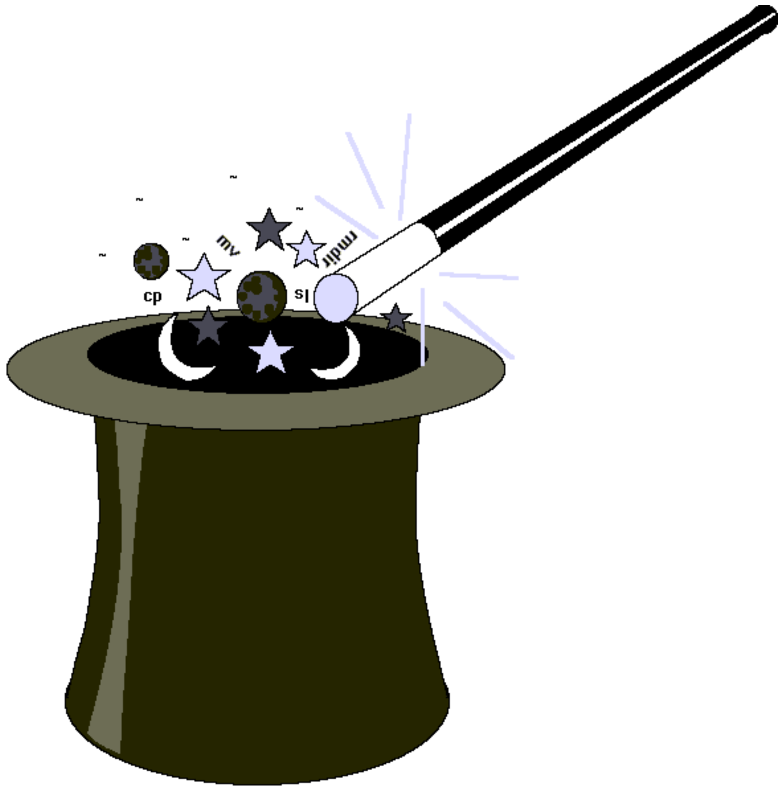
UNIVERSITY OF
BIRMINGHAM
Birmingham Environment for Academic Research

Unix – Guide for Beginners (Part 1)



Alan Reed, Aslam Ghumra, Chris Bayliss, and Jenny Harrison

Edition 5.08

Table of Contents

Part 1: Introduction

What is UNIX?

UNIX is a multi-user operating system; in other words it allows more than one user to have access to a computer at the same time. The users share the computer's various resources amongst them. Although each user has only a small portion of the resources of the machine at their disposal, the computer operates so fast that they have the impression of complete control over the resources they need. There are versions of UNIX in use on microcomputers and mainframes at business and academic sites all over the world.

What is Linux?

Linux is a free version of the UNIX operating system, primarily composed of tools developed over a 15-year period by Richard Stallman and Project GNU. Linux Torvalds wrote a kernel (completed in 1994) which launched the operating system. Its strengths are its open source and rapid development, in addition to the traditional strengths of UNIX (stability, internet, networked graphics, scripting) and the fact that it will run on a variety of hardware.

All the commands featured on this course run on all UNIX and Linux platforms, although there may be a few minor differences (e.g. error messages, some command parameters).

Usernames and Passwords

In order to be able to use UNIX, you need a username and password. Your username is the unique code to identify you and your files. Traditionally, usernames were limited to eight characters, but most modern systems will support longer usernames.

Your password is used to control access to the system and your files. This password acts as a "key" in order to allow you and no one else to create, examine, change or destroy your files.

As passwords are a major part of the UNIX security, it is important to choose a sensible one, which would be difficult for others to guess, and to keep it secret. Many attacks on systems by "hackers" have been due to bad choices of password.

The password consists of a minimum of six charact3rs. When choosing a password, avoid using simple words, names, or personal information such as car registration or your telephone number.

On IT Service servers, UNIX passwords must include a number as well as letters. In addition, the systems force you to change the password periodically in case it has become known to others. On many systems, only the first eight characters are significant, but longer ones can be used as they can sometimes be easier to remember.

On the IT Services Sun servers, new passwords must differ from previous ones in at least three places and force at least two characters to be something other than lower case letters.

Typing in Commands

In order to use a UNIX system, you need to log in. This is described below, but there are one or two important points common to all UNIX commands which are also useful for logging in.

Some commands require additional information, others do not. In all cases, once the command line has been typed, it must be followed by the use of the Enter key. It is necessary to use the Enter key after any input into UNIX.

Sometimes, you will make a mistake when typing. Errors can be corrected by using the Delete key. This key may be different on different systems, but is normally on the top right of the main block of keys.

Case Sensitivity

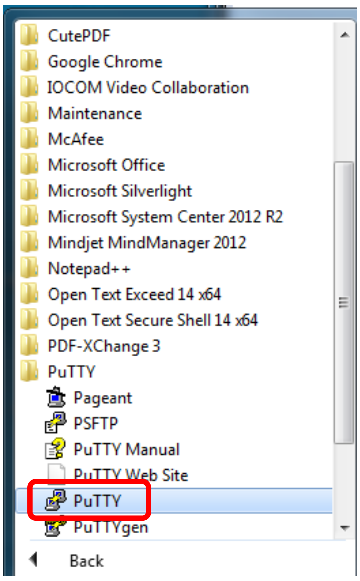
Unlike many other computer systems, UNIX makes use of the difference between upper and lower case characters. This is important when typing commands, filenames, passwords, and usernames. The system will not recognise any filename, command, or password which is given the wrong case.

Lower case is almost always used for usernames. Use of upper case can produce unwanted results on some older versions of UNIX.

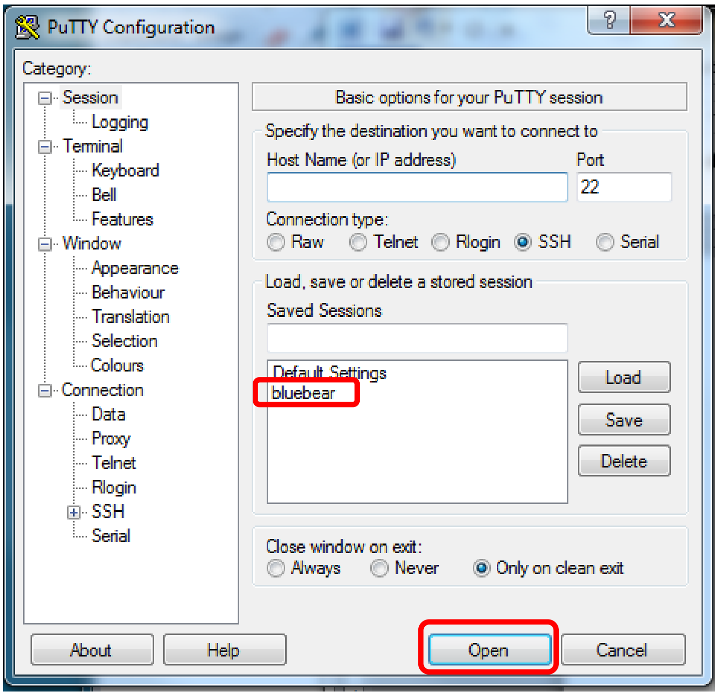
Using PuTTY to access BlueBEAR

Setup and configuration information can be found at (https://intranet.birmingham.ac.uk/it/teams/infrastructure/research/bear/bluebear/accessing-bluebear-using-putty-and-exceed.aspx#ConfiguringPuTTY)

Once installed, double-click on the PuTTY icon on your desktop, or go to the Start button in Windows¦ All Programs ¦ PuTTY ¦ click once on PuTTY to start the program.



Double-click on bluebear to start the program or click once on bluebear and click Open.



Specify the destination you want to connect to

Host Name (or IP address):

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Load, save or delete a stored session

Saved Sessions:

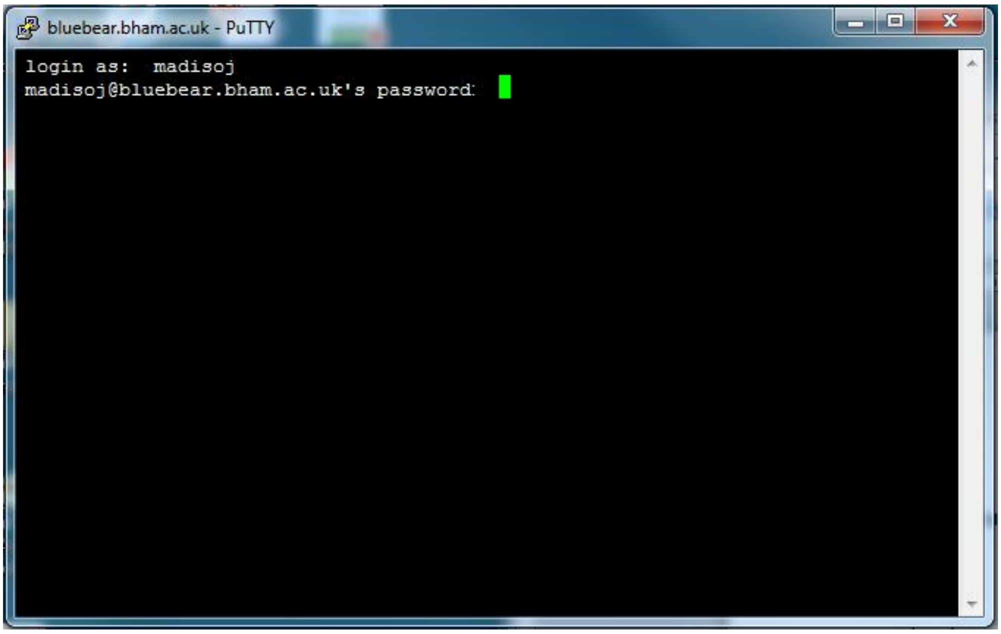
Default Settings
bluebear

Close window on exit: ○ Always ○ Never ● Only on clean exit

Logging In/Out

Once you have access the system from a telnet session, the system will prompt you to log in with a login: prompt.

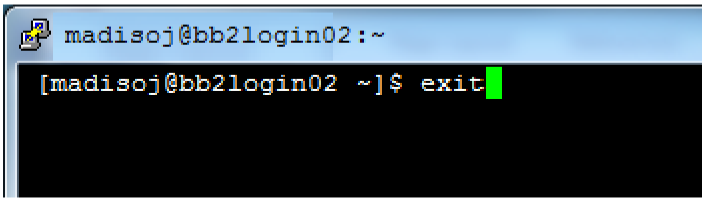
Log in using your user ID and password.



```
login as:  madisoj
madisoj@bluebear.bham.ac.uk's password:
```

You should respond with your username. Note that the password is not displayed on the terminal when it is typed. This is to help to keep your password secret.

