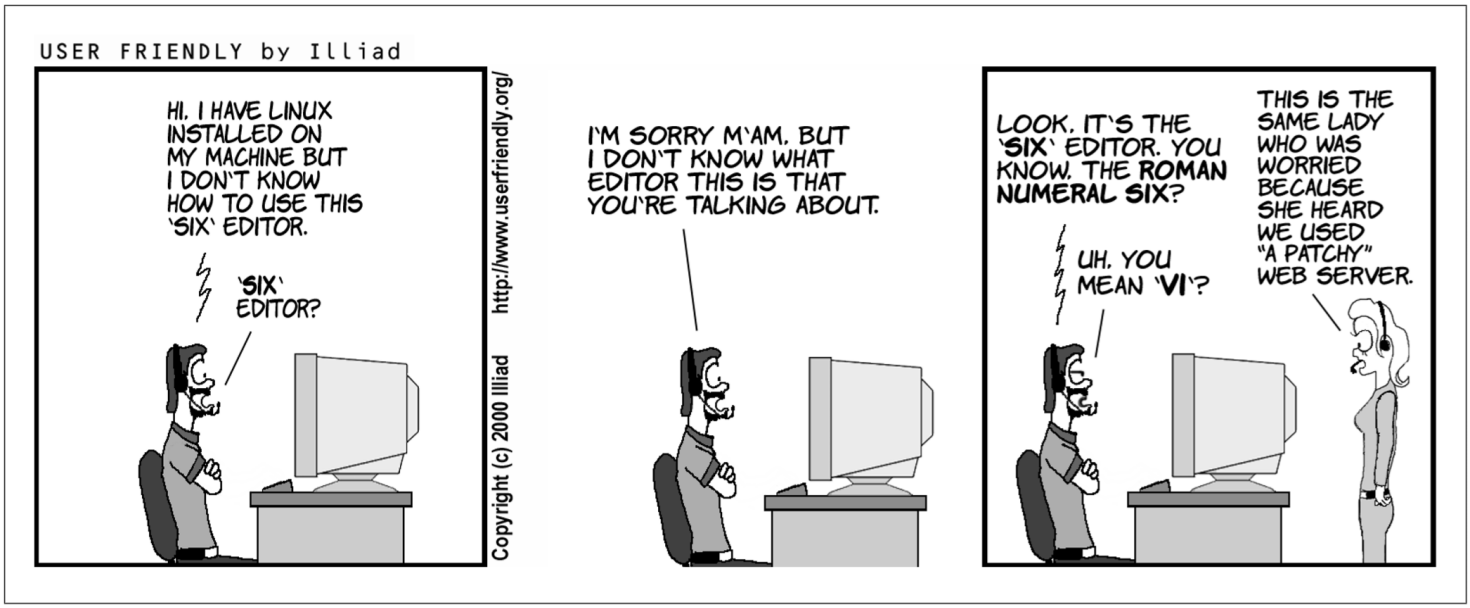
VI tutorial



Some preliminaries

You can use `vi` to edit any text file. `vi` copies the file to be edited into a buffer, displays the buffer, and lets you add, delete, and change text. When you save your edtis, vi copies the edited buffer back into a file, replacing the old file of the same name.

Minimal vi

Opening a file

Type `vi [filename]` to open an existing or a new file. Notice the tildes at the left side, which indicates empty lines and the name and status of the file on the last line (called the prompt line or the status line).

Two faces of vi

Improtant!
When you open a file, you start out in the *command* mode, one of the two modes in which vi operates (the other being the *insert* mode). In the command mode every keystroke represents a command, rather than a text that you try to type. To switch to the *insert* mode, you use `i`, `I`, `a`, `A`, and a few other commands. To

get back to the command mode, you use the `ESC` key.

Basic editing

After you entered the editing mode, you can move by using arrow keys, delete pre-existing text with the `delete` key, and type your new text as in any other editor (but note that you can't use the mouse in vi (but see below)). To switch back to the command mode, use the `ESC` key.

To save and quit a file

Use `ZZ` command to save edits and quit (note that ZZ is capitalized; vi commands are case sensitive). However, it is very common in `vi` to use commands from another editor, called `ex` (in fact, vi is the visual mode for that editor!). To save and quit a file using the `ex` command, you type `:wq` or `:x`. You type `:q!` to exit the editor without saving the file.

Congratulations! You can now edit in vi!

Intermediate vi

There are many additional vi command, which will make your editing more efficient. Here are some that I found useful:

Simple edits

`i` to insert text before the cursor (`I` at the beginning of the line)
`a` to add text after the cursor (`A` at the end of the line)
`d` to delete
- `dw` to delete a word
- `dd` deletes a line
- `dl` or `x` deletes one character
- `D` deletes from the cursor position to the end of the line

`r` will replace a single character (without switching to the editing mode)
`~` will replace the case of the letter

You can also use `cw`, `cc`, `C` to change a word, line, or the rest of the line and `s` to substitute character at cursor (`S` will substitute a line).

