Computer commands that come in handy

Robert Simpson

January 15, 2012

Overview

This document is merely a collection of the Linux commands that I have used in the past and I have found it useful to write them down for future reference. Each command has all sorts of flags and extra parameters associated with it, but I have not included an exhaustive list here since they can be found readily on the internet. Instead, I just explain in simple terms what the command is and then give an example use.

Linux

---

cd

Change to another directory. e.g. To change to a directory that is located from the root directory as documents/myfolder

```
cd ~/documents/myfolder
```

---

ls

List the documents in the current directory. Quite simply:

```
ls
```

Or, to find out detailed information about the files including chmod permissions:

```
ls -l
```

Or even better, to list the files in the current directory using the long format, with the file size in a readable format, in order of reverse time

```
ls -lhtr
```

If we wish to list only the directories in a folder, we can use pipes in the following way

```
ls -l | egrep '^d'
```

or to just show files:

```
ls -l | egrep -v '^d'
```

---

tail

If we wish to show the bottom of a file on the terminal in "real-time" (ie. if it gets updated through some other running process), we can type

```
tail -f <name-of-file>
```

Output to screen and file

If we wish to ouput to both the screen and a file, we can use the "tee" command which converts standard input to standard output. The full command will look something like:

```
./exec 2>&1 | tee -a output.log
```

where "exec" is the executable we are running, the "2¿&1" says that error output is directed to standard output, we then "pipe" this output into the tee command which converted this into a file.

nohup

This runs a process in the background and will not stop running if you log out of the console. It basically means "no hang-up".

```
nohup python myscript.py
```

This would run a python script called "myscript.py" and would continue to run even if we shut down the console. Useful for scripts that are going to take a long time and for scripts that you execute on another machine.

---

whois

A nice way to find information about a a registered URL including details about the person who owns the URL. For example

```
whois gnomehome.co.uk
```

---

cat & lpr

To print to a printer connected to another computer through SSH we type

```
cat filename.pdf | ssh user@remotehost.org lpr
```

---

xmodmap

Annoyingly, the default for the scroll button on the mouse on linux is to copy. To disable this, we type in the terminal

```
xmodmap -e pointer = ‘‘1 9 3 4 5 6 7 8 2’’
```

which switches around buttons 2 and 9.

Joining pdf files

The following commands can be used to join two pdf files

```
pdftk fly1.pdf fly2.pdf cat flyer_combined.pdf
```

---

gcc and g++

These are used to compile C and C++ programs. eg.

```
g++ -c main.cpp
```

where the -c flag is used to say that we are compiling. g++ actually comes under the general gcc (Gnu Compiler Collection) and when we use gcc to compile a c++ program we are acutally using g++ behind the scenes. However, they have subtle differences in usage, so be careful!

Cmake

This is one of the most powerful programs that I have used while programming. Generally, the way that programs are created is that the source files are compiled and linked together by writing an appropriate Makefile. But once we start moving to other platforms with different compilers (usually termed "build environments"), then we often encounter problems. To handle this, and therefore allow compilation in a platform-independent way, we use Cmake.

As an example, imagine a very simple file called main.cpp that resides in a directory called "src". In this directory we create a file called "CMakeLists.txt" which would look like this:

```
cmake_minimum_required( VERSION 2.6 )
project( TESTER )

add_executable( main main.cpp )
```

The first line tell cmake which version to use, the second line gives a name to the project and the last line gives a name to the exectuable with the source files that we need to compile to produce the exectuable (binary). If our header files were kept in a different directory, then this would need to specified as

```
include_directories( ${TESTER_SOURCE_DIR}/include )
```

Now you may ask why we would separate out the source files and header files in this manner? The reason is that we may wish to distribute our code to other people without giving them our source files. We would create a "library" (essentially all the source files compiled and then combined into a single file) along with the header files so that they know what the functions are.

Once we have created our CMakeLists.txt file, then we may wish to create an exectuable (or binary as the programming guys call it) in a different directory from all our source code. We call this the build directory. So imagine the following:

```
cd ../
mkdir build
cd build
```

We have just created a build directory and moved into it. We want to create the exectuable in this directory, so we run the following command

```
ccmake ../src
```

(where "myProgramFolder" is the location of our CMakeLists.txt file). We see a user interface where certain options can be set, but we simply type 'c' to configure the build directory. We will then see list of entries with a * next to each to indicate it is new. We press 'c' again to tell cmake to process each of these new entries. Finally press 'g' to exit the program and generate the cmake files. We should see a number of new files in the directory, including a Makefile. So just type

```
make
```

and then an executable will be created in this directory.

Generating Xcode builds

Cmake also has the ability to generate Xcode builds which allows to work with our code in Xcode if we so wish. To do so, we change in to the build directory where we want the Xcode project to reside, and run the following command

```
ccmake path/to/Src -G Xcode
```

Running through the options in ccmake's gui (where 'c' must be pressed twice and then finally 'g'), we will see a .xcodeproj file. Open this up in Xcode and see all your code.

One thing I did notice when going through the source file folders in Xcode was that the header files were not appearing. This is because we must add the header files to the "add-executable()" command in the CMakeList.txt file. So we might see something like

```
add_executable( main ${SOURCE_FILES} ${HEADER_FILES} )
```

CMake uninstall

The general idea with CMake is that you can create 'out-of-source' builds. That is, give a folder which contains all the header and source files for your project, you can create the libraries and executables in a seperate folder. With CMake we can also run the command

```
make install
```

which will put the relevant header files, library files and exectuables in the installation directory we specified in the CMake build. However, in some Makefiles there is an option of uninstalling a program. This removes all the header files, library files and exectuables. CMake by deafult does not provide this facility, but we can get around this if we wish by modifying the CMakeLists.txt file and creating an input file in our source directory. But a much easier option is to simply run the following command in the build directory

```
xargs rm < install_manifest.txt
```

nm

Used to inspect an object file

```
nm objectFile.o
```

Not entirely of much use to me at the moment, since I don't understand the output. But essentially, an object file is produced from a translation unit (ie. a source file which has been pre-processed with all header files included). Object files will then contain blanks which refer to other functions or variables in separate translation units. It is the job of the linker to fill in these gaps.