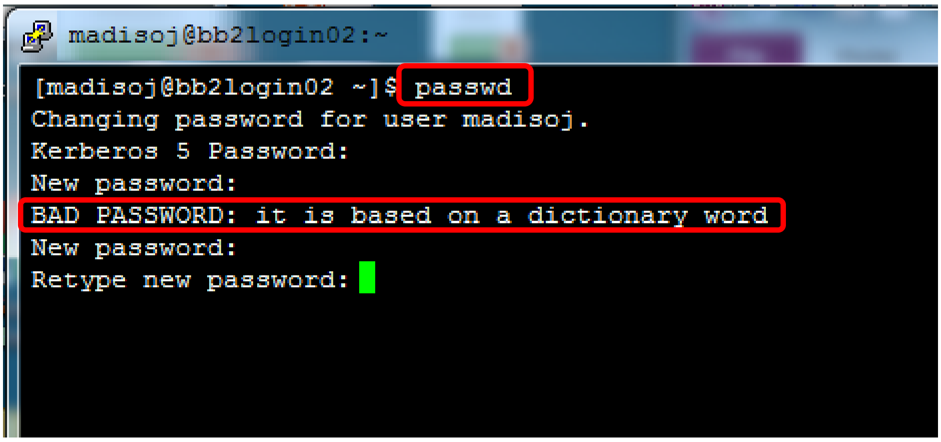
If you make a mistake typing the password, you will receive a message stating that your login is incorrect. At this point you can try your credentials again. In order to log out of the UNIX system, type the "***exit"*** command.



```
madisoj@bb2login02:~
[madisoj@bb2login02 ~]$ exit
```

Changing Passwords

There are times when the system will ask you to change your password when you log in. On many systems, this is done when you first log in and at regular intervals afterwards. You will be prompted for your old password, then your new password twice. You can change your password at any time by using the "***passwd***" command.



```
madisoj@bb2login02:~
[madisoj@bb2login02 ~]$ passwd
Changing password for user madisoj.
Kerberos 5 Password:
New password:
BAD PASSWORD: it is based on a dictionary word
New password:
Retype new password:
```

If you mistype your new password, you will receive the BAD PASSWORD: it is based on a dictionary word message. You will have the opportunity to try again. The passwords will not be displayed on the screen.

Part 2 Directory Structure/Hierarchy

Files and Filenames

A file is a collection of information with a name.

All information on UNIX systems is stored in files. Typical examples that might be kept in files are: computer programs, experiment results, address lists and documents such as this one.

Ideally filenames should allow you to identify the content of the file without opening it. Think about using version information, and the ordering of the elements within a filename, e.g. YYYMMDD allows chronological ordering of the files. An example would be the following:

FILENAME-YYMMDD-CATEGORY-VnMn-STATUS.EXT

Where:

FILENAME	The name or title, preferably without spaces or special characters	Mandatory
YYMMDD	Version or publication date	Highly Recommended
CATEGORY	CONFIDENTIAL, RESTRICTED, or OPEN	Mandatory for confidential items, use capital letters for clarity
VnMm	Version (n) and Modification (m) should be numerical	Optional
STATUS	'Draft', 'Published', etc.	Optional
EXT	File extension	Dependent upon the Operating System

Table 1. File Naming Conventions[1]

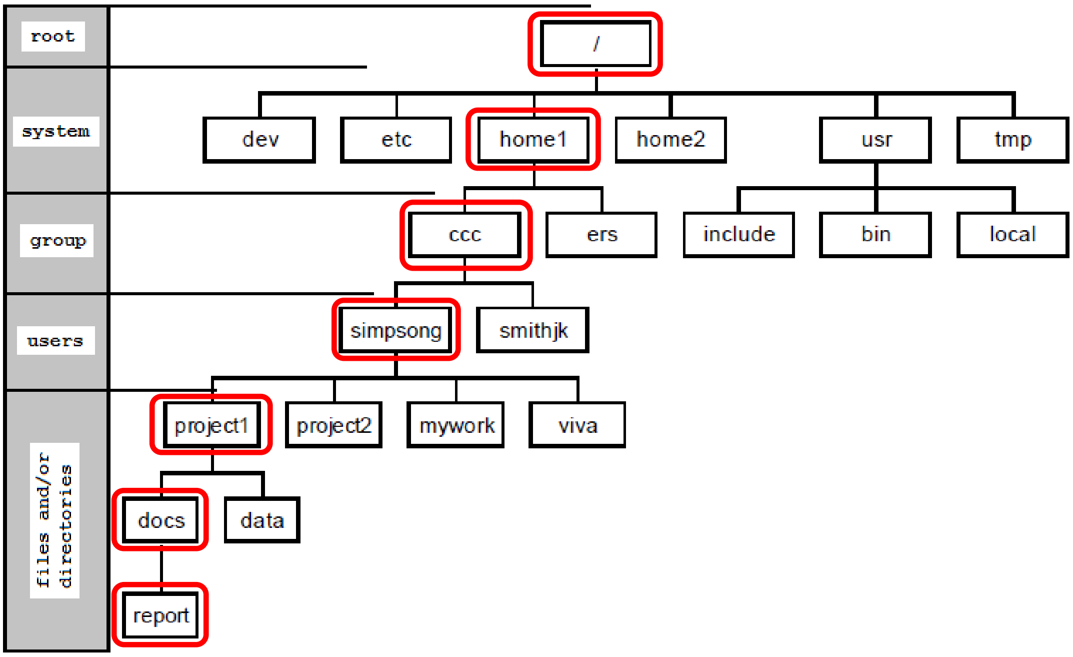
Directories

A UNIX system can contain thousands of stored files. For convenience, these files are stored together in names groups called directories. These directories are organized in what is known as the tree structure (or hierarchical structure). This begins from a single directory (root) which may contain both files and other directories. Each directory may contain additional directories.

Directories have names which follow the same rules as files. The only exception is the root directory which can be referred to by the '/' (forward slash) character.

All directories may contain both files and other directories.

[1] Deighton, D., Guidelines – File Naming Convention Published, https://intranet.birmingham.ac.uk/it/documents/public/Guidelines-File-Naming-Convention.pdf, retrieved 22/10/2015



Pathnames

If you need to refer to a file in a directory other than your current working directory, you will need to describe its position in the file hierarchy in addition to giving its name. This is called the "full path name", or "absolute path name" of the file.

A forward slash "**/**" is used both to indicate the root directory and separate the names of the directories in the path name; the context always makes it clear which meaning is intended. An example of the full path name of the file 'report' can be seen below (it is also shown in red):

/home1/ccc/simpsong/project1/docs/report

In some cases, it may be easier to give the "relative pathname". This begins from the current working directory instead of the root. For example, if the current working directory is:

/home1/cc/simpsong

And you wish to refer to the file 'report' in the above example, the relative pathname would be:

project1/docs/report

Any path name with starts with a "**/**" is a full path name, beginning at the root of the file system. Any other path name is a relative path name and begins from the current working directory. A name which does not contain a "**/**" such as report will always refer to a file in the current working directory.

There are two abbreviations which can be used in path names:

.	A single dot means your current working directory
..	Two dots mean the next directory up the path towards root. This is called the parent directory

Symbolic Links

There is one other type of file which you may need to be aware of. This is the "symbolic link". This contains no data other than a "pointer" to another file or directory in the system.

These have a variety of uses. When their name is used as an argument to a command, the command normally operates on the file or directory being "pointed to" by the link.

Access Control

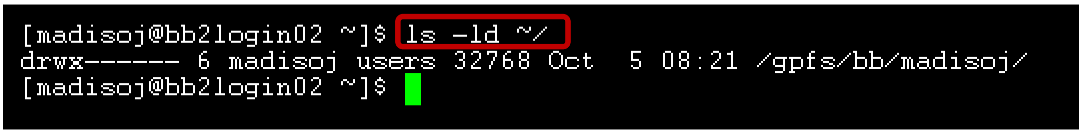
Every file on the system can be protected from, or made accessible to other users. Access controls are implemented using file modes (sets of access permissions) which are associated with files.

Directory Permissions	File Permissions
r (list contents of directory)	**r** (read the file)
w (Create new files/folders)	**w** (write to the file)
x (traverse, e.g. cd to the directory)	**x** (execute or run)

These access controls apply to three classes of user (owner, group, and user) and may be set differently for each. Groups are setup by the system administrator for the purpose of such selective sharing of files.

u	User/owner
g	Group owner
o	All other users
a	All, user/owner, group owner, and all others

To show permissions on the home directory, type: ***ls –ld ~/***

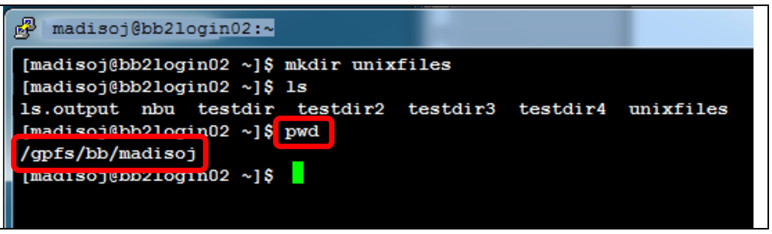


```
[madisoj@bb2login02 ~]$ ls -ld ~/
drwx------ 6 madisoj users 32768 Oct  5 08:21 /gpfs/bb/madisoj/
[madisoj@bb2login02 ~]$
```

Part 3: First Commands

pwd – print working directory or present working directory

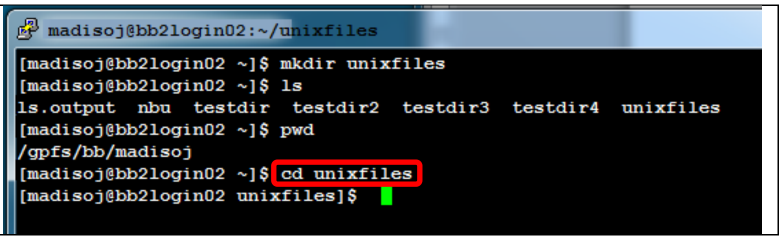
The ***pwd*** command displays your working directory.

pwd	print working directory or present working directory	

```
[madisoj@bb2login02 ~]$ mkdir unixfiles
[madisoj@bb2login02 ~]$ ls
ls.output  nbu  testdir  testdir2  testdir3  testdir4  unixfiles
[madisoj@bb2login02 ~]$ pwd
/gpfs/bb/madisoj
[madisoj@bb2login02 ~]$
```

cd – change working directory

To change your working directory, use the ***cd*** (change directory) command. This can be referred to as "moving" to another directory.

cd	change directory	

```
[madisoj@bb2login02 ~]$ mkdir unixfiles
[madisoj@bb2login02 ~]$ ls
ls.output  nbu  testdir  testdir2  testdir3  testdir4  unixfiles
[madisoj@bb2login02 ~]$ pwd
/gpfs/bb/madisoj
[madisoj@bb2login02 ~]$ cd unixfiles
[madisoj@bb2login02 unixfiles]$
```

To move to the unixfiles directory, type ***cd unixfiles*** instead of using the full path name:

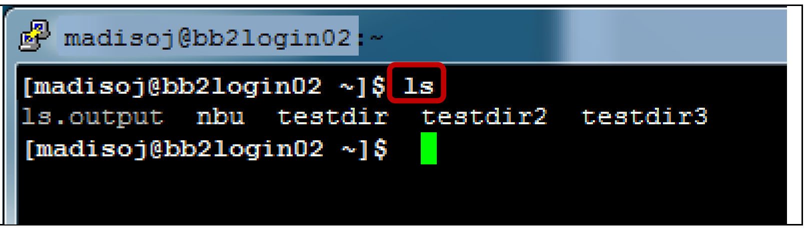
cd /gpfs/bb/madisoj/unixfiles

cd "shortcuts"

cd..	In order to move to a "parent" directory – i.e. the directory above your working directory
cd	If no path name is specified, your home directory is assumed. You can get directly back to your home directory by typing cd alone.
cd ../demo0	To move to another directory owned by your parent directory, combine '..' with the name of the directory in order to avoid typing the full pathname.
cd ../..	To move up the directory structure by more than one level.
cd -	To return to your previous working directory.

ls - List files

To list the files located within a directory, use the ***ls*** command. Optionally, this command will accept a pathname which may be absolute or relative. This will cause the command to list the contents of the directory specified by the path name. If the path name refers to a file and not a directory, the details of the file are displayed.

