. When you know this then you’ll start to pick up what is good and what is bad. Writing ‘good code’ takes a long time … so be patient and don’t worry too much about making mistakes when you are learning.
You’ll find some material on the following topics in the extra material

- more complicated uses of *sed*

- Some jargon you’ll encounter

- Writing your own *Makefile*

- Simple use of a debugger to identify null pointers