---

gprof

Used to profile an executable if compiled with g++ or gcc.

If we wish to find out if a particular function is taking a lot of cycles in our exectuable, then we can use this program to give us more information on what functions are hogging the cpu. To use gprof, we need to include the flag -pg when we compile. So if we have a c++ file called main.cpp, then we would use gprof as follows:

```
g++ -Wall -pg -c main.cpp
```

This will compile the file. We then create the executable:

```
g++ -Wall -pg -o main main.o
```

We run the exectuable to generate the profile

```
./main > output.txt
```

(Here I have forced output to go to a file called output.txt). We then see the profile info by typing

```
gprof main
```

And it should appear on the screen.

An example make file for a program that consists of three files main.cpp, vector.h and vector.cpp is given below:

```
CC = g++
CFLAGS = -Wall -pg

OBJECTS = main.o vector.o

main: $(OBJECTS)
$(CC) $(CFLAGS) -o main $(OBJECTS)

main.o: vector.h
$(CC) $(CFLAGS) -c main.cpp
vector.o: vector.h
$(CC) $(CFLAGS) -c vector.cpp
clean:
rm -f *.o gmon.* output.txt
```

More details given at

```
http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html#SEC2
```

---

zip

Well, to make things easier when uploading images when submitting a journal (eg. to Comp Mech), it is convenient to use zip to compress the files and upload them as one. What I did (which was successful) was to compress all the figures which start with 2, 3, etc. and upload them in batches of 10 (or less). This can be done with the following command

```
zip archive.zip figure2*
```

And all the figures that start with 'figure2' will be put in the zip file.

tar

To compress a folder into a "tar" archive, we can type the following

```
tar -zcvf nameOfCompressiveArchive.tar.gz folderToCompress
```

And voila!, the folder will be compressed. The flags mean the following

- -z: use gzip for compression
- -c: create an archive file
- -v: verbose output
- -f: specify a file name

Uncompress tarball

If we wish to uncompress a tar archive we can do the following

```
tar xvzf folderToUncompress.tar.gz
```

As a side note, we do not need to precede the arguments with a dash, as long as they are 'clumped' together in what is known as the 'old style'. An equivalent 'new style' would be

```
tar -x -v -z -f folderToUncompress.tar.gz
```

Inspect contents of tarball

To inspect the contents of a tarball, simply type

```
tar ztvf folderToInspect.tar.gz
```

Diff

The diff command is used to tell the differences between one file and another file (which we might have 'fixed'). We usually use the unified format, which means we must pass the 'u' flag to the command. To create a diff file we can do the following

```
diff -u originalFile.txt modifiedFile.txt > patch.txt
```

This will output the diff command to a patch file that can be sent to other developers to use as a fix.
We can also make a patch from an entire directory structure

```
diff -ur originalFolder modifiedFolder > patch.txt
```

Patch

And when we have a patch file we can incorporate the changes using the patch command as follows:

```
patch -p0 < patch.txt
```

The 'p' flag tells the patch command to use the same directory structure for the existing directory and the patch directory. If, for instance we were to pass '-p1' then the first character of each of the directory names in the patch file would be omitted (ie. instead of using /user/Documents it would use user/Documents).

sftp

Used to upload and downloade files between a remote server and local machine. First, we login

```
sftp gnome-im@gnome.im
```

Then we are prompted for a password. One we are in, we see the prompt

```
sftp>
```

We can type the following to see what files are in the current directory on the remote machine

```
ls
```

Or by puting a 'bang' in front of the command we can see the local files

```
!ls
```

We can print the working directory in the remote and local machines as

```
pwd
```

```
lpwd
```

But the interesting feature is the ability to download and upload using the get and put commands eg.

```
put work.zip
```

This will upload the zip file to the current directory on the remote computer.

Matlab

Find current users on licence (Linux)

If we want to find out who is using our current licence, first change directory to the matlab installation directory ie.

```
cd <installation directory>/matlab/etc
```

then we can type the following command

```
./lmstat -a
```

which will list all sorts of details about the licence.

---

find

A very useful command that, on its own will return a matrix of all the nonzero indicies of a matrix. eg.

```
find(a)
```

will return a matrix of the non-zero indices of the matrix a. It can also be used with a condition statement like

```
find(a > 0.5)
```

which will return all the indices of the matrix a which are greater than 0.5.

---

setxor

Given two vectors A and B, this function will return the values that are NOT in the intersection of A and B. This has come in very hand for when I have a vector of nodes which defines the fixed nodes another vector which defines all the nodes in the domain. This function will return all the free nodes.

```
A = [1 2 3 4 5 6 7];
B = [2 4 6];
C=setxor(A,B)  % C is printed which is given by C=[1 3 5 7]
```

Other functions similar to this are the functions *intersect()* and *union()*

---

save

Useful for saving values to ASCII files that can be imported to other applications eg. for plotting

```
save 'export.dat' array -ASCII
```

Mex files

This is used to allow compiled c files to be integrated into matlab programs. Unfortunately it only works with gcc 4.1 so we need to do the following on a linux machine

```
sudo apt-get install gcc-4.1
```

and then

```
mex -setup
```

This creates a file called mexopts.sh in the matlab home directory (which can be found by typing 'matlabroot'). Change the appropriate lines in this file (according to the computer architecture) to

```
CC ='gcc-4.1'
```

Delaunay Triangulation

This is what is used to great 'nice' triangular meshes that do not contain large angles. Say we have a set of nodes which are prescribed in the vectors $\mathbf{x}$ and $\mathbf{y}$, then we can create a triangulation very easily as

```
x = rand(100,1); y = rand(100,1);
elConn = delaunay(x,y);
triplot(elConn,x,y);
```

