Running vi

To run vi and create a new file, simply run the command 'vi' from any shell prompt:

```
$ vi
```

Alternatively, to load an existing text file into vi, run the command 'vi [filename]', from the shell prompt:

```
$ vi myfile.txt
```

Your file will be loaded into vi, and the cursor will be placed at the beginning of the first line. Note that an empty line is shown as a tilde (~).

Command Mode and Insert Mode

vi is different from many editors, in that it has two main modes of operation: command mode, and insert mode. This is the cause of much of the confusion when a new user is learning vi, but it is actually very simple to understand.

When you first load the editor, you will be placed into command mode. To switch into insert mode, simply press the 'i' key. Although nothing will change on the screen to indicate the new mode, any thing that you type from now on will appear in the screen - this is what you are used to if you have ever used any other editor, or word processor. Try typing a few lines of text. When you press 'return' or 'enter' a new line will be created, and you may continue typing.

When you have finished typing, you may return to command mode. This is done by pressing your 'Esc' key. In command mode, key presses do not appear on the screen, but instead are used to indicate various commands to vi.

At first, you may often mistake command mode and insert mode. For example, you may think you are in insert mode, and start typing your text, when in fact you are in command mode, and each keypress you make will issue a command to vi. Be careful - you may accidentally modify or delete parts of your file.

If you are unsure which mode you are in, press 'Esc'. If you were in insert mode, you will be returned to command mode. If you were already in command mode, you will be left in command mode (possibly with a 'beep', to indicate that you were already in command mode).

An Exercise

Try this simple exercise to get used to the two modes:

1. Load vi (if you haven't already), by typing:

```
$ vi
```

(don't forget - the $ is the system's prompt, and may be different on your system. You only type the 'vi' part, shown in bold.)

The screen should show a blank file, with each blank line represented by a tilde (~), for example:

```
~
~
```

By default you are in command mode.

2. Switch to insert mode, by pressing the 'i' key. Then type some text, using ENTER or RETURN to start new lines. For example:

```
Hello.  This is my first session in the vi editor.
This is the second line of text.
~
~
~
```

3. When you have finished entering your sample text, press 'Esc' to return to command mode.

4. Now we will learn a useful command: to save the file. The command for this is ':w' (note the colon before the 'w'). After the 'w', put a space, and the name you want to store the file as. For example:

```
:w firstfile
```

Type it now. Notice how the text ':w' appears at the bottom on the screen. When you have finished the command, press ENTER or RETURN. You should see a confirmation that the file has been saved, which may include the number of lines in the file, and possibly the file size.

5. To finish this simple introduction to the editor, we will learn one final command: How to close the editor. The command for this is ':q' (again, with a colon before the 'q'). Don't forget you need to be in command mode. If you're not already in command mode, or you're not sure which mode you're in, press 'Esc' now. Then issue the command:

```
:q
```

You should be returned to the UNIX prompt.

Sometimes, you may have made changes to a file that you do not want to save. This may be because you have decided the changes are incorrect, or you have become confused using vi (not usual at first!), and incorrectly made some changes, maybe by typing into command mode instead of insert mode. To exit vi without saving, and ignoring any warnings about unsaved data, use a variation of the ':q' command, with an exclamation mark after it:

:q!

This will return you to the prompt, without saving any changes to the file, and with no warnings about unsaved data. Use this command carefully.

A Simple (but useful) Electronic Diary

As an exercise, create an electronic diary, that you can use to store your schedule, one week at a time. This will use many of the skills you have learned so far, and will let you see how you already know enough to use UNIX to help with every day tasks. Follow these steps:

1. Use vi to create a file named 'weekly-diary-template'. The file should look something like this:

```
My Schedule for Week Commencing:
Day         AM                          PM              Evening
-------------------------------------------------------------
Mon
Tue
Wed
Thur
Fri
Sat
Sun
```

2. Use the 'cp' command to make a new copy of the file, which should be named after the date of the Monday in the current week, for example '12-October-2009'. The command may be something like:

```
$ cp weekly-diary-template diary-16-Nov-2015
```

You should now have two files: the template, and the new copy of the template, which is named after the current week.

3. Use vi to fill in your schedule. Only make changes to the version of the file including the date. Leave the template as it is, ready to create next week's diary when you need it.