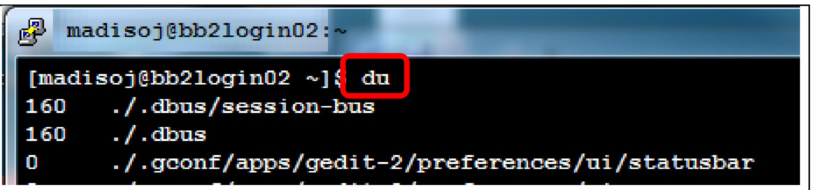
ls	Displays a list of files in a directory	
		`madisoj@bb2login02:~`
		`[madisoj@bb2login02 ~]$ ls`
		`ls.output  nbu  testdir  testdir2  testdir3`
		`[madisoj@bb2login02 ~]$`

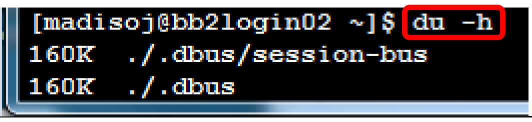
ls "shortcuts"

ls	Used without any arguments, this command lists the names of files, directories, and links in a directory, except hidden ones (those whose names begin with a dot).
ls -a	Using the ***-a*** (all) option will list hidden files.
ls -l	To obtain more information than just the list, the ***-l*** (long) option can be used. The first character indicates whether the file is a directory (d) or symbolic link (l), the next nine characters indicate the mode (access permissions) for the owner, group, and others. This is followed by the number of links to the file, owner, group, size, time of last modification, and finally the name.
ls -F	To list the names in the directory with a character to denote whether they are executable (*), a directory (/) or a link (@).
ls -m	To list files in a compact form.
ls -l mushroom	To list the details of the file called mushroom.
ls -l directory1	To list the contents of directory1.
ls -ld directory1	The ***-d*** option restricts the listing to the directory itself.

du – Disk Usage

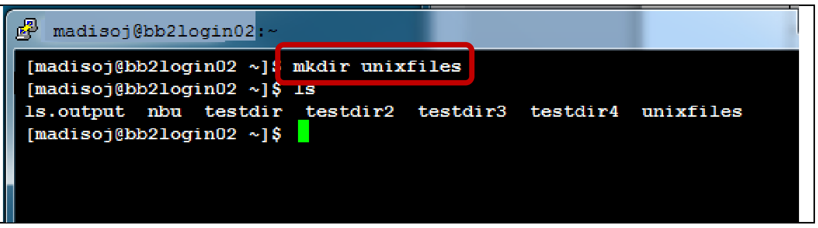
The du command summarises disk usage. By default, this starts from the current directory and lists how name 512K blocks are used in each directory beneath it. A directory can be specified if required. Other options are available. This command is particularly useful if you wish to find out where your filestore is being used when you are close to exceeding your quota. If you wish to display sizes in friendlier quantities, such as Kilobytes and Megabytes, use the ***-h*** option.

du	Summarizes disk usage.	
		`madisoj@bb2login02:~`
		`[madisoj@bb2login02 ~]$ du`
		`160    ./.dbus/session-bus`
		`160    ./.dbus`
		`0      ./.gconf/apps/gedit-2/preferences/ui/statusbar`

du -h	displays sizes in Kilobytes and Megabytes	
		`[madisoj@bb2login02 ~]$ du -h`
		`160K   ./.dbus/session-bus`
		`160K   ./.dbus`

mkdir – Make Directory

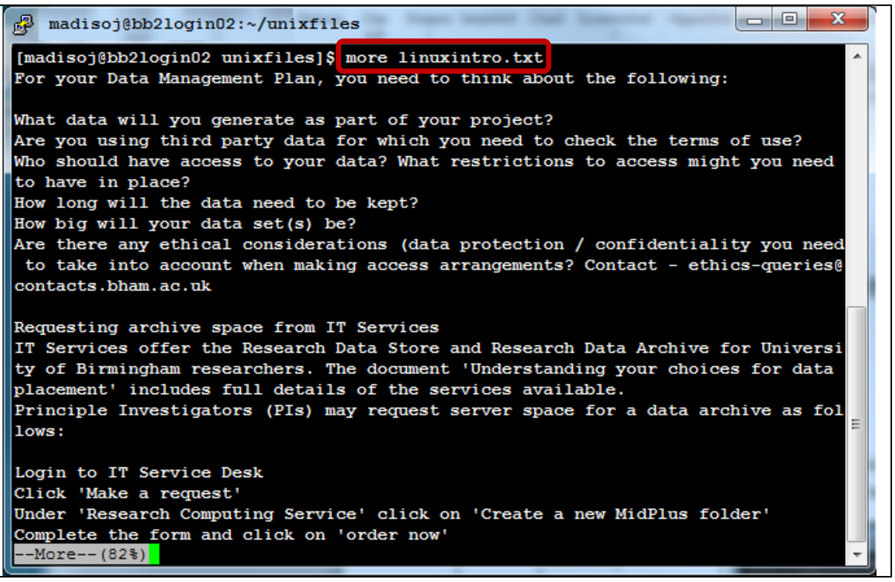
New directories can be created by using the ***mkdir*** command. Remember that this new directory is created within your current working directory. You can use a full pathname to create one elsewhere.

mkdir	Make a new directory	

```
[madisoj@bb2login02 ~]$ mkdir unixfiles
[madisoj@bb2login02 ~]$ ls
ls.output  nbu  testdir  testdir2  testdir3  testdir4  unixfiles
[madisoj@bb2login02 ~]$ 
```

more – view the contents of a file

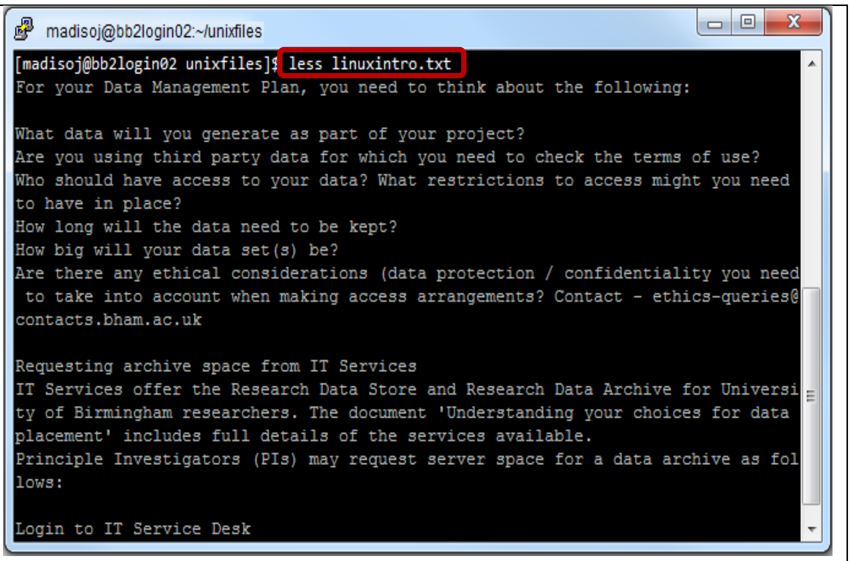
The more command is the easiest way to look at the contents of a file containing simple ASCII text (22 lines on a standard VDU). When you have read the first display, press the space bar to see the next screen of text.

more	View the contents of a file	

```
[madisoj@bb2login02 unixfiles]$ more linuxintro.txt
For your Data Management Plan, you need to think about the following:

What data will you generate as part of your project?
Are you using third party data for which you need to check the terms of use?
Who should have access to your data? What restrictions to access might you need
to have in place?
How long will the data need to be kept?
How big will your data set(s) be?
Are there any ethical considerations (data protection / confidentiality you need
 to take into account when making access arrangements? Contact - ethics-queries@
contacts.bham.ac.uk

Requesting archive space from IT Services
IT Services offer the Research Data Store and Research Data Archive for Universi
ty of Birmingham researchers. The document 'Understanding your choices for data
placement' includes full details of the services available.
Principle Investigators (PIs) may request server space for a data archive as fol
lows:

Login to IT Service Desk
Click 'Make a request'
Under 'Research Computing Service' click on 'Create a new MidPlus folder'
Complete the form and click on 'order now'
--More--(82%)
```

less

Less is similar to more but allows you to scroll backward and forward within the file using your arrow keys. It also allows your to do basic searches within the file, Less does not have to read the entire file so if you have a large file, it will start up faster

less	less writes the contents of a file to the screen a page at a time.	

```
[madisoj@bb2login02 unixfiles]$ less linuxintro.txt
For your Data Management Plan, you need to think about the following:

What data will you generate as part of your project?
Are you using third party data for which you need to check the terms of use?
Who should have access to your data? What restrictions to access might you need
to have in place?
How long will the data need to be kept?
How big will your data set(s) be?
Are there any ethical considerations (data protection / confidentiality you need
 to take into account when making access arrangements? Contact - ethics-queries@
contacts.bham.ac.uk

Requesting archive space from IT Services
IT Services offer the Research Data Store and Research Data Archive for Universi
ty of Birmingham researchers. The document 'Understanding your choices for data
placement' includes full details of the services available.
Principle Investigators (PIs) may request server space for a data archive as fol
lows:

Login to IT Service Desk
```

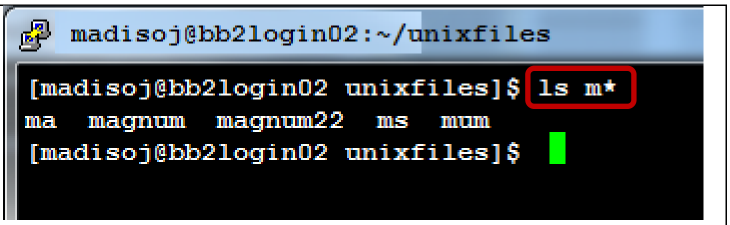
Part 4: Wildcards and a useful feature

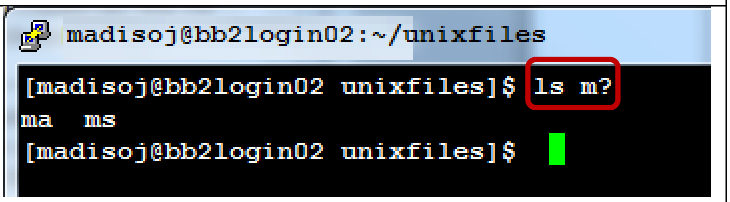
Wildcards

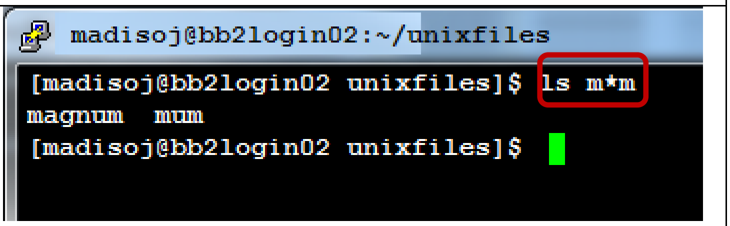
A wildcard is a symbol which can represent any character. The UNIX shell recognises two wildcard symbols:

*****	Stands for zero or more characters
?	Stands for exactly one character

Wildcards are interpreted by the shell and can be used in a variety of ways.

ls m*	Lists all files in the current working directory which begin with the letter m. Note that this will include all files with one or more dots (.) in them.	
		madisoj@bb2login02:~/unixfiles
		`[madisoj@bb2login02 unixfiles]$ ls m*`
		`ma  magnum  magnum22  ms  mum`
		`[madisoj@bb2login02 unixfiles]$`

ls m?	Lists all files in the current working directory which begin with the letter m and consists of two characters	
		madisoj@bb2login02:~/unixfiles
		`[madisoj@bb2login02 unixfiles]$ ls m?`
		`ma  ms`
		`[madisoj@bb2login02 unixfiles]$`

ls m*m	Lists all files in the current working directory which begin and end with the letter m and with one or more characters in between.	
		madisoj@bb2login02:~/unixfiles
		`[madisoj@bb2login02 unixfiles]$ ls m*m`
		`magnum  mum`
		`[madisoj@bb2login02 unixfiles]$`

The ~ symbol (tilde or “squiggle”)

This symbol can be used in the Korn or Bash shell to specify the home directory of another user.