But another function which is very useful is DelaunayTri() which allows us to set constaints on the boundary of the triangulation. The use of this becomes apparent when we start using more complicated geometries. We might use it in the following way

```
% assume we have nodes in the matrix, eg. nodes = [x1 y1; x2 y2; ... ; xn yn];

% assume we have a boundary connectivity matrix, eg.boundConn = [1 2; 2 4; 4 6...];
elConn = DelaunayTri(nodes(:,1), nodes(:,2), boundConn);
IO = inOutStatus(elConn);
triplot(dt(IO, :), elConn.X(:,1), elConn.X(:,2))
```

Cups network printing

To setup the CUPS printing system we need to do a few things. First off all, we need to edit a file located in

```
/etc/cups/cupsd.conf
```

My file looked like this

```
LogLevel warn
MaxLogSize 0
SystemGroup lpadmin
# Allow remote access
#Port 631
Listen 131.251.176.142:631
Listen /var/run/cups/cups.sock
# Enable printer sharing and shared printers.
Browsing On
BrowseOrder allow,deny
BrowseAllow all
BrowseRemoteProtocols CUPS
BrowseAddress @LOCAL
BrowseLocalProtocols CUPS dnssd
DefaultAuthType Basic
<Location />
  # Allow shared printing...
  Order allow,deny
  Allow @LOCAL
  Allow 10.74.*
  Allow 131.251.*
  Allow 10.73.*
  Allow 10.13.*
</Location>
<Location /admin>
  # Restrict access to the admin pages...
  Order allow,deny
  Allow @LOCAL
</Location>
<Location /admin/conf>
  AuthType Default
  Require user @SYSTEM
  # Restrict access to the configuration files...
  Order allow,deny
</Location>
<Policy default>
  <Limit Send-Document Send-URI Hold-Job Release-Job Restart-Job Purge-Jobs Set-Job-Attributes Create-
    Require user @OWNER @SYSTEM
    Order deny,allow
  </Limit>
  <Limit CUPS-Add-Modify-Printer CUPS-Delete-Printer CUPS-Add-Modify-Class CUPS-Delete-Class CUPS-Set-
    AuthType Default
    Require user @SYSTEM
    Order deny,allow
  </Limit>
  <Limit Pause-Printer Resume-Printer Enable-Printer Disable-Printer Pause-Printer-After-Current-Job H
    AuthType Default
    Require user @SYSTEM
    Order deny,allow
  </Limit>
  <Limit Cancel-Job CUPS-Authenticate-Job>
    Require user @OWNER @SYSTEM
```

```
    Order deny,allow
  </Limit>
  <Limit All>
    Order deny,allow
  </Limit>
</Policy>
<Policy authenticated>
  <Limit Create-Job Print-Job Print-URI>
  AuthType Default
  Order deny,allow
</Limit>
  <Limit Send-Document Send-URI Hold-Job Release-Job Restart-Job Purge-Jobs Set-Job-Attributes Create-
AuthType Default
Require user @OWNER @SYSTEM
Order deny,allow
  </Limit>
  <Limit CUPS-Add-Modify-Printer CUPS-Delete-Printer CUPS-Add-Modify-Class CUPS-Delete-Class CUPS-Set-
  AuthType Default
  Require user @SYSTEM
  Order deny,allow
    </Limit>
  <Limit Pause-Printer Resume-Printer Enable-Printer Disable-Printer Pause-Printer-After-Current-Job H
    AuthType Default
    Require user @SYSTEM
    Order deny,allow
      </Limit>
  <Limit Cancel-Job CUPS-Authenticate-Job>
      AuthType Default
      Require user @OWNER @SYSTEM
      Order deny,allow
        </Limit>
  <Limit All>
        Order deny,allow
          </Limit>
</Policy>
```

I then restarted Cups by typing the following command

```
sudo /etc/init.d/cups restart
```

After a bit of faffing around I managed to configure a printer server on my linux machine so that others can use it. After installing it, it is a simple process of simply going to the url:

```
http://localhost:631
```

and all the settings for modifying the printer server are there. Also to restart and stop the printer we can simply type in the command line

```
sudo /etc/init.d/cups restart
sudo /etc/init.d/cups stop
```

Windows

We can also connect windows machines to the CUPS server on my linux machine. To do this, we go through the "add a printer" option and then type in the URL:

`http://131.251.176.142/printers/HP_LaserJet2055dn`

and we should be able to install an appropriate driver for the printer (HP).

Getting a mac to print to the printer through EDUROAM

I have also managed to get my mac to print through the linux server by doing the following

1. Open up a web-browser and type localhost:631
2. Click 'Adding Printers and Classes'
3. Click 'add a printer'
4. Scroll down to 'Other Network Printers' and click 'Internet Printing Protocol (ipp)'
5. In the box type

   `ipp://131.251.176.142/printers/HP_LaserJet_P2055dn`

6. Click continue and then type in a name (something like "RobsAmazingPrinter") and a description (like "HP Laser jet which is controlled by Rob") and location ("Room 2.14").
7. Click continue and then go through the options to first select "HP" as the printer make and "HP LaserJet 2055 duplexer" as the printer
8. Click "add printer" and then click "set default options"
9. Print a test page to see if it works. And if not, get in contact with me (robertnsimpson@gmail.com)

Mac software

Adobe Illustrator (Mac)

To change to another open window we simply type the following command

`cmd + ~`

Pages

Sometimes if we want to put an umlaut over certain letters we can do the following: Type

`ALT + u`

Then type the relevant letter you want the umlaut over.

Windows software

Rhino

To export an "analysis-ready" geometry from T-splines we issue the following secret command:

Mac terminal commands

Moving to the end and beginning of a line in the termainl

Quite simply, we type ctrl-E to move to the end of the line and ctrl-A to move to the beginning. We can also delete the text on the current line by simply typing Ctrl-U.

