

vi UPT 17.1

- ☞ “vi” stands for the Visual editor.
- ☞ Newest forms such as vim and gvim are much more featureful than the original barebones editor.
- ☞ It’s “standard” on all Unix machines, and a great way to get emacs going!
- ☞ While it doesn’t make automatic backups of files edited, it also doesn’t leave tilde files all over the place.



☞ It is generally quite efficient.



Calling vi

The vi editor is invoked by issuing the command in the following form. The -r option is for recovering a file where the system crashed during a previous editing session. The -t option is to indicate the position within a file the editing should start.

```
vi [-t tag] [-r ] filename
```



Modes in vi

☞ It has three main modes:

⇒ character input mode: where text can be entered

▣➔ insert, append, replace, add lines

⇒ window mode: where regular commands can be issued

▣➔ basic cursor motions

▣➔ screen control

▣➔ word commands

▣➔ deletions



- ▣➔ control commands
- ▣➔ miscellaneous commadns
- ⇒ line mode: where ex or ed commands can be issued



Character input/output

After invoking `vi`, the user is in the window command mode.

There are a few different commands to enter character input mode.

At that point, a user types in any desired text. The user then uses the ESC key to return back to command mode.



Commands to enter Character Input Mode

a append text after the cursor position
A append text at the end of line
i insert text before the cursor position
I insert text before the first nonblank character in the line
o add text after the current line
O add text before the current line (letter O)
rchr replace the current character with ‘chr’
R replace text starting at the cursor position



Basci cursor motion

h go back one character
j go down one line
k go up one line
l go forward one character (space also works)
0 go to the beginning of the line (zero)
\$ go to the end of the line
H go to the top line on the screen
L go to the last line on the screen



Word movement

w position the cursor at the beginning of the next word
b position the cursor at the beginning of the last word
e position the cursor at the end of the current word



Screen control

`^U` scroll up one half page
`^D` scroll down one half page
`^B` scroll up one page
`^F` scroll down one page
`^L` redisplay the page



Deletions

dd delete the current line
D delete text from the cursor to the end of the line
x delete character at the cursor
X delete character preceding the cursor
dw delete characters from the cursor to the end of the word



Searching

/pattern	search forward for "pattern"
/	search forward for last "pattern"
?pattern	search backward for "pattern"
?	search backward for last "pattern"
n	re-perform the last / or ? command



Miscellaneous

u undo previous command
U restore entire line
Y save current line into buffer
p put saved buffer after cursor position
P put saved buffer before cursor position
J join current line with following line
% position cursor over matching "(", ")", "{", or "}"
ZZ save file and exit (same as :wq)



Repetition

You can specify how many times a command is to be performed:

```
3dd    delete 3 lines
4w     advance 4 words
7x     delete 7 characters
5n     perform last search 5 times
```



Working with tags

The `ctags` and `etags` programs let you take in a set of source files as input and creates a `tags/TAGS` file as output.

The tags file contains for each function and macro

- ➡ Object name
- ➡ File in which the object is defined.
- ➡ Pattern describing the location of the object.



The output of `etags` is also useful with `emacs`.



Using a tags file

You can use the `-t` option when invoking `vi` to find a particular function.

```
vi -t main  
vi -t max
```



gvim

There is a graphical version of vi called gvim.



Multi-level undo in vim (not vi, though)

Can use the `u` command to undo multiple changes, as opposed to `vi`, which can only undo the last change. Each time you enter `u`, the previous change is undone.