For example: ***cd ~demo0***

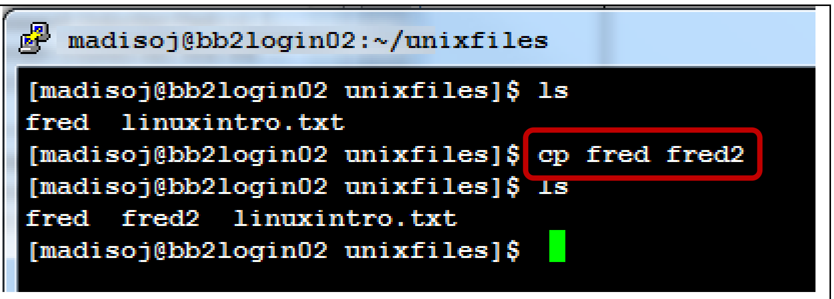
Means move to the home directory of user demo0. This may be useful if you are accessing another user’s files and have been given appropriate access.

Part 5: Copying and Moving Files

cp – copy

The ***cp*** command creates an exact copy of a file. If the new pathname is the name of a directory, the file is copied into that directory with its existing name.

For example, to make a copy of fred called fred 2, type: ***cp fred fred2***

cp	To create an exact copy of a file.	

```
madisoj@bb2login02:~/unixfiles
[madisoj@bb2login02 unixfiles]$ ls
fred  linuxintro.txt
[madisoj@bb2login02 unixfiles]$ cp fred fred2
[madisoj@bb2login02 unixfiles]$ ls
fred  fred2  linuxintro.txt
[madisoj@bb2login02 unixfiles]$ 
```

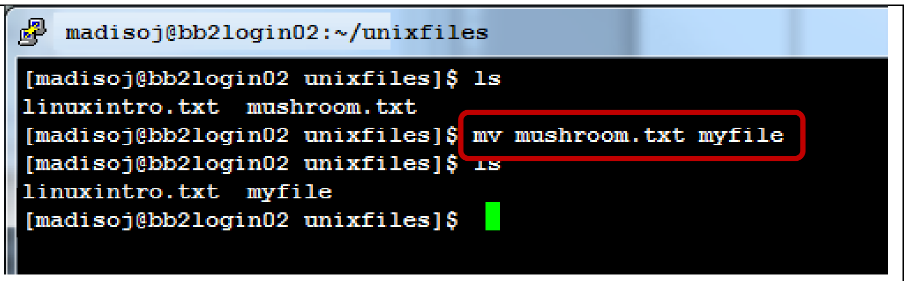
BEWARE: If you choose a new file name that already exists, the existing file will be destroyed and replaced by the copy you are making now.

cp can also be used to copy directories.

mv – move, or rename

To move a file, use the command mv (short for move).

For example: ***mv mushroom.txt myfile***

mv	To move a file or entire directory.	

```
madisoj@bb2login02:~/unixfiles
[madisoj@bb2login02 unixfiles]$ ls
linuxintro.txt  mushroom.txt
[madisoj@bb2login02 unixfiles]$ mv mushroom.txt myfile
[madisoj@bb2login02 unixfiles]$ ls
linuxintro.txt  myfile
[madisoj@bb2login02 unixfiles]$ 
```

BEWARE: If a file called my file already exists, the mv command will overwrite it without warning, losing the contents of the file.

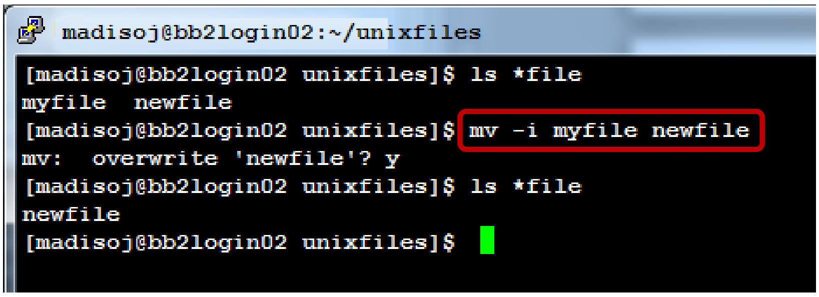
If myfile is in the same directory as mushroom.txt, then it will simply be renamed.

mv can also be used to move entire directories.

As with ***cp***, if the second argument is the name of a directory the file will be moved into that directory keeping its original name.

The ***cp*** and ***mv*** commands can be made to check if the new file already exists by using the -i option. In this case if the new file exists, mv will prompt you with its name and a question mark. Typing anything beginning with ***y*** will allow the move to take place; typing anything else will prevent it.

For example, suppose you have two files, myfile and newfile, and you want to change the name of myfile to newfile:

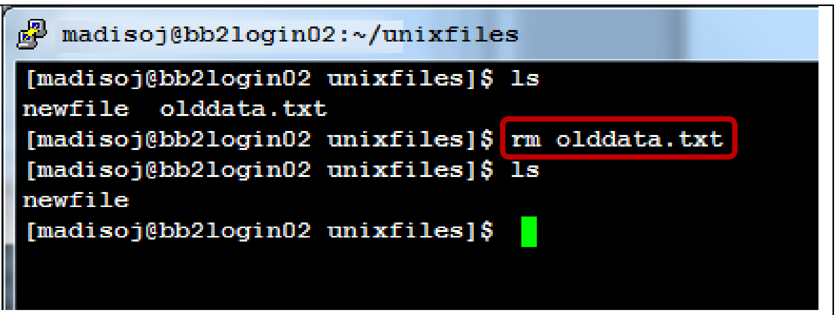


```
[madisoj@bb2login02 unixfiles]$ ls *file
myfile  newfile
[madisoj@bb2login02 unixfiles]$ mv -i myfile newfile
mv:  overwrite 'newfile'? y
[madisoj@bb2login02 unixfiles]$ ls *file
newfile
[madisoj@bb2login02 unixfiles]$
```

rm -remove

To destroy a file, use the command ***rm*** (remove).

For example: ***rm olddata.txt***

rm	To remove a file.	

```
[madisoj@bb2login02 unixfiles]$ ls
newfile  olddata.txt
[madisoj@bb2login02 unixfiles]$ rm olddata.txt
[madisoj@bb2login02 unixfiles]$ ls
newfile
[madisoj@bb2login02 unixfiles]$
```

NOTE: you cannot reverse the effect of this command. Once a file is destroyed, it cannot be retrieved except from a backup tape.

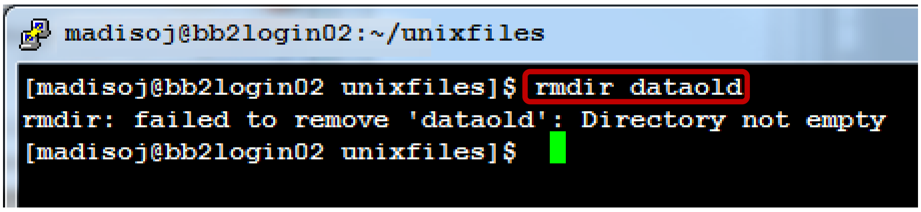
rmdir – remove directory

The ***rmdir*** command removes a directory. It normally requires the directory to be empty.

For example, to remove the directory dataold, you would type:

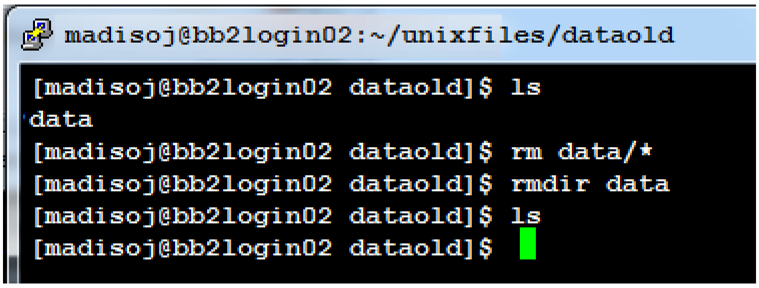
rmdir dataold

If the directory is not empty, an error message is generated.



```
[madisoj@bb2login02 unixfiles]$ rmdir dataold
rmdir: failed to remove 'dataold': Directory not empty
[madisoj@bb2login02 unixfiles]$
```

The files within the directory must be deleted before the directory can be deleted. For example:



madisoj@bb2login02:~/unixfiles/dataold

```
[madisoj@bb2login02 dataold]$ ls
data
[madisoj@bb2login02 dataold]$ rm data/*
[madisoj@bb2login02 dataold]$ rmdir data
[madisoj@bb2login02 dataold]$ ls
[madisoj@bb2login02 dataold]$
```

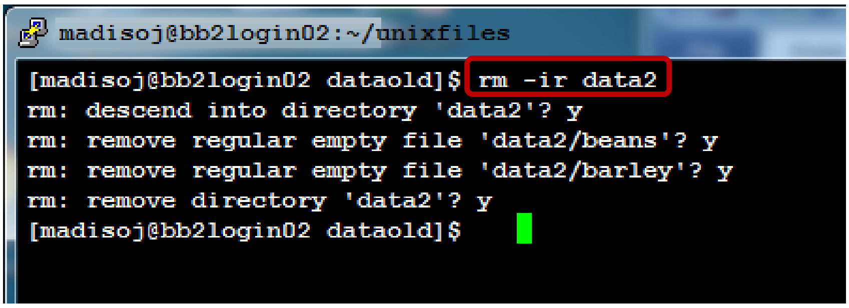
A slightly more drastic command that will delete all files within a named directory before deleting the directory itself is ***rm –r***.

For example: ***rm -r data2***

If given enough access, the ***rm*** command with the ***-r*** option can be used to destroy an entire file system, so use this with caution.