Open a package with default program

```
open document.pdf
```

This will open the document.pdf with the default program on the mac. On mine, it was adobe reader. We can also specify the application to use with the -a flag. For instance,

```
open -a /Applications/TextWrangler.app foo.txt
```

Diffpack

Installation

Now this is a task and a half. Having just about installed Diffpack on a Ubuntu machine I can say that there are a few things that were not immediately obvious to me during the process. Here are some pointers

1. The first thing to do is to copy the files from the cd to an appropriate directory. In all the examples given by Diffpack, it is installed in /usr/local/, so I just put the files there.
2. I created a temporary folder called dptmp and within this I put the folder kernel (of the appropriate architecture type). eg. for a 32-bit linux system this folder is in the linux-gcc402 folder on the cd.
3. We then issue the following command to run the installation script

   ```
   sh dptmp/kernel/install-dp.sh -r /usr/local -m linux-gcc-4.2.3
   -s /usr/local/dptmp/kernel
   ```

4. We now need to edit the .profile file by placing come lines at the end. We can edit it by typing **sudo vi /.bashrc** (in the case of linux) or **sudo vi /.profile** (for my mac). We then put the following lines at the end of the file

```
export NOR=/usr/local/NO
export MACHINE_TYPE=linux-gcc-4.2.3
. $NOR/etc/setup/dpshrc
```

5. We are now able to create a project with Diffpack by changing to a suitable directory and then issuing the command **Mkdir newprojectname**. This will create a folder where we will store our source files.
6. We can create a file called **newprojectname.cpp** and in this we can use all the various Diffpack classes and variable types to get a FE code running. Once we have saved the file we can compile and link the code just by simply typing **Make**.
7. However, on my first attempt it didn't compile straight away since there were several libraries that I needed to install. For example, I got the error about some library called -lXext - this is solved by going into the Ubuntu package manager, searching for the library and installing it. This may have to be done for several libraries[1].
8. Finally, it is necessary to get a licence key for Diffpack which requires both a hostname and hostid. We can get the hostname by typing **uname -a**. But to get the hostid I need to change into the directory $NOR/ext/FLEXlm/linux-gcc-4.0.2 and run the command **./lmutil lmhostid**

Mac usage

To use Diffpack on a mac requires some extra little tricks to get things working. For a start, I had to modify my .profile file stored in my home directory (cd). I added the following lines

```
export PATH=.:$PATH
export NOR=/usr/local/Diffpack/NO/
export MACHINE_TYPE=mac-gcc-4.2
. /usr/local/Diffpack/NO/etc/setup/dpshrc
alias dmkdir='/usr/local/Diffpack/NO/bin/Mkdir'
```

Notice the last line which creates an alias for the command 'Mkdir'. The command works in linux, but since the mac terminal is case-insensitive mkdir and Mkdir perform the same command. So I have made an alias called 'dmkdir'. When we run

```
dmkdir project
```

we will see that a directory is created with the appropriate make file. We can simply type 'make' to compile the source code with the diffpack library.

Latex

Changing the default font

Just type this in the preamble

```
\renewcommand{\familydefault}{\sfdefault}
```

---

[1] Some of the packages I needed to install were Xext, libxt6-dbg, libxt-dev, x11proto-xext-dev, tcl, osmesa, tk8

Accelerate framework (Mac)

Compiling

If we have a project which is using the accelerate framework we can compile with the following

```
gcc lapack.c -framework Accelerate -std=c99 -o lapackExample
```

We need to specifiy we are using the accelerate framework with the -framework flag. By specifying -std=c99 we are making sure that we are making the most of the latest compiler. The final flag specifies the output file.

OOFEM

Installing

This is detailed in the readme file, but the basic steps are

```
untar -zxvf OOFEM.tar.gz
./configure
```

we then go to the directory

```
cd targets/default
```

and run the make file

```
make
```

We can test it by running

```
make tests
```

Now, in order to run the code, we fine that the executable is contained in targets/default/bin. To run it with an appropriate input file we place the input file next to the execuatble and run

```
./oofem
```

GSL

Compiling and Linking

We can install this package through the package manager easiliy in Ubuntu, but it seems that it installs the headers in a path that is not recognised by the compiler. So to compile on my machine I had to type

```
g++ -c hw.cpp
```

And then to link

```
g++ -L/usr/include/gsl hw.o -lm -lgsl -lgslcblas
```

Usually the libraries are kept in the path /usr/local/lib

To get the same thing working on a mac I had to download gsl using macports and then I can compile the same program doing

```
g++ -I/opt/local/include -c hw.cpp
```

And then to link

```
g++ -L/opt/local/lib hw.o -lgsl -gslcblas -lm
```

Done!

GIT

This is version control system that I am using to keep track of my isogeometric BEM code. By storing it on the GIThub, the code is opensource and available to all. It should make working on things colloboratively much easier in the future.

First of all, in order to contribute to Github you will need to set up an account at www.github.com. After this, I will be able to add you as a contributor to my project but otherwise, you will still be able to "fetch" code from the repository (read-only access). Let's assume you have an account on GIThub and you have gone through the setup up process on your machine (it is very well described by GIThub). You would get my isogeometric stuff as follows

config

When we first use git on our machine we need to set up certain user defaults for the user email and username. To do this, we type

```
git config --global user.name = "Rob Simpson"
git config --global user.email = "robertsimpson@gmail.com"
```

And to see if our changes are correct, type

```
git config --global -l
```

Clone

```
git clone git://github.com/bobbiesimpson/Isogeometric-BEM.git
```

This will create a directory along with all the files that are stored on the repository. It is good to read the README file to make sure that things will work correctly when you run the matlab scripts.

One thing which puzzled me for a whle was the syntax "git://" with other terms like "http://" and "ssh://" also valid.This refers to the *transfer protocol* and if you want more details, then type in

```
man git-clone
```

and scroll down to read more.

Add

```
git add isoBEM.m
```

Here, we have changed the file isoBEM.m and we have "staged" it. This means that we have included it in a snapshot that will be later included in future "commits" to the repository. Basically, every time we modify a file and we want those changes to be seen on the repo, we will call this command on those files.

.gitignore

Sometimes there are files that we do not wish to include in our commits to the repository. What we can do is to create a file in the local working directory called

```
.gitignore
```

and put all the file types in here that we wish to exclude. For example, it might look like:

```
*.log
*.gz
*.bbl
```

rm

If we wish to remove items from the working tree, then we do not just simply use the rm command, since this will confuse git. Instead, we remove files using the following

```
git rm <fileToRemove>
```

Status

```
git status -s
```

This will show which files are modified on our current branch and therefore will be included in our next commit. It is good to run this command before we do any commits.