4. Whenever you need to start a new week's diary, use a similar 'cp' command. Again, copy the template file to a new file containing the date of the Monday in that week. For example, the following week, you may use:

```
$ cp weekly-diary-template diary-16-Nov-2015
```

Start using the electronic diary on a regular basis. This will give you good practice in vi, and basic file management commands.

Vi Tips

	Topic	Hints
1	The vi operating modes.	**Command mode**. The vi editor opens in command mode awaiting a command. Press the ESC key to enter the command mode at any other time. **Text input mode.** Enter text input mode by pressing **i,** **a** or **o** commands (see below) **External mode.** This mode is used to access Unix system commands whilst still in the editor.
2	How to get into and out of the editor.	**vi** *filename* - Invoke the text editor to edit a file **view** *filename* - View (read-only) a text file. **:q!** Abort the edit session without saving. **:w** Write the file to disk and stay in the editor. **:wq** Write the file to disk and quit the editor.
3	Cursor and display control	**Ctrl/e** - Scroll forward 1 line (Ctrl/Y for backwards) **Ctrl/d** - Scroll forward 1/2 screen (Ctrl/U for backwards) **Ctrl/f** - Scroll forward 1 screen (Ctrl/B for backwards) Use **j, k, h, l** keys to move cursor up, down, left, right Use **H, M, L** to move cursor to top (home), middle or lower position of current screen.
4	How to insert and delete text.	**i** - Insert new text until ESC is pressed (to return back to command mode). **a** - Append new text until ESC is pressed. **o** - Open a new line after the current line and add new text until ESC is pressed. **x,X** - delete characters **dd** - delete current line
5	How to undo your mistakes.	**u** - Undo the most recent text change. **U** - Undo all of the changes made to the current line. Repaint the screen by pressing **Ctrl/L** then **Ctrl/R**. This is sometimes useful if you have got the screen is an unusual state.

Examples of use of metacharacters. When used in the vi editor press the ESC key then the colon key before typing in the metacharacters. Note how the search strings are placed between forward slashes. If you look at the examples above you can imagine how complicated a search can get. The examples below will give you a flavour of what can be achieved, if you want to do anything more complicated, it is quite acceptable to proceed with a book on your desk or assisted by the man pages or a website.

	Example	Explanation
1	**/Monday/**	This positions the cursor on the first occurrence of the word **Monday**.
2	**/^Monday/**	This positions the cursor on the first line that starts with the word Monday.
3	**/Monday$/**	This positions the cursor on the first line that ends with the work Monday.
4	**/Monday/p**	Perform a search for the word **Monday** and **p**rint out any line containing that word.
5	**10,20**	Perform a search restricted between lines 10 and 20 for the

Getting in and out of the vi text editor

Command	Description
:q	Abort an editing session. The current editing command must be finished before the abort will work.
:q!	Force an immediate abort - no matter what start the current session is in.
:w	Write the file to disk.
:wq	Write the file to disk then exit the editor.

Cursor movement

Command	Description. Move...
h	one space left
j	one line down
k	one line up
l	one space right
^	to beginning of current line
$	to end of current line
Enter	to beginning of first word on next line
1G	to the first visible character in the first line of the file
G	to the first visible character in the last line of the file
nG	to the first visible character in the nth line of the file
CTRL/G	Display info on the current position of the cursor.
:n	line n
w	beginning of next word
e	end of next word
CTRL/E	Forward 1 line
CTRL/Y	Backward 1 line
CTRL/D	Forward 1/2 screen
CTRL/U	Backward 1/2 screen
CTRL/F	Forward 1 screen
CTRL/B	Backward 1 screen
	Type n before the command for multiple steps (n is an integer)

Inserting Text

Command	Description
a	Append to the right of the cursor.
A	Append at the end of the current line.
i	Insert starting at the left of the cursor position.
I	Insert at the beginning of the current line.
o	Open a line below the cursor, then enter insert mode
O	Open a line above the cursor, then enter insert mode
:r *newfile*	Add the contents of the file *newfile* starting below the current line.