In each case, ***rm*** can be used with the ***-i*** (interactive) option to check if you really want to delete each file before you actually do so.

Thus, running the last example again with the ***-ir*** option:



madisoj@bb2login02:~/unixfiles

```
[madisoj@bb2login02 dataold]$ rm -ir data2
rm: descend into directory 'data2'? y
rm: remove regular empty file 'data2/beans'? y
rm: remove regular empty file 'data2/barley'? y
rm: remove directory 'data2'? y
[madisoj@bb2login02 dataold]$
```

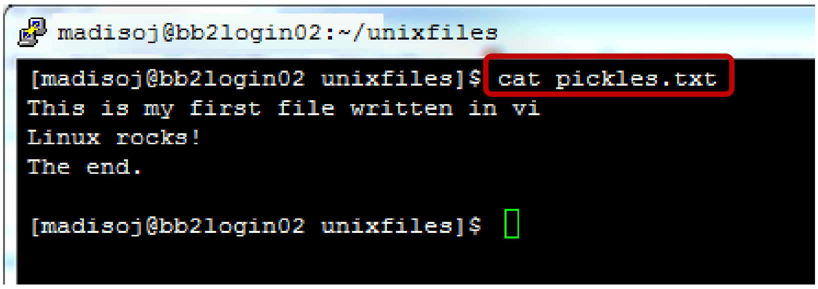
NOTE: The results of the ***-i*** differ slightly between different UNIX implementations. All of them generate queries, but not all versions ask if you want to examine files at the directory level.

Part 6: More about more, cat, and piping

cat - concatenate

The ***cat*** command has the simple function of listing the specified file or files to the standard output, in this case, the screen, e.g.:

cat pickles.txt

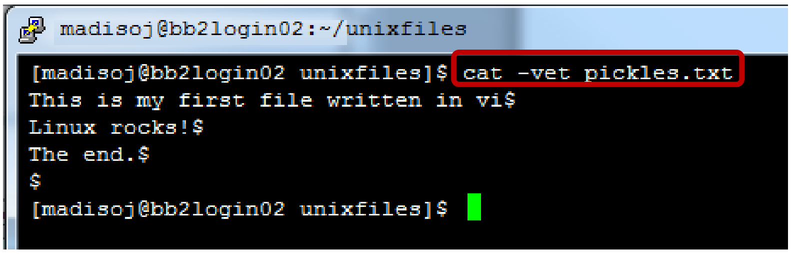


madisoj@bb2login02:~/unixfiles

```
[madisoj@bb2login02 unixfiles]$ cat pickles.txt
This is my first file written in vi
Linux rocks!
The end.

[madisoj@bb2login02 unixfiles]$
```

The ***more*** command is convenient to use for simply displaying files of more than one page, but the cat command is useful in conjunction with other commands. The ***cat*** command has other useful features. A very popular option is to display all characters including nonprintable ones:

cat -vet pickles.txt



madisoj@bb2login02:~/unixfiles

```
[madisoj@bb2login02 unixfiles]$ cat -vet pickles.txt
This is my first file written in vi$
Linux rocks!$
The end.$
$
[madisoj@bb2login02 unixfiles]$
```

Redirecting input and output (>, >>, <)

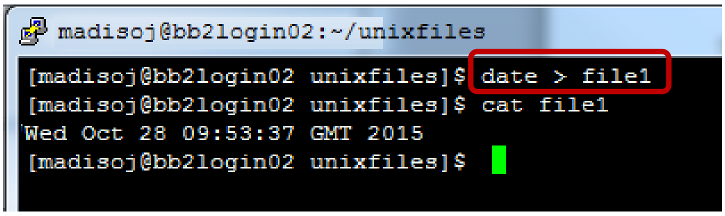
All information stored for you by the system is organised into files. UNIX considers any collection of information to be a file, and also regards every device attached to the computer as a file, even your terminal is a file to UNIX.

Unless you instruct UNIX otherwise, commands will read input from your terminal and will write output to your terminal. By default, your terminal is connected as the “standard input” and the “standard output” files. This action can be changed by redirecting input and output from and to any other file.

The **>** (greater than) character redirects the output to a specific file, e.g.:

date > file1

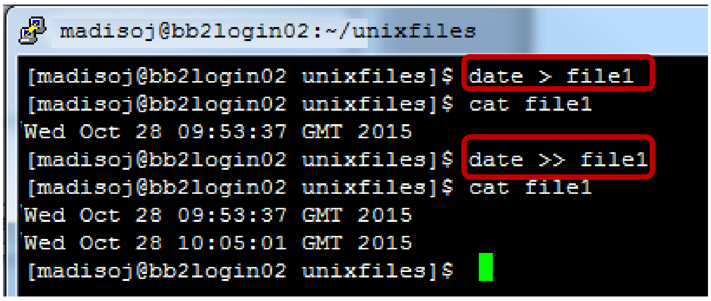
This will write the output from the ***date*** command to file1. If you then use the ***cat*** command to list this file, you would see the following:



```
madisoj@bb2login02:~/unixfiles
[madisoj@bb2login02 unixfiles]$ date > file1
[madisoj@bb2login02 unixfiles]$ cat file1
Wed Oct 28 09:53:37 GMT 2015
[madisoj@bb2login02 unixfiles]$
```

Redirecting output to a file creates either creates a new file with the date information, or if a file already exists with the same name, it will be overwritten with the new information. If you wish to avoid overwriting the file, you can use the **>>** operator (two greater than signs) and the output will be added to the end of the file if it already exists.

>	overwrites
>>	appends



```
madisoj@bb2login02:~/unixfiles
[madisoj@bb2login02 unixfiles]$ date > file1
[madisoj@bb2login02 unixfiles]$ cat file1
Wed Oct 28 09:53:37 GMT 2015
[madisoj@bb2login02 unixfiles]$ date >> file1
[madisoj@bb2login02 unixfiles]$ cat file1
Wed Oct 28 09:53:37 GMT 2015
Wed Oct 28 10:05:01 GMT 2015
[madisoj@bb2login02 unixfiles]$
```

The **<** (less than) character is used to change where the input comes from an example would be to send a file as email to another user using the mail program. In order to send the file pickles.txt to demo0, you could type:

mail demo0 < pickles.txt

The use of email will be covered later in this course. In general, we would not recommend the use of mail, but alternatives such as elm or pine.

Pipelines (|)

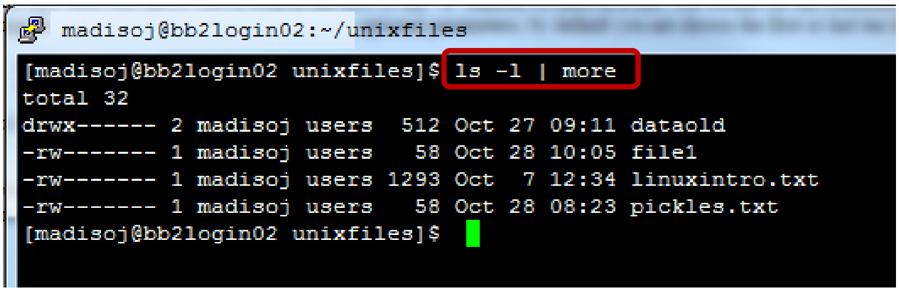
As an extension to the facilities described above, UNIX has a feature called piping which allows you to connect the standard output from a command on the left-hand side of the pipe (the **|** character – this character is often displayed as a broken vertical bar on keyboards) to the standard input of another command on the right-hand side of the pipe. The two commands run concurrently.

For example, the command ***ls -l*** gives a list of the files you have with full details, which is often more than can fit on one screen. The command ***more*** allows text to be displayed one screen at a time.

Thus, if you were to put the list of file names produced by ***ls*** into a file, you could use more to view the output of a page at a time. However, this would require you to create an extra file to contain the list of file names and, to be tidy; you would want to destroy that file again after it was used.

UNIX pipes can often be used to avoid the use of temporary files like this. For example, you can pipe the output from the ***ls*** command directly into the ***more*** command, allowing you to see the output one page at a time.

ls -l | more



```
madisoj@bb2login02:~/unixfiles
[madisoj@bb2login02 unixfiles]$ ls -l | more
total 32
drwx------ 2 madisoj users  512 Oct 27 09:11 dataold
-rw------- 1 madisoj users   58 Oct 28 10:05 file1
-rw------- 1 madisoj users 1293 Oct  7 12:34 linuxintro.txt
-rw------- 1 madisoj users   58 Oct 28 08:23 pickles.txt
[madisoj@bb2login02 unixfiles]$
```

It is worth noting that if the output from ***ls*** is piped, each file normally appears on a separate line whether or not you are using the ***-l*** option, but the ***-m*** option suppresses this feature.

More about more

Previously, we used the more command in a simple way. There are many options to the command. When you have read a page, several commands can be used:

Space bar	Displays the next screen of text
RETURN	Displays the next line of text
q	Exit from more (this can be done at any time)
d	Scroll forward about have a screen of text
b	Skip backwards one screen of text
h	Display a list of commands (help)

Note that the ***b*** option does not work when piping output from another command into ***more***.

man (the manual command)

When you are using UNIX, you will need help from time to time on the commands and facilities available to you. You can consult written documentation or you can use the UNIX on-line help system.

The UNIX help system is based on the UNIX system manuals, which are held on-line. These manuals are generally intended for the more experienced user and especially the programmer, but comprehensive descriptions of all the usual commands are included. This guide only attempts to give you enough information for you to start using the on-line help information. As you become more experienced, you will be able to interrogate the system yourself to find new features.

The command for accessing the manual information is: ***man***

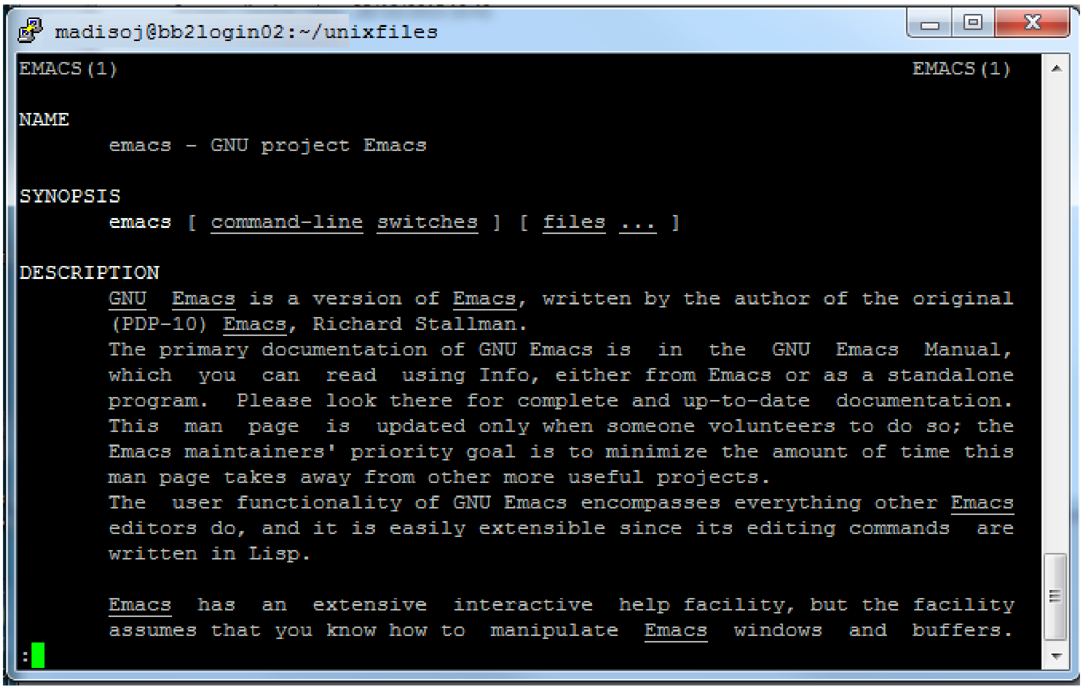
This is short for manual which gives you the page of the UNIX manual describing a particular command.