Diff

```
git diff
```

We can see what changes have been made before we do any commits. Another useful thing to do before a commit.

Commit

```
git commit -m ‘‘A relevant message is written here about the commit’’
```

This command will actually put the changes we have marked in our snapshot onto the repository.

One further comment which should be made on this command is the format of the messages submitted on the command line. I have found that git **does not wrap words over lines** when you use the command git log. To over come this, when the message is being typed in during a commit, make sure to use a carriage return at appropriate points in the text. Then, when you look at the git logs, everything will appear in a nice format, rather than one, often very long, line.

We can also commit while bypassing the need to add files to the staging area. This is achieved with the 'a' flag as follows:

```
git commit -am "A commit which does not require the add command"
```

Log

```
git log
```

See all the commits that have taken place and who made them

If we wish to see a (kindof) graphical output of the logs, we can type the following:

```
git log --oneline --graph
```

Remote

Remote add

```
git remote add isobem git@github.com:bobbiesimpson/Isogeometric-BEM.git
```

This will add an alias for our remote repository - makes things a lot more convenient later on

remote -v

Just typing in

```
git remote -v
```

will show all the currently saved remotes and the URL for each.

Push

```
git push isobem master
```

This will put all the changes we have made in the master branch onto the remote repository. This can be seen by all on Github!

Fetch

```
git fetch isobem
```

This will grab all the changes made as seen on the remote repository

Merge

```
git merge isobem/master
```

This will follow a Fetch command and will merge the remote changes into the current branch
This can also be performed using the following command

```
git pull . /remotes/isobem/master
```

which can be read as "pull from the remote (located in the current directory) from the branch master and merge with the current branch".

Rebase

This is a slightly more advanced command that is similar to merge. Instead of taking out latest local commit and then applying the merge on top of this version, rebase will 'roll back' our code to a suitable point at which the remote code can be added. Then the changes are added and our local changes are added on top. This has the potential to lose some of our commits (I think) but keep the log much cleaner.

We use it like so (after perfomring a fetch):

```
git rebase isobem/master
```

which is very similar to merge, but with very different consequences.

Pull

We can also combine the previous two commands into one using

```
git pull isobem master
```

which merges the branch on the server (isobem) into the current working branch.

remote branches

```
git branch -r
```

Will show the remote branches

0.1 Creating a remote repository

Sometimes we want to create a remote repository on a server so that we can do wonderful things like collaborative paper writing using version control. Let's assume that we are starting from scratch and therefore wish to create an empty repository on the server. First of all, let's create a git repo on our local machine which will contain the files that will initialise the remote repo.

```
<on the local machine>
mkdir myrepo
cd myrepo
<add some files>
git init
git add .
git commit -m "Start of my repo"
```

Now log-in to the remote machine through SSH and create a new repository folder

```
mkdir myrepo.git
cd newrepo.git
git --bare init
```

To make things easier on the local machine, we can create an alias for the remote server as

```
git remote add origin ssh://gnome-im@gnome.im/home/gnome-im/gitrepos/marking/MSc/Y3_briefs_2011.git
```

We are now in a position to "push" our local files to the remote repo as

```
git push origin master
```

and now people can "clone" the git repo from the remote server using the "clone" command as detailed above.

Push with 'u' flag

When we push to a repository for the first time, it is often a good idea to specify the 'u' flag as follows:

```
git push -u origin master
```

This essentially sets up a relationship between the remote branch and the local branch so that if you want to pull from the remote, you can simply type

```
git pull
```

(assuming you are currently in the relevenat local branch).

Blame

To find out who changed a file and what they changed, we type

```
git blame fileToInspect.txt
```

and we will see the relevant output on the screen.

format-patch

To create a patch after adding and commiting changes to the repo, we can create a patch file as follows

```
git format-patch origin/master
```

and it will be created with an appropriately named file.
10mm

opennurbs

This rather neat library which has been written by the guys at Rhino allows us to interface with CAD obtaining all the necessary data to construct NURBS etc. This is especially useful for the isogeometric analysis side of things. But to get things working it is necessary to know a few details.

Xcode

We can use opennurbs as long as we do two things - include the static library and let xcode know where the header files are. To link the library we can do the following

- Click on the "target" and the click "info"
- Browse to where the file "libopennurbs.a" resides and add it to the linked libraries section

However, if the library resides in a system directory like /usr/local/lib then we can go the "build" section and add the following to the "library search paths": /usr/local/lib
We also need to specify where the header files are, and to do this we edit the "header search paths": /Users/Robert/Documents/Work/Cardiff/Lectureship/ResearchTopics/isogeometric/code/opennurbs
Finally, we must specify a linker option which we type into the "other linker flags": -lopennurbs
Hopefully the program should now compile.

Mac (terminal)

I managed to get a code to run which was compiled in the terminal and linked against the openNURBS static library. To do this, we first need to build the openNURBS library, which is done by changing into the directory containing the openNURBS source code and typing 'make'. Then, we can create code which uses the openNURBS library by including the following header:

```
#include opennurbs.h
```

We can then compile the file (in this case it is called main.cpp) in the following way

```
g++ -Wall -I/<path_to_header_files> -L/<path_to_library_file> main.cpp -lopenNURBS -o main
```

where the -I flag specifies the path to the openNURBS header files and the -L flag specifies the path to the static openNURBS library (libopenNURBS.a). We link with the library using the -l flag and create an executable called main.

Linux

I've also got it to work on Linux. But since I copied the files across from my mac, it was first necessary to run (after changing into the opennurbs directory)

```
make clean
```

Then it is a simple case of running

```
make
```

Note that when linking with the static library we must use the flag

```
-lopenNURBS
```

which differs from the mac version

```
-lopennurbs
```

Eigen C++ library

```
http://eigen.tuxfamily.org/index.php?title=Main_Page
```

This library is a collection of classes that allow matrices to be stored and provides routines for solving linear systems of equations. It can almost be viewed as an extension of the STD library (e.g. the ¡vector¿ class). In order to install it, download the code from the website (given above) and then we use the library in one of two ways:

1. Include the "eigen" folder that you downloaded in the same folder that the code you are compiling. We can then simply run e.g.