Important!

For most of these commands, you can add an integer in front to execute them multiple times. *E.g.*, `2dd` will delete two lines, `2cb` will change back two words, and `20i hi, there!` + `ESC` will insert that phrase 20 times.

Finally, `.` will repeat the previous command and `u` will undo the last change.

Complex movements

There are vi commands to scroll forward and backward through the file by full and half screens: `^F`, `^D`, `^B`, and `U`, respectively, where `^` stands for the `CTRL` key.

On the screen itself, you can move to the top line, middle line, and last line with `H`, `M`, and `L`.

Use `0` / `^` and `$` to move to the beginning and end of a line.

Use `#G` to go to a specific (#) line of your file. `G` without a line number moves the cursor to the last line of the file. Typing two backquotes returns you to your original position. BTW, to see numbers in vi type `:set nu` or `:set number` (`:set nu!` or `:set nonumber` to remove them).

Obviously, there are commands for "simple movements" as well: `h`, `j`, `k`, and `l` move your cursor left, down, up, and right, respectively, while `e` and `b` move it to the beginning/end of a word. `+` and `-` will move you to the first character of the next and prevous line, respectively.

Movement by Searches

A quick way to move around in a large file is by searching for a pattern of characters using the `/` (slash) command. When you enter a slash, it appears on the bottom line of the screen; you then type in the pattern that you want to find: `/pattern` and press `ENTER`. You can use the regular expressions in your search pattern.

To search backward, type `?pattern`.
To repeat the search forward/backward, type `n` / `N`, respectively.

You can also search for a specific character in a line, by using `fx` and `Fx` commands (`x` can be any character). `;` and `,` will repeat the search.

Cutting, copying, and pasting.

To cut any text from a file, you delete it (all versions of the `d` command, like `dd` and `dw` will work). The deleted text is temporarily saved in a special buffer. To paste it back, use the `p` (put) command. The lowercase `p` puts the text after the cursor position. The uppercase `P` puts the text before the cursor.

There are *nine* unnamed buffers, where your deletions are saved. You access them with `"`. The last delete is saved in buffer 1, the second-to-last in buffer 2, and so on. To recover your second-to-last deletion, type: `"2p`.

To copy some text, use the `y` (for yank) command. The yank command can be combined with any movement command (`yw` , `y$` , `y^`). Both `yy` and `Y` yank the whole line. After yanking some text, you can paste (put) it with `p` or `P` .

In addition, to unnamed buferrs, there are 26 named buffers (a-z) where you can save deleted or yanked text.

`"z5dd` will delete five lines into buffer z
`"z5yy` will copy five lines into buffer z.

If you specify a buffer name with a capital letter, your yanked or deleted text will be appended to the current contents of that buffer.

Joining Two Lines with J

Use `J` to join the current line with the following line.

Repeating or Undoing Your Last Command

`.` (a period) will repeat the previous command `u` will undo the previous command `U` undoes all edits on a single line, as long as the cursor remains on that line.

Congratulations! You can be very efficient in vi now! But there is more...

Advanced vi

Ex editor

You have seen some of the ex commands already (`:x` , `:wq` , *etc.*). In fact, every time we type a colon in **vi** we invoke an ex command. Technically, **ex** is a line editor and **vi** is its visual mode (also known as a screen editor). You can use **ex** by itself in unix, but it can be tricky. The thing to remember is that **ex** deals with lines, so its basic commands `d` , `m` , `co` will delete, move, and copy a line. The power of **ex** comes from its ability to deal with multiple lines at the same time.

Three ways to access lines in ex:

- With explicit line numbers

- With symbols that help you specify line numbers relative to your current position in the file
- With search patterns as addresses that identify the lines to be affected

The first way is rather selfexplanatory: `:3,18d` will delete lines 3 through 18 (remember, that you can see line numbers with `:set nu` or `:set number`).

You can also use symbols for line addresses. A dot `.` stands for the current line; and `$` stands for the last line of the file. `%` stands for every line in the file. For example: `:.,$d` deletes from current line to the end of file and `:%d` deletes all lines in the file.

Finally, ex can address lines by using search patterns. For example: `:/pattern/d` delete the next line containing pattern. `:/pattern/+d` deletes the line below the next line containing pattern, while `:/pattern1/,/pattern2/d` deletes from the first line containing *pattern1* through the first line containing *pattern2*.

Finally, all three ways to access lines can be combined in the same command.

Global replacements

Global replacements use two **ex** commands: `:g` (global) and `:s` (substitute).

The substitute command `:s/old/new/` changes the first occurrence of the pattern *old* to *new* on the current line. If you add `g` after the pattern: `:s/old/new/g` it will change every occurrence of *old* to *new* on the current line.

Important!
By prefixing the `:s` command with addresses, you can extend its range to more than one line: `:50,100s/old/new/g` or (my favorite) `:%s/old/new/g`.

