Chapter 6: The vim Editor

An Exercise in Memorizing Your Alphabet

In this chapter ...

- `ed`, `ex`, `vi`, `vim`
- `vim` basics
- Command Mode
- Input Mode
- Last Line Mode
- Buffers
- Yanking

In the beginning …

- There was `ed` … single line editor
- Then came `ex` … had a nifty visual mode
- Visual mode lead to `vi`
- Written by Bill Joy (BSD, csh, Sun) in 1976
- `vi` a Unix utility – so we need a free clone
  - `elvis`, `nvi`, `vile`, and `vim`

vim

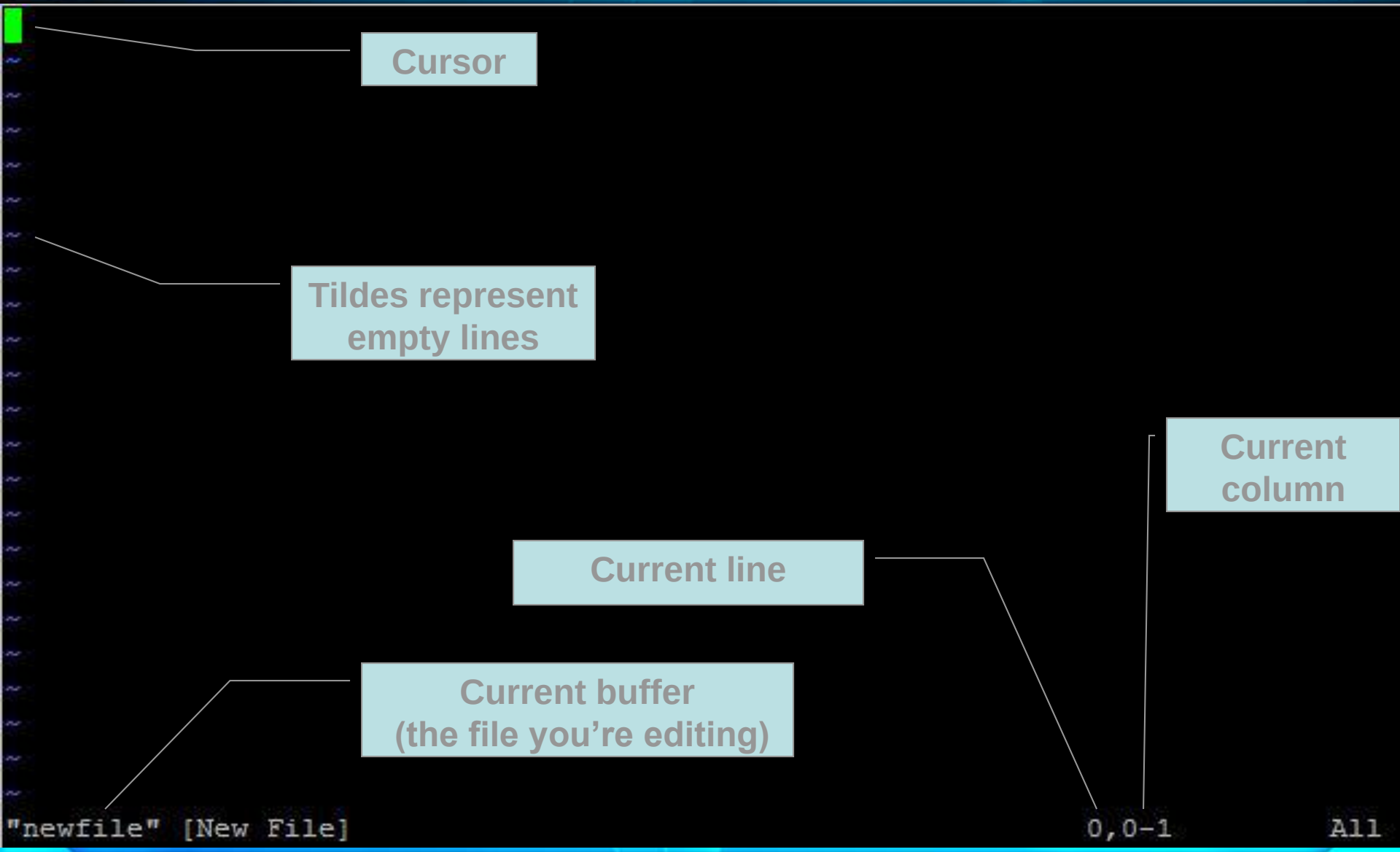
- We'll be using `vim` - vi improved
- Written by Bram Moolenaar
- In our RedHat distro, we have `/bin/vim`
- `vi` is just an alias to `vim`
- Standard in just about all Linux distros
- Available from www.vim.org
  - gVim, multi-platform