   ```
   g++ -Wall main.cpp -o main
   ```

   If the folder is stored somewhere else, then we would explicty need to specify the include path

   ```
   g++ -Wall -I /path/to/eigen/folder main.cpp -o main
   ```

2. We can copy the eigen folder to the include directory on your system. On my mac, this was in /usr/local/include . After this, we can simply run the first command given above (where we do no need to specify the include path since by default, the compiler will look in /usr/local/include)

SHELLs

echo

It appears that the default shell on my mac is bash, but to see what shell you are using, simply type

```
echo $SHELL
```

Or, to display special characters like newline, we can type

```
echo -e 'Display text with a newline \n'
```

0.2 Echo the return code of an exectuable

You may notice that the function main() in C and C++ programs has a return type of int. This value is known as the exit code. To query what an executable has returned, we can simply type

```
echo $?
```

Crontab

This is a very useful program that allows shells scripts to be executed at a specified time. One reason that I have wanted to do this is to backup my files and then send them to a server. We can create a crontab as follows. Type

```
crontab -e
```

in bash. Then we can create a lines as follows:

```
0 13 * * * ~/Documents/backup.sh >> ~/Documents/backupLogfile.log 2>&1
```

This executes the script backup at the first minute of 1300 everyday and appends the results to a logfile. There is also an additional command at the end "2>&1" which means that standard error output is directed into standard output. The flags for standard input/output are as follows: By default, STDOUT is chosen.

Table 1: Std input/output flags

STDIN	0
STDOUT	1
STDERR	2

Backup Shell Script

Following on from the previous section, we can look into the backup script uses rsync to see what files have been updated and send the appropriate files across to a remote machine. It works like this:

```
#!/bin/bash

if rsync -va ~/Documents/Work ssh robert@131.251.176.142:Documents/LaptopBackup
then echo 'upload successful'
else echo 'problem uploading'
fi

date
```

The rsync function is called with two arguments -v (verbose mode) and -a (archive mode where all permissions, symbolic links are preserved). The files are transferred to a remote machine into the folder /Documents/LaptopBackup. A error/success message is printed and finally, the date is printed.

ssh-keygen

In the previous bash script the rsync function was used without any need for a password. This is because I used secure id_rsa based authentication. This is the recommended way of connecting through ssh since essentially we are setting up a lock and key system, where the public key corresponds to a unique lock, and the private key (stored on our local computer) corresponds to the key. The system relies on making two files: a private key and a public key. We can do this as follows: first,

```
ssh-keygen -t rsa -b 2048 -f ~/.ssh/id_rsa
```

Now, a small point. I *did not use a passphrase when generating this key for the above bash script*. This might seem like a compromise on security, but it makes things a lot easier when using the crontab, since if we *do* use a passphrase, then the crontab cannot login using ssh and the bash script will fail. There are ways around this (http://meinit.nl/using-rsync-from-cron-with-ssh-keys-that-have-a-passphrase), but I have not implemented them.

This will generate two files in the .ssh folder: is_rsa (private key) and id_rsa.pub (public key). The 2048 means that it is generated using a 2048 bit key. We then create a file called "authorized_keys" by

```
touch ~/.ssh/authorized_keys
```

Then change the permissions to make it execuatable

```
chmod 600 ~/.ssh/authorized_keys
```

Then append the public key just generated to this file

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Apparently on some systems we need to uncomment a line in /etc/ssh/ssh_config

```
# IdentityFile ~/.ssh/id_rsa
```

(But I didn't need to do this on the mac)
Now we upload the authorized keys to the server (assuming the dirtory /.ssh exists)

```
stfp> put ~/.ssh/authorized_keys .ssh/authorized_keys
```

(or just use an FTP client like FUGU)
We can now check if everything works if we try logging in using ssh. If all is well, we shouldn't need a password.

Ubuntu version of settoing up SSH keys

Ubuntu also has a very nice tutorial on how to set up keys for securely logging in with SSH. It essentially does the same thing and can be summarised as follows:
If the directory

```
~/.ssh
```

doesn't exist, then create it and give it executable permissions with

```
chmod 700 ~/.ssh
```

We then simply create a key with

```
ssh-keygen -t rsa
```

which will prompt us for a password. Type one in if you wish, but it is not required. The consequence of typing one in now will be that a password will be need to be entered every time you connect using SSH.
A more secure option is to use more bits to genereate the key with

```
ssh-keygen -t rsa -b 4096
```

We now transfer the public key to the required server with

```
ssh-copy-id user@server.com
```

and we should be able to access the server through ssh and our newly generated key with

```
ssh user@server.com
```

Creating alias for ssh

Rather than having to type in "ssh robert@131.251.176.142" we can create an alias instead. Open, or create the file

```
~/.ssh/config
```

Then add the following lines (for example)

```
Host=linuxBox
Hostname=131.251.176.142
User=robert
```

We can then connect simply by typing

```
ssh linuxBox
```

/dev/null/

This is a special file that discards all data written to it eg.

```
cat /dev/null > fileToDeleteContents.txt
```

This will delete the contents of the file 'fileToDeleteContents.txt'

Ubuntu

Command line mode

Simply type Alt + F2 and you will be there!

GPU programming

Installation of CUDA

Quite simply, the documentation provided by NVIDIA is excellent. And so go to

```
http://developer.nvidia.com/cuda-toolkit-40
```

and download the relevant package for your system.

