vi Editor Basic Commands

vi operates in two modes: ***command mode*** and ***input mode***. When you open a file, you start in the *command mode*. In the *command mode*, whatever you type is taken as commands to the editor and not as text to be typed into the file, while in *input mode*, your keystrokes appear as text in the file. You can enter *input mode* by typing commands such as ***a*** (for append), or ***i*** (for input). To get back to *command mode*, hit the ***Esc*** key.

Command	Description
Switching between command and input modes	
i (insert)	Enter *input mode*; text to be entered at current cursor position
I	Enter *input mode*; text to be entered at beginning of current line
a (append)	Enter *input mode*; text to be entered after current cursor position
A	Enter *input mode*; text to be entered at the end of current line
s/S (substitute)	Enter *input mode*; current character/line to be replaced
cw (change word)	Enter *input mode*; word starting from cursor to be replaced
o/O (open)	Open a blank line after/before the current line; enter *input mode*
Esc	Get back to *command mode*.
*Cursor Movement (in command mode. i.e, after hitting **Esc**)*	
h,j,k,l	Move cursor left, down, up and right respectively
Arrow keys	Move cursor in the direction of arrows (works in *input mode* also)
0	Move to the beginning of current line
^	Move to the first non-blank character of current line
$	Move to the end of current line
:n	Move to the first non-blank character of the n^{th} line
w	Move one word forward
b	Move one word backward
%	Move to the corresponding opening/closing bracket (), [] and {}.
*File read, write, quit (in command mode. i.e, after hitting **Esc**)*	
:w (write)	Write the current contents into file
:w *myfile.c*	Write the current contents into file named **myfile.c**
:q (quit)	Exit/quit the editor
:q!	Quit the editor and discard any changes after last write
:wq (write & quit)	Write/save the current buffer and quit the editor
:r *filename* (read)	Read the contents of '*filename*' and insert it after current line
Delete, Copy, Paste text *(in command mode. i.e, after hitting **Esc**)*	
x	Delete character under cursor
dd (delete line)	Delete current line
dw (delete word)	Delete current word
yy (yank line)	Copy the current line into buffer
p/P (paste)	Paste the contents of buffer after/before current cursor or line
Other Commands: Undo, Redo, Repeat, Find	
u	Undo last edit (you can repeat for previous changes)
CTRL-r	Redo what was undone (you can repeat for newer changes)
.	Repeat last command at current cursor position
/string	Search for *string* and go to the first occurrence
n	Repeat last search; go to first occurrence after cursor

Bash Usage

bash stands for *bourne again shell*, and pun on and improvement of the bourne shell developed by Stephen Bourne at the Bell Labs. When you open a terminal window, it is bash (or another shell) that interprets the commands that you type into it and executes them for you.

Each instance of bash (you can have multiple instances in multiple terminals) has a directory (or folder) referred to as the current directory. Think of this as your present location in the file system. You can move around the file-system and peek at the contents of files and directories using bash using the following commands:

Command	Description
Commands to explore the file system	
pwd	Show the current directory (present working directory)
cd *dir-name*	Change the current directory to *dir-name* (change directory)
cd ..	Change current directory to the parent of current directory
ls	List the files in current directory, including sub directories
ls -l	Longer form of ls, which displays some details of each file
ls -lh	Longer form of ls, in human readable form
ls -la	Show all files (including hidden files) in the longer form
cat file1	Show the content of *file1* on the screen (concatenate)
head/tail fill	Show the first/last 10 lines *fil1*

You can also manipulate (create, delete, copy, move) files and directories using bash.

Command	Description
Commands to explore the file system	
cp file1 file2	Make a copy *file1* under the name *file2*
cp file1 dir1	Make a copy of *file1* inside the directory *dir1*
mv file1 file2	Move *file1* to *file2* (can be thought of as renaming)
mv file1 dir1	Move *file1* to the directory named *dir1*
rm file1	Remove (or delete) the file *file1*
rmdir dir1	Remove (or delete) the directory *dir1*
mkdir dir1	Make a directory named *dir1* inside the current directory
touch file1	Create a new empty file named file1

Other useful commands: **which** (path of command), **man** (manual page), **whatis** (brief command description), **apropos** (find a command), **date** (current date and time), **who** (list of users logged in) , **id** (your details), **top** (dynamic list of processes), **ps** (process status), **kill** (terminate a process), **chmod** (change permissions of a file/directory), **ssh** (create a secure remote shell), **sudo** (run a command that requires superuser permissions), **du** (disk usage), **df** (free space), **locate** (a file), **less** (page through file).

One can use ***history*** to see the previously executed commands, *up and down arrows* to step through and *!prev_com…* execute previous commands and the *tab* key to auto-complete partially typed commands. Use **pushd** / **popd** to jump to a directory and back.

Use **|**, **<**, **>** and **2>** to redirect inputs and outputs of commands. See ***~/.bashrc*** file for settings.

Shell Scripting

bash allows writing of highly effective scripts using a few programming constructs and a wealth of commands. Here is a summary of constructs and commands from *bash*:

Scripting Construct	Description / Usage
A=*Name*	Creates a variable: '*A*' with value as the string: *Name*
$A	Gives the value of the variable *A*.
echo	Evaluate the argument and print it
read *varname*	Reads input from the user; stores the value in *varname*
declare	Declare a new variable and initialize value
Arr=(v1 v2 v3)	Declare an array, ***Arr***; Accessed as: ${Arr[0]}, ${Arr[1]}
$1, $2, $@	Command line arguments passed to the shell script
if then else fi	*if - then - else* construct
case *exp* **in esac**	Multiple condition matching and execution
while do .. done	*while* loop
for do .. done	*for* loop
exit	Terminate the process running the script

Any content after a # symbol is a comment, except if a #! appears on the first line followed by the path to the script interpreter.

A few useful commands to use with scripts (in addition to the ones mentioned in the previous page). Add your own below:

- **find / locate**: search for a file in a directory or its sub-directories
- **grep, cut, sed**: for text/string filtering and processing
- **awk**: Table processing
- **file**: type of a file (can be used along with if to select files of a specific type)
- **tar**: create an archive of (also compress, encrypt) a given directory.
- **curl**: access files on remote machines or upload using http, ftp, ldap, smtp, etc.
- .
- .
- .
- .
- .

Commands/concepts useful in conjunction with shell scripting

- **time**: run a command and report statistics
- **cron**: used to run a script at specific times in a day/week.
- **$PATH** variable.