In addition, some UNIX commands give a very brief summary of their intended usage when you type the command name without any arguments or with erroneous arguments. This facility should be used with caution unless you know what you are doing since just typing commands at random can have unpredictable effects.

To get help about any particular command, you should type ***man*** followed by the command name.

For example, to get help about the GNU Emacs editor, type:

man emacs



```
EMACS(1)                                                          EMACS(1)

NAME
       emacs - GNU project Emacs

SYNOPSIS
       emacs [ command-line switches ] [ files ... ]

DESCRIPTION
       GNU  Emacs is a version of Emacs, written by the author of the original
       (PDP-10) Emacs, Richard Stallman.
       The primary documentation of GNU Emacs is  in  the  GNU  Emacs  Manual,
       which  you  can  read  using Info, either from Emacs or as a standalone
       program.  Please look there for complete and up-to-date  documentation.
       This  man  page  is  updated only when someone volunteers to do so; the
       Emacs maintainers' priority goal is to minimize the amount of time this
       man page takes away from other more useful projects.
       The  user functionality of GNU Emacs encompasses everything other Emacs
       editors do, and it is easily extensible since its editing commands  are
       written in Lisp.

       Emacs  has  an  extensive  interactive  help facility, but the facility
       assumes that you know how to  manipulate  Emacs  windows  and  buffers.
:
```

You will see a screen of information (22 lines) about the command emacs. At this point, you can use the options of the ***more*** command to proceed.

Each command has a one line summary. These can be searched using keywords. For example:

man -k copy

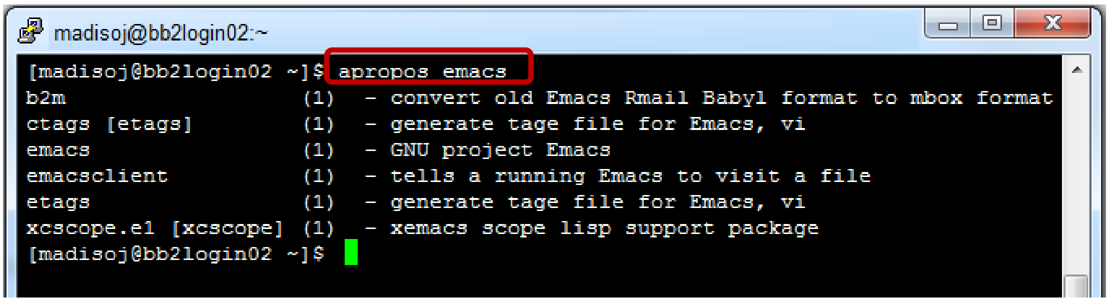
This will list all commands which have the word ‘copy’ in their one line summary. This can be useful if you are unsure about the name of the desired command.

apropos

Apropos is a command that can be used when searching for commands when you aren’t exactly sure of their names. For example, by typing:

apropos emacs

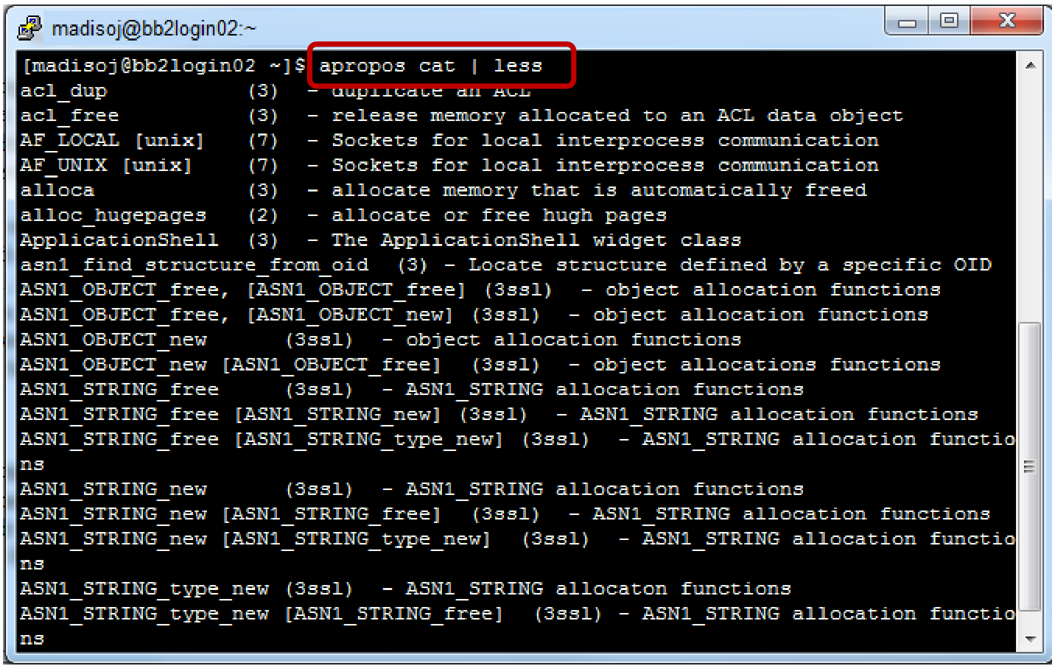
a single line is returned for every instance of emacs that is found in each man page. Apropos is not case sensitive which makes searching easier.



```
[madisoj@bb2login02 ~]$ apropos emacs
b2m                    (1)  - convert old Emacs Rmail Babyl format to mbox format
ctags [etags]          (1)  - generate tage file for Emacs, vi
emacs                  (1)  - GNU project Emacs
emacsclient            (1)  - tells a running Emacs to visit a file
etags                  (1)  - generate tage file for Emacs, vi
xcscope.el [xcscope]   (1)  - xemacs scope lisp support package
[madisoj@bb2login02 ~]$
```

apropos cat | less

would return a window similar to the one below:



```
[madisoj@bb2login02 ~]$ apropos cat | less
acl_dup            (3)  - duplicate an ACL
acl_free           (3)  - release memory allocated to an ACL data object
AF_LOCAL [unix]    (7)  - Sockets for local interprocess communication
AF_UNIX [unix]     (7)  - Sockets for local interprocess communication
alloca             (3)  - allocate memory that is automatically freed
alloc_hugepages    (2)  - allocate or free hugh pages
ApplicationShell   (3)  - The ApplicationShell widget class
asn1_find_structure_from_oid  (3) - Locate structure defined by a specific OID
ASN1_OBJECT_free, [ASN1_OBJECT_free] (3ssl)  - object allocation functions
ASN1_OBJECT_free, [ASN1_OBJECT_new] (3ssl)  - object allocation functions
ASN1_OBJECT_new       (3ssl)  - object allocation functions
ASN1_OBJECT_new [ASN1_OBJECT_free]  (3ssl)  - object allocations functions
ASN1_STRING_free      (3ssl)  - ASN1_STRING allocation functions
ASN1_STRING_free [ASN1_STRING_new] (3ssl)  - ASN1_STRING allocation functions
ASN1_STRING_free [ASN1_STRING_type_new] (3ssl)  - ASN1_STRING allocation functio
ns
ASN1_STRING_new       (3ssl)  - ASN1_STRING allocation functions
ASN1_STRING_new [ASN1_STRING_free]  (3ssl)  - ASN1_STRING allocation functions
ASN1_STRING_new [ASN1_STRING_type_new]  (3ssl)  - ASN1_STRING allocation functio
ns
ASN1_STRING_type_new (3ssl)  - ASN1_STRING allocaton functions
ASN1_STRING_type_new [ASN1_STRING_free]  (3ssl) - ASN1_STRING allocation functio
ns
```

Part 7: Shells

Upon logging into UNIX, you are presented with a prompt, inviting you to issue commands. The prompt is issued by a “shell”. Shells are programs which allow you to communicate with the UNIX system. Shells read your input from the command line, interprets the input, and submits your commands to the system so that they can be run. Commands are run as separate processes that are initiated by your shell.

In addition to allowing you to input commands, shells are programming languages. Files containing commands can be written and run just as if they were programs. In addition, shells support variables, parameter passing, and a variety of programming control structures (loops, if-the-else, etc.), allowing sophisticated programs to be written. However, many of these features will not be of interest to the basic user.

There are numerous different shells which can be used in UNIX systems. The most common are the Bourne shell (sh), Bourne again shell (bash), the C-shell (csh) and the Korn shell (ksh).

Traditionally, the Bourne shell was considered to be good for programming and the C-shell for interactive use. The Korn shell is intended to provide the best environment for both types of use.

The Korn shell is very rich in features and arguably the most “user friendly”. There are many other shells available from a variety of sources, the Bourne-again shell (bash), the Korn and Bash of the shells have a lot of features in common from the end user’s perspective. Bash is the most commonly used shell in Linux.

By default, IT Services Sun servers offer users the Korn shell (ksh), and the Korn shell will be covered in this course. BlueBEAR uses bash. If you wish to use a different shell from the one allocated at login, you may do so by giving the name of the shell you wish to use as a command.

Sometimes, it is difficult to tell which functions are provided by the shell and which are provided by UNIX. Most of the time it doesn’t matter to the users; however, in some cases it is worth being aware, particularly when problems arise and documentation needs to be consulted. Some of the shell features are documented as part of the shell, so in order to find information about these, you would type:

```
man ksh
```

The following sections describe some shell features in more detail.

Wildcards and Special Characters

Wildcards (***,?**) and other special characters (**|><**) are interpreted by the shell and not the commands themselves. These are documented as part of the shells.

Built-in Commands

Some commands are provided by the shells and not the UNIX system. These are known as built-in commands and functions. The most commonly used is: ***cd***

This is documented on its own manual page.

Other commands, such as ***set*** and ***read***, are documented with other shell features.

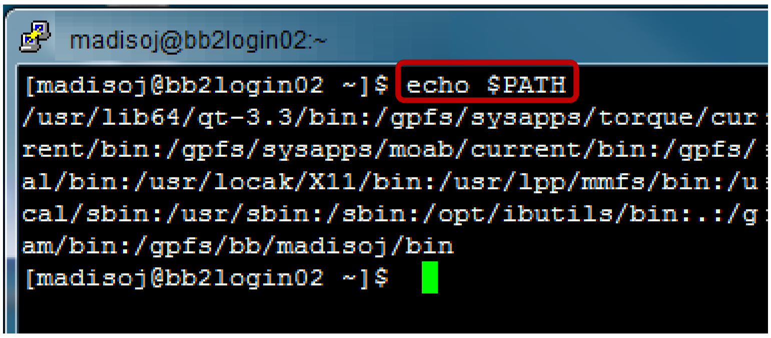
Variables and Environmental Variables

As has been stated earlier, shells support variables. The variables of most interest are the environment variables. These can be displayed by giving the following command: ***env***

The variables contain information about terminal type, preferred editor, name and a variety of other attributes.