Installation on Mac OS X

When we install on the mac, the installation directory (on my machine is kept at)

```
/Developer-3.2.6/GPU Computing
```

Using CMake with CUDA

After a bit of faffing around, I have got CMake working with CUDA. If we have a file called vectorAddition.cu that uses cuda code, then we can create a CMakeLists.txt file as follows:

```
PROJECT( vectorAddition )
CMAKE_MINIMUM_REQUIRED( VERSION 2.8 )

FIND_PACKAGE( CUDA )

IF ( APPLE )
   FIND_LIBRARY(CUDA_CUTIL_LIBRARY NAMES cutil_i386 HINTS "${CUDA_SDK_ROOT_DIR}/C/lib")
ELSE( APPLE )
   FIND_LIBRARY(CUDA_CUTIL_LIBRARY NAMES cutil HINTS "${CUDA_SDK_ROOT_DIR}/C/lib")
ENDIF( APPLE)

CUDA_ADD_EXECUTABLE( vectorAddition vectorAddition.cu )

TARGET_LINK_LIBRARIES(vectorAddition ${CUDA_CUTIL_LIBRARY})
```

Notice that there is an if statement to take account of the fact the mac version of cuda is 32 bit and therefore the library name is different.

A few pointers when incorporating CUDA in an existing c++ project

One of the key requirement when using CUDA is that is can be incorporated into an existing C or C++ project. Once you know what to do though, this task is actually than you first might think. Some of the commands which might help include the following:

```
OPTION( BUILD_WITH_CUDA "Build with cuda" OFF )
IF ( BUILD_WITH_CUDA )
 # Set up CUDA package
 FIND_PACKAGE( CUDA )
ENDIF( BUILD_WITH_CUDA )
```

which sets up an option to build with or without cuda.

```
CMAKE_ADD_LIBRARY()
```

which creates a library using the nvcc compiler

```
CMAKE_ADD_EXECUTABLE()
```

which creates an executable using nvcc.

VI/VIM commands

vimrc config file (mac)

The config file for the mac is stored in

```
/usr/share/vim
```

We edit the config file by typing

```
sudo vi vimrc
```

We can change things like tab spacing and shift width in this file. And also enable syntax highlighting by default. This is what mine looks like

```
" Configuration file for vim
set modelines=0     " CVE-2007-2438

" Normally we use vim-extensions. If you want true vi-compatibility
" remove change the following statements
set nocompatible    " Use Vim defaults instead of 100% vi compatibility
set backspace=2     " more powerful backspacing
set ts=4
set sw=4
set ai
syntax on

" Don't write backup file if vim is being called by "crontab -e"
au BufWrite /private/tmp/crontab.* set nowritebackup
" Don't write backup file if vim is being called by "chpass"
au BufWrite /private/etc/pw.* set nowritebackup
```

The lines that I added were the "ts" (tab spacing), "sw" (shift width), "ai" (auto indent) and "syntax on" (syntax highlighting).

Indent

To index 5 lines we type

```
5>>
```

and to deindent we type

```
5<<
```

Emacs (or aquamacs)

Ubuntu customisation

To customise emacs for Ubuntu we need to put the preferences in the following file:

```
~/.emacs.d/init.el
```

I also found that some annoying message came up if I ran emacs from the command line. But after doing some searching on the internet, some people had answered my question. What I found that you need to modify the file located at

```
/usr/share/themes/Ambiance/gtk-2.0/gtkrc
```

and change the line

```
GtkRange::trough-under-steppers = 0
```

to

```
GtkRange::trough-under-steppers = 1.
```

Aquamacs customisation

To customise emacs for Aquamacs we need to put the preferences in the following file:

```
~/Library/Preferences/Aquamacs\ Emacs/Preferences.el
```

Emacs is the editor which is preferred by programmers and using the ctrl and alt keys instead of being in certain "modes" as in VI. There are a few commands which I have picked up. The letter"c" denotes the ctrl key is pressed, and "m" denotes that the alt key is pressed.

Moving around

```
c-v : Move down one screen
m-v : Move up one screen
c-l : Move screen around cursor


c-n : Move to next line
c-p : Move to previous line
c-f : Move forward one character
c-b : Move backward one character
```

```
m-f : Move forward one word
m-b : Move backward one word


c-a : Move to beginning of line
c-e : Move to end of line


m - a : Move to beginning of sentence
m - e : Move to end of sentence.


m-< : Move to beginning of file
m-> : Move to end of file
```

Repeat commands

```
c-u 8 <command to repeated 8 times>
c-u 8 c-n (move forward 8 lines)
```

Stop a command

```
c-g: stop the current command
```

Undo a command

```
c-x u: Undo previous command
c-/ : The same (but easier)
```

Windows

Whenever we split the screen into different areas, we refer to each region of the "frame" as a "window." This means that multiple windows reside within a single frame. There are various command to manipulate windows.

```
c-x 0 : Delete current window
c-x 3 : open window to the right
c-x 2: open window below
c-x 1 : close all windows except the one we are in
c-x o : change to other buffer
```

Files and buffers

The ideas of a buffer is a place where we can stores some text, but this is not necassarily assciated with any file. So we could create a buffer, write some text in it and then delete it without ever creating a file. Buffers are fundamental to working with emacs, and here a few commands I have picked up

To save a file just type

```
c-x c-s : Save file
c-x c-w : Write to another location
```

```
c-x c-b : List all current buffers.
```

Sometimes confusingly, this opens a new window in our frame. If you are in the buffer which is displaying all the current buffers, then you can quit the buffer by simply typing 'q'.

```
c-x c-f <filename> : open a file in a new buffer. You can use tab-completion here.
c-x k <buffername> : kill a buffer
c-x 4 b <buffername> : open a buffer in a new window
m-x kill-some-buffers <RETURN>: this allows us to kill many buffers in succession
```

Using the shell

We can use the shell from within emacs as follows:

```
m-x shell
```

And then in the current window the shell will be shown. This is useful if we wish to execute commands like "make" or "cmake"

Copying and pasting

To cut a section of text first of all we must select the region by first typing

```
c-<spacebar> : and then moving to the point at which we end the selection
```

We then cut the selection with

```
c-w
```

and paste with

```
c-y
```

0.3 Deleting

Some nice commands for deleting text

```
c-x c-0 : delete every blank line except one
m-d : delete words forward
c-d : delete characters forward from current position
```

0.4 Searching

```
c-s : allows you to search for an item. Pressing the command again
finds the next item
c-r : backward incremental search
```

To exit the search mode, just type return, or c-g.