Deleting Text

Command	Description
x	Delete a single character at the current cursor position (ccp). (4x deletes 4 characters.)
X	Delete single character to the left of ccp. (4X deletes 4 characters to the left of ccp).
dw	Delete word. (4dw deletes 4 words)
dd	Delete line. (4dd deletes 4 lines)
cw	Delete word (leaving you in insert mode, i.e. change word)
cc	Delete line (leaving you in insert mode, i.e. change line)
s	Change character - delete character and start insert mode
D	Delete from cursor to end of line
dG	Delete all lines, including current line to end of file.
d1G	Delete all lines, including current line to beginning of file.
dnG	Delete all lines, including current line to line n in the file.
J	Join the current line with the line that follows.

Shell Programming 1

The shell provides a simple user interface to the UNIX system providing the user with a command line interpreter. Shells also provide a powerful programming language with which we can create shell programs (sometimes called shell scripts) to create useful software tools that can be used to control or administer a UNIX system. A shell program is simply a file containing a set of UNIX commands that are to be executed sequentially. The file needs to have executed permissions set on it so that it can be executed just by typing in the name of the file at the command prompt.

The shell provides much more than simply 'batch' processing of a list of UNIX commands. It has many features of a high-level programming language, such as:

- Variables for storing data
- Decision-making controls (the **if** and **case** statements)
- Looping controls (the **for,while** and **until** loops)
- Function calls for program modularity

The best way to study shell programming is to look at as many examples as possible. Each of the following examples are written in 3 parts:

- The shell program. (The lines are numbered only to aid with the explanation, do not type them into the program)
- The output generated by executing the shell program
- An explanation of the program

Type each of these example programs into a file and run them, make sure that you understand what each program does and why.

To execute a shell program held in a file named **shelldemo1** do the following

$ chmod +x shelldemo1
(This puts execute permission on the file - a topic we will look at in later sessions)
$ ./shelldemo1
(This runs or executes the program. The dot represents your current directory)

Demo Program 1 - Demonstrates the use of comments, user-defined variables and echo.

1. `#!/bin/sh`
2. `#Filename: shelldemo1  Author: RS`
3. `#Define variables`
4. **`name`**`=John`
5. **`car`**`="Ford Escort"`
6. **`age`**`=21`
7. `#Display the contents of the variables`
8. `echo "My name is `**`$name`**`"`
9. `echo "I am `**`$age`**` years old and \c"`
10.   `echo "I drive a `**`$car`**`."`

Program output

```
My name is John
I am 21 years old and I drive a Ford Escort.
```

Demo Program 2 - Demonstrates how the output of Unix commands can be stored in user-defined variables

1. #!/bin/sh
2. #Filename: shelldemo2 Author: RS
3. #Define variables
4. **todaysdate**=`date`
5. **myworkingdirectory**=`pwd`
6. #Display the contents of the variables
7. echo "The date is **$todaysdate**"
8. echo "My current working directory is **$myworkingdirectory**"

Program output

```
The date is Fri Aug 17 14:30:58 BST 2001
My present working directory is /home/martin/shelldemos
```

Demo Program 3 - Demonstrating how the read command is used to get input from the user via the keyboard.

1. #!/bin/sh
2. #Filename: shelldemo3 Author: RS
3. echo "Please enter your first name: \c"
4. read **firstname**
5. echo "Please enter your surname: \c"
6. read **surname**
7. echo "Please enter your date of birth: (dd/mm/yyyy): \c"
8. read **dateofbirth**
9. echo "Welcome **$firstname $surname**, your date of birth is on record as **$dateofbirth**."

Program output (user input is shown in *bold italics*)

```
Please enter your first name: Joe
Please enter your surname: Bloggs
Please enter your date of birth (dd/mm/yyyy): 01/04/1980
Welcome Joe Bloggs, your date of birth is on record as 01/04/1980
```

Demo Program 4 - Demonstrating how user input can be obtained from the command line as command line arguments.

1. #!/bin/sh
2. #Filename: shelldemo4 Author: RS
3. echo "This program has obtained its input from the command line"
4. echo "Welcome **$1 $2**, your date of birth is on record as **$3**."

Program output
This program is exectued by entering the command:
./shelldemo4 Joe Bloggs 01/04/1980

```
Welcome Joe Bloggs, your date of birth is on record as 01/04/1980.
```

Explanation

This program is executed by typing in the shell program name followed by 3 pieces of information (program arguments).
The positions of the words on the command line are identified by the following special variables (here is the full list):
$0 The command name
$1 The first argument (Joe in this example)
$2 The second argument (Bloggs in this example)
$3 The third argument (01/04/1980 in this example)
.
.
$9 The ninth argument
$# The number of arguments
$* A space separated list of all the arguments.