Individual variables can be accessed by using the echo command. When accessing the value of a variable, a **$** is added to the name of the variable. For example, to see the value of PATH, you can type:

echo $PATH

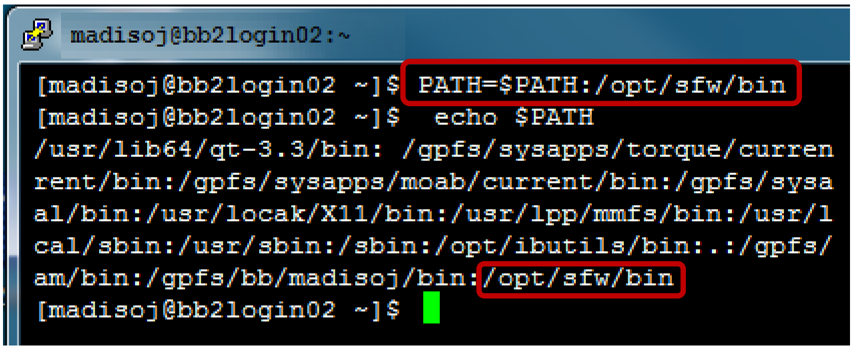


```
[madisoj@bb2login02 ~]$ echo $PATH
/usr/lib64/qt-3.3/bin:/gpfs/sysapps/torque/cur
rent/bin:/gpfs/sysapps/moab/current/bin:/gpfs/
al/bin:/usr/locak/X11/bin:/usr/lpp/mmfs/bin:/u
cal/sbin:/usr/sbin:/sbin:/opt/ibutils/bin:.:/g
am/bin:/gpfs/bb/madisoj/bin
[madisoj@bb2login02 ~]$
```

Each time you type a command that is not built into the shell, all the directories in the PATH variable are searched and the first matching program is used. You may sometimes need to change some of the variables. In the Korn and BASH shells, this can be achieved by assigning a new value. This can replace the current value or you can use the existing value of a variable if you wish to append to it. For example, in order to add the directory /opt/sfw/bin to PATH, you would type the following:

PATH=$PATH: /opt/sfw/bin

You can check what this has done by using the ***echo*** command.



```
[madisoj@bb2login02 ~]$ PATH=$PATH:/opt/sfw/bin
[madisoj@bb2login02 ~]$  echo $PATH
/usr/lib64/qt-3.3/bin: /gpfs/sysapps/torque/curren
rent/bin:/gpfs/sysapps/moab/current/bin:/gpfs/sysa
al/bin:/usr/locak/X11/bin:/usr/lpp/mmfs/bin:/usr/l
cal/sbin:/usr/sbin:/sbin:/opt/ibutils/bin:.:/gpfs/
am/bin:/gpfs/bb/madisoj/bin:/opt/sfw/bin
[madisoj@bb2login02 ~]$
```

Environment variables are available to commands and processes initiated by your shell. Not all variables are available in this way by default. In order to make sure that the variable is available to other processes started by your shell, the export command is used. For example:

export TERM=vt100

numeric variables

Numeric integers can be used in Korn and Bash, however, if you wish to perform arithmetic operations on them, they must be declared as such first. This can be done using the ***typeset*** command. For example, if we wanted to set a variable called ***x*** to be used as an integer, we would type:

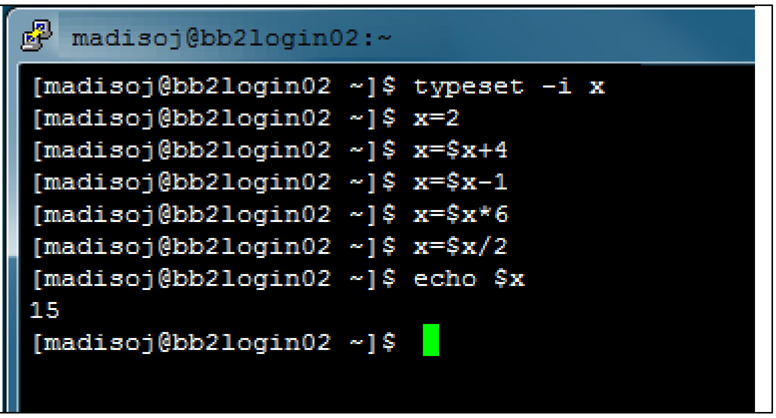
typeset -i x

In Korn shell, we could alternatively use the ***integer*** command which some people find easier to remember

integer x

Having set the variable up as a number, we can perform simple arithmetic operations. For example:

```
x=2
x=$x+4
x=$x-1
x=$x*6
x=$x/2
echo $x
```



```
[madisoj@bb2login02 ~]$ typeset -i x
[madisoj@bb2login02 ~]$ x=2
[madisoj@bb2login02 ~]$ x=$x+4
[madisoj@bb2login02 ~]$ x=$x-1
[madisoj@bb2login02 ~]$ x=$x*6
[madisoj@bb2login02 ~]$ x=$x/2
[madisoj@bb2login02 ~]$ echo $x
15
[madisoj@bb2login02 ~]$
```

You can check the results with the ***echo*** command. The above should give a value of 15.

Part 9: Text editors

Many system applications require you to input text (e.g., email, news). Rather than force you to use a potentially different method for each application, most such applications on UNIX allow a choice of text editor. This can be selected on a per user basis. The disadvantage to this approach is that it is necessary to learn to use some form of text editor. However, the basics of most text editors are fairly simple.

Text editors create basic text files. They are similar in respect to word processors, but traditionally differ in that text editors do not handle different fonts or other advanced formatting features. However modern versions of some text editors give the functionality of a good word processor.

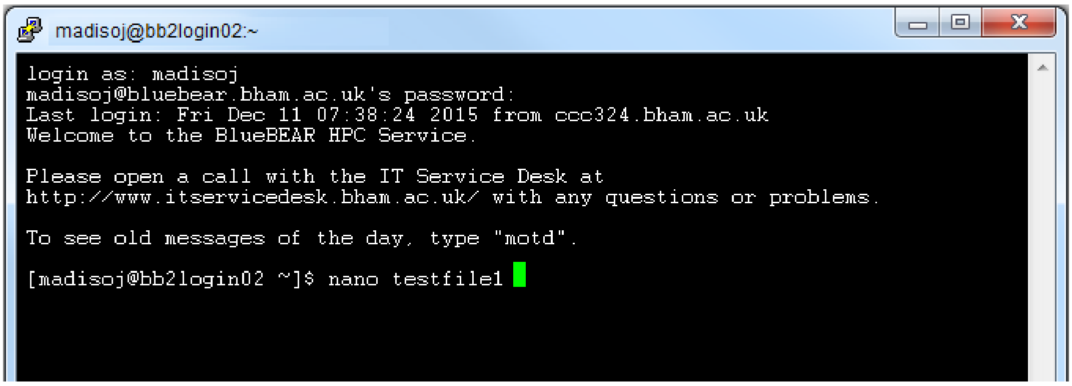
Two text editors that we recommend are Nano and Emacs.

Nano

Nano is a versatile, easy-to-use text editor that can be used with little training.

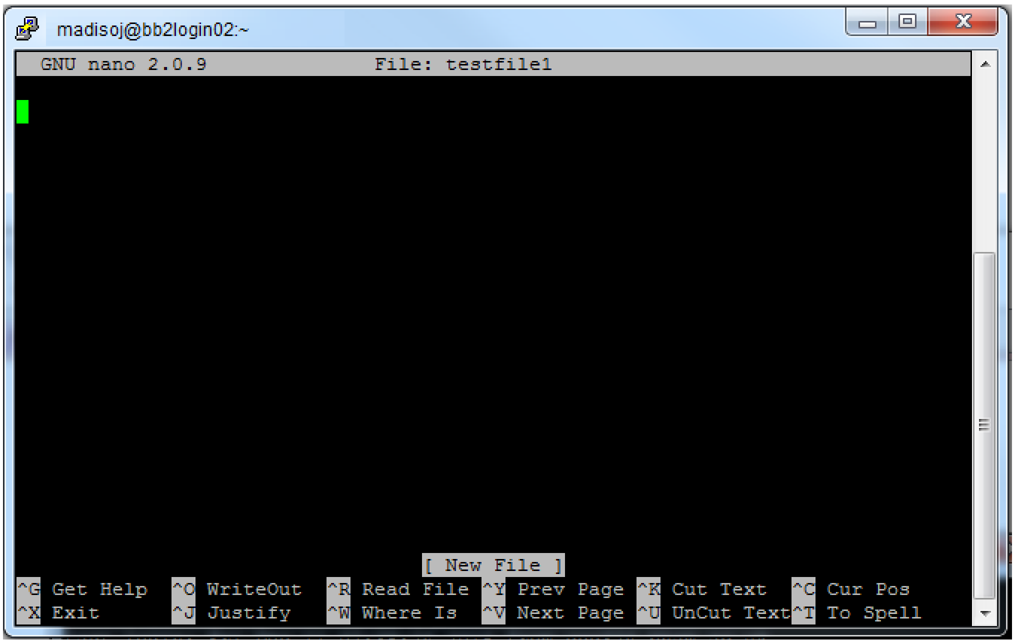
To open a new file called testfile1, type:

nano testfile1



```
login as: madisoj
madisoj@bluebear.bham.ac.uk's password:
Last login: Fri Dec 11 07:38:24 2015 from ccc324.bham.ac.uk
Welcome to the BlueBEAR HPC Service.

Please open a call with the IT Service Desk at
http://www.itservicedesk.bham.ac.uk/ with any questions or problems.

To see old messages of the day, type "motd".

[madisoj@bb2login02 ~]$ nano testfile1
```