To confirm each replacement before it is made, add the c option (for confirm) at the end of the substitute command: `:1,30s/his/the/gc`. Alternatively, one can use `n` (repeat last search) and dot `.` (repeat last command).

You can also make replacements only on lines that match a specific pattern by combining the global command `g` and the substitute `s` command `:g/pattern/s/old/new/g` (you can skip *old* if it is the same as *pattern*).

Important!
When making replacements, you can search not just for fixed strings of characters, but also for regular expressions (using `.`, `*`, `[]`, etc).

`\( \)` Saves the pattern enclosed between (and) into a “hold buffer.” Up to nine patterns can be saved in

this way on a single line and used in replacement string by the sequences `\1` to `\9`.

`\< \>` match characters at the beginning (<) or at the end (>) of a word.

In addition to the regular characters and regular expressions, **vi** can use *POSIX Bracket Expressions* in searches. Examples include alphanumeric characters `[:alnum:]`, collating symbols `[.sch.]`, and equivalence classes `[=e=]` matching e, è, or é in a French locale. Note that these expressions should themselves be used inside the square brackets (*e.g.*, `[[:alpha:]]`).

Working with multiple files

Copying a file into another file

To read the contents of another into a file you are editing, use the `:read filename` or `:r filename` command.

If you want to specify a line other than the one the cursor's on, simply type the line number (or other line address) you want before `r` (or `read`). You can also use all other ways to access lines in **ex**.

Editing multiple files

You can edit multiple files in **vi(m)** by providing several filenames as vi arguments: `vi file1 file2 file3` or `vi file*`. You can also load another file at any time with the ex command `:e`. But note, that to edit another file within **vi**, you first need to save your current file (`:w`), then give the command: `:e filename`. After you have finished editing the first file, the ex command `:w` writes (saves) file1 and `:n` calls in the next file (file2). The `:args` command (abbreviated `:ar`) lists all the files you are editing, with the current file enclosed in brackets. You can also use vi `Ctrl^` command to switch to the file you edited previously.

Opening files in different tabs or windows(splits)

Use `vi -p file1 file2 file3` command to open several files in different tabs;
Use `vi -o file1 file2 file3` to open files in horizontal splits;
Use `vi -O file1 file2 file3` to open files in vertical splits.

Searching and replacing in multiple files

Often search and replace is needed in multiple files. This tip uses the procedures from run a command in multiple buffers to show how a substitute may be executed multiple times using :argdo (all files in argument list), or :bufdo (all buffers), or :tabdo (all tabs), :windo (all windows in the current tab), or :cdo (all files listed in the quickfix list).

The following performs a search and replace in all buffers (all those listed with the `:ls` command):

```
:bufdo %s/pattern/replace/ge | update
```

These are actually two commands under `bufdo` : the first is by now familiar `%s` search and replace and the second is `update` , which writes a file only if changes were made. The e option in th `%s` command tells it not to print an error if the pattern is not found and `|` is a separator between commands.

One alternative is to set the `autowriteall` option so changed buffers are automatically saved when required:

```
:set autowriteall
:bufdo %s/pattern/replace/ge
```

Another alternative is to set the 'hidden' option so buffers do not need to be saved, then use `:wa` to save all changes (only changed buffers are written):

```
:set hidden
:bufdo %s/pattern/replace/ge
:wa
```

If you don't wish to save the results of your replacement, but want to review each changed buffer first, you can force the bufdo to continue without saving files with bufdo!:

```
:bufdo! %s/pattern/replace/ge
```

For more info see Vim tips wiki

VIM (Vi IMproved)

- **Syntax extensions**
  Vim lets you control indentation and syntax-based color coding of your text.
- **Programmer assistance**
  Vim offers many features normally found in Integrated Development Environments (IDEs).
- **Graphical user interface (GUI)**
- **Scripting and plug-ins**
  You can write your own Vim extensions or download plug-ins from the Internet.
- **Initialization**
  Vim, like vi, uses configuration files to define sessions at startup time, but Vim has a vastly expanded repertoire of definable behaviors.
- **Session context**
  Vim keeps session information in a file, .viminfo. Not only does Vim remember edits in your last session for a file, it remembers basic things between files. This is handy for editing activities such as grabbing a sequence of lines in one file (with y [yank], or d [delete]) and “putting” them in another.

- **Postprocessing**
  In addition to performing presession functions, Vim lets you define what to do after you've edited a file. You can write cleanup routines to delete temporary files accumulated from compiles, or do real-time edits to the file before it's written back to storage. You have complete control to customize any postedit activities
- **Transparent editing**
  Vim detects and automatically unbundles archived or compressed files. For example, you can directly edit a zipped file such as myfile.tar.gz. You can even edit directories. Vim lets you navigate a directory and select files to edit using familiar Vim navigation commands.
- **Meta-information** Vim offers four handy read-only registers from which the user may extract meta-information for "puts": the current filename (%), the alternate filename (#), the last command-line command executed (:), and the last inserted text (., a period).
- ***etc.***