Demo Program 5 - Demonstrating how decisions are made
using the *if...then* statement (testing and branching).

```
1. #!/bin/sh
2. #Filename: shelldemo5  Author:  RS
3. clear
4. echo "Would you like to see a joke (y/n)? \c"
5. read reply
6. if [ "$reply" = "y" ]
7. then
8.    echo "Question: How many surrealists does it take to change a light
   bulb?"
9.    echo "Answer: Fish."
10.    fi
11.    echo "\n\nHave a nice day."
```

Program output (examples of both yes and no user responses are shown)

```
Would you like to see a joke (y/n)? y
Question: How many surrealists does it take to change a light bulb?
Answer: Fish


Have a nice day.


Would you like to see a joke (y/n)? n


Have a nice day
```

Line 6 - The start of the **if** statement. **Notice the test is in square brackets with a space either side of them and that there is a space either side of the equals sign.**

Demo Program 6 - Demonstrating how decisions are made
using the *if...then...else* statement (testing and branching).

```
1. #!/bin/sh
2. #Filename: shelldemo6  Author:  RS
3. clear
4. echo "Would you like to see a joke (y/n)? \c"
5. read reply
6. if [ "$reply"  =  "y" ]
7. then
8.    echo "Question: How many surrealists does it take to change a light
   bulb?"
9.    echo "Answer: Fish."
10.    else
11.        echo "Not in the mood for jokes?  Never mind perhaps
   another day."
12.    fi
13.    echo "\n\nHave a nice day."
```

Program output (examples of both yes and no user responses are shown)

```
Would you like to see a joke (y/n)? y
Question: How many surrealists does it take to change a light bulb?
Answer: Fish.


Have a nice day.


Would you like to see a joke (y/n)? n
Not in the mood for jokes?  Never mind perhaps another day.


Have a nice day
```

Demo Program 7 - Demonstrating how decisions are made using
nested *if...then...else* statements (testing and branching).

```
1. #!/bin/sh
2. #Filename: shelldemo7  Author:  RS
```

3. echo "UNIX COMMAND SELECTOR"
4. echo "1. Show date"
5. echo "2. Show hostname"
6. echo "3. Show this month's calendar"
7. echo "Please make your selection (1,2,3) \c"
8. read menunumber
9. if   [$menunumber -eq 1]
10.   then
11.   date
12.   else if   [$menunumber -eq 2]
13.   then
14.   hostname
15.   else if [$menunumber -eq 3]
16.   then
17.   cal
18.   else
19.   echo "INVALID CHOICE! \07\07"
20.   fi
21.   fi
22.   fi
23.   echo "\nThank you for using the Unix command selector."

Program output (The example is for the user input of 1)

```
UNIX COMMAND SELECTOR
1. Show date
2. Show hostname
3. Show this month's calendar
Please make your selection (1,2,3) 1
Tues Oct 23 17:32:45 BST 2001


Thank you for using the Unix command selector.
```

Explanation

Lines 3-7 Display a title, menu and prompt the user to select the number of a menu option.
Line 8 - The user's response is read into a user defined variable named *menunumber*.
Lines 9-22 A series of nested if...then...else statements each performing a test for one of the possible menu option numbers.
The 3 possible Unix commands are: 1. date   2. hostname   3. cal
Notice how **-eq** is used to test for equality between two numbers.
Notice how \07\07 are used to get the computer to beep twice.
Line 23 Displays a final ending message.

Nested *if...then...else* structures always look complicated as there is a lot of coding to contend with. A simpler way of writing the nested **if...then...else** parts is shown below using the **if...then...elif** with one final **fi**. The word **elif** is a contraction of the words **el**se **if.**

if [$menunumber -eq 1]
then
   date
elif [$menunumber -eq 2]
then
   hostname
elif [$menunumber -eq 3]
then
   cal
else
   echo "INVALID CHOICE! \07\07"
fi

This may be a little better to deal with but nested *if...then..elif's* are still a handful.

Courtesy: Rushdi Shams Sir