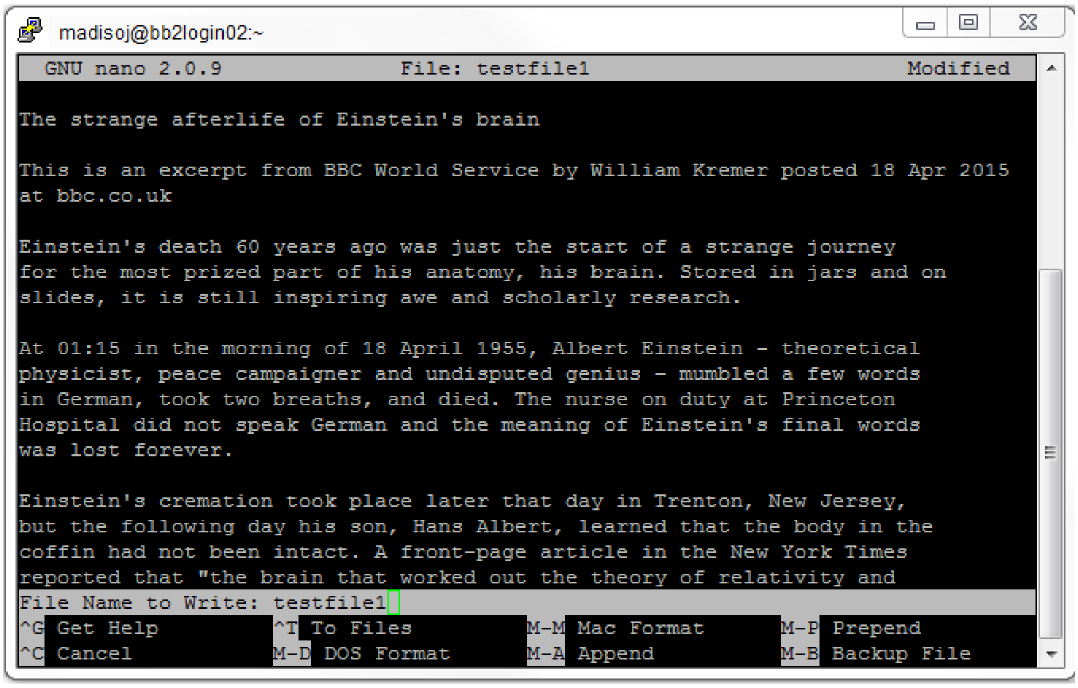
The default window will look something like this:



```
  GNU nano 2.0.9                File: testfile1




                              [ New File ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Ctrl + O (WriteOut)

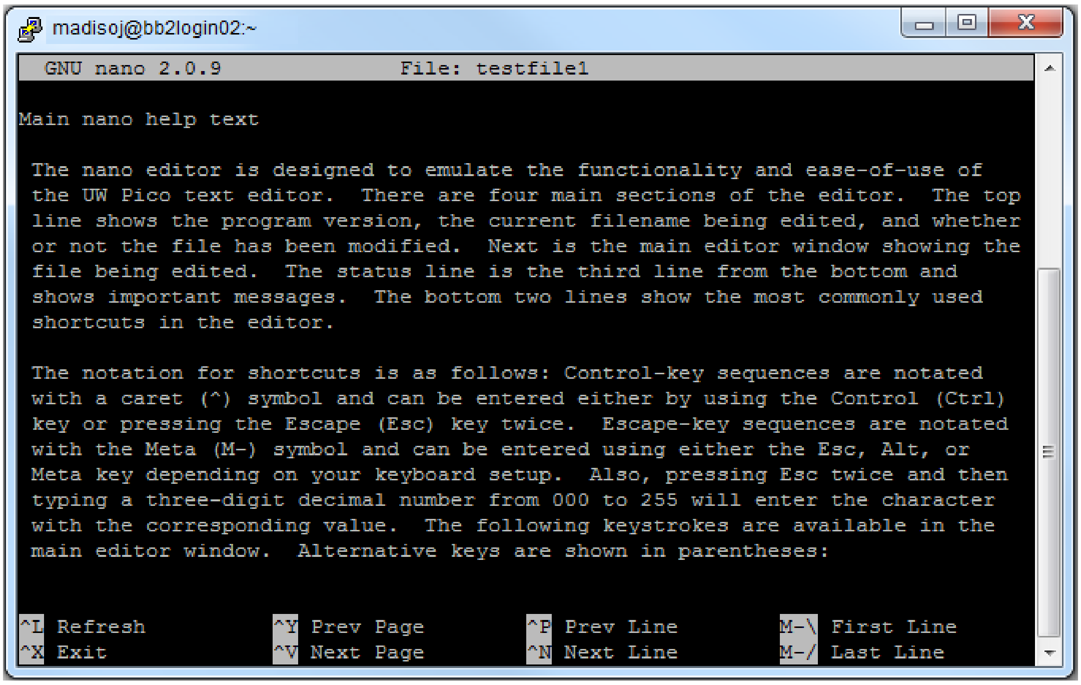
After you have typed your document, Ctrl + O will allow you to Save or WriteOut and continue working.



```
  GNU nano 2.0.9                File: testfile1                       Modified

The strange afterlife of Einstein's brain

This is an excerpt from BBC World Service by William Kremer posted 18 Apr 2015
at bbc.co.uk

Einstein's death 60 years ago was just the start of a strange journey
for the most prized part of his anatomy, his brain. Stored in jars and on
slides, it is still inspiring awe and scholarly research.

At 01:15 in the morning of 18 April 1955, Albert Einstein - theoretical
physicist, peace campaigner and undisputed genius - mumbled a few words
in German, took two breaths, and died. The nurse on duty at Princeton
Hospital did not speak German and the meaning of Einstein's final words
was lost forever.

Einstein's cremation took place later that day in Trenton, New Jersey,
but the following day his son, Hans Albert, learned that the body in the
coffin had not been intact. A front-page article in the New York Times
reported that "the brain that worked out the theory of relativity and
File Name to Write: testfile1
^G Get Help            ^T To Files            M-M Mac Format         M-P Prepend
^C Cancel              M-D DOS Format         M-A Append             M-B Backup File
```

Ctrl + G (Help)

Ctrl + G will open the Help documentation. Scroll down with the arrow key to see more information.



```
  GNU nano 2.0.9                File: testfile1

Main nano help text

 The nano editor is designed to emulate the functionality and ease-of-use of
 the UW Pico text editor.  There are four main sections of the editor.  The top
 line shows the program version, the current filename being edited, and whether
 or not the file has been modified.  Next is the main editor window showing the
 file being edited.  The status line is the third line from the bottom and
 shows important messages.  The bottom two lines show the most commonly used
 shortcuts in the editor.

 The notation for shortcuts is as follows: Control-key sequences are notated
 with a caret (^) symbol and can be entered either by using the Control (Ctrl)
 key or pressing the Escape (Esc) key twice.  Escape-key sequences are notated
 with the Meta (M-) symbol and can be entered using either the Esc, Alt, or
 Meta key depending on your keyboard setup.  Also, pressing Esc twice and then
 typing a three-digit decimal number from 000 to 255 will enter the character
 with the corresponding value.  The following keystrokes are available in the
 main editor window.  Alternative keys are shown in parentheses:


^L Refresh           ^Y Prev Page         ^P Prev Line         M-\ First Line
^X Exit              ^V Next Page         ^N Next Line         M-/ Last Line
```

To close Help, press Ctrl + X.

List of commands

Ctrl + E	Go to the end of the current working line.
Ctrl + A	Go to the beginning of your current working line.
Ctrl + W	Search your text.
Ctrl + T	Check the spelling of your text.
Ctrl + U	Uncut (or Paste) text
Ctrl + K	Cut text.
Ctrl + C	Display the current cursor position
Ctrl + R	Read a file into your current working file. This enables you to add text from another file while working from within a new file.
Ctrl + X	Exits the editor. If you are in the middle of editing a file the exit process will ask you if you want to save your work.

Emacs

Emacs is widely available on many systems (e.g., UNIX, VMS, DOS, Windows, Mac, Amiga), and is generally considered to be easier to use than vi, which is the standard UNIX text editor.

Overview of Emacs

Emacs has often been described as the most powerful text editor available. It has many features which can be very useful in the editing, manipulating, and reformatting of text files. These files can contain anything and could be part of an email message or a large amount of data for processing by an application program.

There is not enough time in this course to cover every feature of using such a flexible and comprehensive editor, but there should be time to learn how to create and modify a simple file, insert an existing file and try out a few of the other features. There will be an opportunity to use the built-in emacs tutorial.

Buffers

Emacs works with the concept of “buffers”. Buffers are areas which hold the text being edited. Emacs has a capability of working with more than one buffer at a time, which allows editing of multiple files at once and transferring text between different files. Most of the time you will be working with a single buffer, as you will be editing one file at a time.

Text input and command input

Normally, any characters typed into emacs will appear in the current buffer, so will be added to the file being created or modified. However, it is sometimes necessary to issue a command to emacs in order to save a file or search for a sequence of characters, for example. In order for emacs to recognise the commands as separate from the text, these commands are often either control characters or start with control characters. In addition, some are prefixed by the escape key, which is referred to as “meta” in emacs documentation.

In X-windows and Windows versions of emacs, some of these functions are available as buttons or pull down menus.

Creating a file

In order to create a new file called testfile.txt, type

```
emacs testfile.txt
```

You will be presented with a screen which is blank, apart from the highlighted line towards the bottom of the screen which gives basic information about the file being edited.

At this point, you can type in any text you wish. Before the end of every line, it is necessary to use the Enter key. Failure to do this results in very long lines. These are difficult to read and to edit. The Delete key can be used to correct any mistakes.

Once you have finished typing, you need to save the file by typing Ctrl + X followed by Ctrl + S. On some terminal emulations, the Ctrl + S character can cause problems, so you may need to use the "Save As" option. This can be achieved by typing Ctrl + X followed by Ctrl + W. At this point, the cursor moves to the bottom of the screen and prompts with the following:

write file: ~/

If you press Enter at this point, the file will save to whatever filename you used when starting emacs (in this example, testfile.txt). Some systems will ask you to confirm that you want to save the new version of the file, overwriting the existing file. You may give a new filename instead of pressing Enter.

Having saved the file, you need to exit from the editor. This is done by typing Ctrl + X followed by Ctrl + C.

Modifying an Existing File

An existing file can be altered in much the same way as creating a new file. For example, to modify the file testfile2.txt, you would issue the following command:

emacs testfile2.txt

The file would then be displayed on the screen, with the status and command lines at the bottom.

The curser (arrow) keys can be used to move about within the file, and the delete key can be used to delete any text. The Enter key can be used to split or add lines as necessary.

Once the modifications are complete, you can save the file and leave emacs in the way described above.

Emacs automatically creates a backup file each time the modified file is saved. This is useful in case major mistakes are made and you need to recover from the last saved copy. The backup file is named using the original filename followed by a tilde (~).

In the example testfile2.txt, the backup file would be called

testfile2.txt~

If you leave emacs without saving the file, emacs gives the opportunity to save any changed files.

Inserting a File

In order to insert a file at any point when editing a file, type Ctrl + X followed by I (i). Emacs then prompts for a pathname, assuming the current working directory.

Insert file: ~/

would be displayed if you are working in your home directory. You may simply type in a filename. For example, to insert the file called letter.txt, you would type letter.txt at the prompt.

Insert file:~/letter.txt

Alternatively, if you are unsure of the filename, you can type in part of the filename and emacs will display a possible list of files. If you wish to type in a complete pathname, delete the /~ if necessary and type in the new pathname.

Accidental use of Features

Emacs is very rich in features. As with all word processors and text editors, it is easy to accidentally activate features that you do not wish to use by accidentally typing in control characters, for example. The following are common pitfalls for the new user.

Suspending Emacs

This can be achieved by typing Ctrl + Z. Using the above file example, the system generates the message

[1] + Stopped (user) emacs testfile2.txt

The command prompt is then displayed, allowing input of UNIX commands. Emacs will still be there, but in the “background”. To restart issue the command

Fg

This puts emacs back in the “foreground” and allows continued use of emacs. No text or alterations will be lost.

Command Mode

Accidental use of Ctrl + X, Esc followed by X and some other sequences will put emacs into a mode which will allow command input. Normal use can be restored by use of Ctrl + G.

Applications which use Emacs

If you are using emacs with a mail system or news reader, for example, there is no need to specify a filename as the system will generate one automatically. Use of emacs to edit and save text is in other respects the same.

Learning more about emacs

Emacs has a tutorial system built in which enables anyone to learn more about using the editor at their own pace. In order to activate the tutorial system, issue the emacs command without a filename, and then type Ctrl + H followed by T. Full instructions are given.