0.4.1 Search and replace

If we wish to search and replace items then we can type the following

```
m-x query-replace <stringToReplace> <RET> <newString> <RET>
```

Then, to actually replace each item in turn, we can press the space bar. Instead, if we wish to replace all future occurences, simply type

```
!
```

Regions

Emacs has a concept called a region which can be set manually using

`c-<spacebar>`

We can also see the current region by typing

`c-x c-x`

Programming

To indent the current block of code using the appropriate format, type:

`c-c c-q`

Compiling

We can compile a program through emacs using the following command:

`m-x compile`

In the small buffer "make -k" will appear, but we can simply delete this and replace it with (for example)

```
g++ -o main main.cpp
```

and this will be compiled within emacs.

Inserting special characters

I'm not entirely sure this is a generasl emacs command, but to insert a tab character in aquamacs we must type

`c-q <TAB>`

which is particularly important for makefiles.

Uppercase and lowercase

To convert a word to lowercase

`m-l`

and to upper case:

`m-u`

Auctex

This is feature which is installed with Aquamacs that allows some really rapid editing of latex documents. Here a few ones I have used in the past

Setting up Aquamacs

There are few things which are not immediately obvious when using Auctex with aquamacs. Probably the first issue I came across was that auctex creates pdfs by default, while I prefer working with dvis (to allow the inclusion of eps files). In addition, I noticed that when I viewed files from aquamacs, the pdf was shown through Texshop. So what I did was to go to the following menu in Aquamacs

```
Latex -> Customize Auctex -> Browse options
```

You will be presented with a file which allows you to change options, and this can be saved for all future sessions. The important ones I changes were "Tex view program list" (to just use Skim - a nice viewing program) and "Tex Pdf mode" which I switched to "off".

Get Aquamacs to make dvi, convert to pdf and then open Skim

Hold press! I managed to find one line which seems to solve all my problems! What this line does is to convert the dvi to a pdf and then open this in Skim - a very nice viewing program. What we do in Aquamacs is to go into the customize page (as above) and navigate to "Tex command" and then find "Tex command list". Under the sub-head "latex" we want to make sure that the values for "command:" is given as:

```
%`%l%(mode)%' %t && dvips -Ppdf %d -o && ps2pdf %f && open -a Skim.app %s.pdf
```

And, if we wish to have the file update automatically after we have compiled latex, then go into the "Preferences" in Skim and check the box "Check for file changes."

Now everything is set up perfectly for creating nice latex pdfs!

We can also omit the commands after 'open' and simply set the default program to open dvi files as Skim in Mac OX.

Change to pdf mode

If we want to use latex in pdf mode (using png and pdf image files), we can type

```
c-c c-t c-p
```

0.5 Previewing

For equations and other items that might be hard to visualise by inspecting latex code, it is sometimes useful to perform a preview using the following command

```
c-c c-p c-p: preview at point
c-c c-p c-s : section preview
c-c c-p c-b : buffer preview
c-c c-p c-e : environment preview
```

And we can remove buffers using the following commands

```
c-c c-p c-c c-d : remove preview from document
c-c c-p c-c c-r : remove preview from region
c-c c-p c-c c-b : remove preview from buffer
```

Compiling and viewing

```
c-c c-c LaTeX
```

The text "LaTeX" can be filled in with auto-completion with the tab key, and it is the default option so when can usually just type return after the two control key commands.

```
c-c c-v : View the output file
```

Environments

So if we want to include the commonly used " begin" and " end" syntax we can do this with

```
c-c c-e Equation : inserts equation environment
```

This can be used with other environments by pressing the tab key.

A similar command exists for sections, parts, chapters etc.

```
c-c c-s Section: insert section
```

0.6 Itemize

We can insert the item command if we are in an "itemize" environment by typing

```
c-c c-j
```

0.7 Spelling

Some spelling commands for emacs

```
m-$ : check spelling of existing word
m-x ispell : check spelling of entire buffer
m-x flyspell-mode : highlight all mispelt words
```

Math environment

This is a really nice feature which allows us to type maths commands really fast. To start math mode, we type

```
c-c ~
```

And then we can type a whole host of maths using commands like

```
`a == \alpha
`b == \beta
```

We exit math mode by typing the same command to start math mode.

Reftex

If nothing seems to be working with reftex, then we might have to turn reftex on with

```
m-x refex-mode <enter>
```

And if reftex mode is not started automatically when opening a latex file (which seems to be the case with emacs on Ubuntu) then we can edit the .emacs file by adding the following lines

```
(add-hook 'LaTeX-mode-hook 'turn-on-reftex)   ; with AUCTeX LaTeX mode
(add-hook 'latex-mode-hook 'turn-on-reftex)   ; with Emacs latex mode
```

If we have included a bibtex file in our document, then we can insert a citation quickly by typing

```
c-c [ <search-term>
```

Where we can search for a particular value to find the reference we want. Typing return will place the citation in the tex file.

I have found that sometimes this command does not work, and we may need to "reset the buffer" by typing the following:

```
c-c c-n : reset the buffer
```

and the command should work now.

And sometimes this still doesn't work, so I found that if we type

```
m-x reftex-parse-all
```

I find that the the bib file is now seen.

Automatically update the bibtex file

If we are simultaneously working on the tex and bib files, we can end up in a situation where an update in the bib file is not seen by reftex. A solution to over come this is to open up the bib file in emacs, and then type

```
m-x auto-revert-mode
```

This will ensure that any updates are seen immediately. It should be noted that this is only an issue when editing the bib file using an external editor.

0.7.1 Insert reference

If we wish to refer to a figure or section heading, then we can do this easily with

```
c-c )
```

and we should be shown a list of possible items we can use. Just by scrolling through these we can find the appropriate one we need and then hit enter. Pressing 'r' will also refresh the list.

Table of contents

Simply type

```
c-c =
```

and the table of contents will appear

Macros

For inserting things like graphics quickly we can type

```
c-c <return>
```

and then by using autocompletion (ie.the tab key), we can fill in a whole host of things like

```
\includegraphics{figure1.eps}
```