vim con’t

- Powerful, quick text editor
- Excellent for programming due to intelligent language detection
- NOT a formatting tool … plain text only
- Nearly limitless options and commands
- Excellent tutorial - `vimtutor`

Starting vim

- Syntax: **`vim [`*`options`*`] [filename]`**
- Filename is optional
  - With it, it opens that file for editing
  - Without, it opens a default screen
- Many options available, most commonly used ones are for file recovery


Cursor
Tildes represent empty lines
Current column
Current line
Current buffer (the file you're editing)
"newfile" [New File]
0,0-1
All

How it works

- vim copies the contents of the file you want to edit into memory
- This memory is referred to as the Work Buffer
- Changes are made to the buffer, not the file
- You must write changes to file when done editing the buffer

Modes

- `vim` has three modes
  - Command
  - Input
  - Last Line
- When you start `vim`, you begin in Command Mode by default
- Hitting `ESCAPE` will get you back to Command Mode from other modes

Command Mode

- Default mode
- Used to enter commands
  - Text manipulation
  - Change modes
  - Save/exit
- Most commands are just alpha characters, not control sequences
- Case sensitive!

Insert Mode

- The mode that lets you edit and enter text
- Several sub-modes
  - Insert
  - Append
  - Open
  - Replace
  - Change
- You'll spend most of your time here

Last Line Mode

- From command mode press :
- Cursor jumps to the last line on the screen
- Here you can manage files, issue shell commands, change editor settings
- Also where you go to exit

Getting into Input Mode

- `i` nsert before cursor
- `I` nsert before first nonblank character on line
- `a` fter cursor
- `A` t end of line
- `o` pen line below
- `O` pen line above
- `r` eplace current character
- `R` eplace characters

Command Mode - Essentials

- `h` move cursor left
- `j` move cursor down
- `k` move cursor up
- `l` move cursor right
- `x` delete character
- `dw` delete word
- `dd` delete line
- `ZZ` write and quit

Command Mode con't

- `/`*`regexpr`* search forward
- `?`*`regexpr`* search backwards
- `n` repeat last search (ie, find next result)
- `N` repeat last search, in opposite direction
- *`n`*`G` Jump to line *n* (omit *n* to go to last line)

Last Line Mode Essentials

- `w` write file
- `q` quit
- `w!` write read-only file
- `q!` quit without saving changes
- `e` *`filename`* opens a file for editing

Last Line Mode con't

- `sh` open a shell
- `!`*`command`* open a shell, run a command, then exit the shell
- `.!`*`command`* open a shell, run a command, exit the shell, placing the standard output into the work buffer
  - Can also do `!!`*`command`* from Command Mode

Buffers

- Work buffer
- General Purpose Buffer – kind of like the clipboard in Windows
- Named buffers
- Numbered buffers

General Purpose Buffer

- Contains recently edited or deleted text
- It’s where undo information is stored
- You can copy (yank) text to this buffer and then paste (put) it elsewhere

Named Buffers

- Similar to the General Purpose Buffer
- Does not contain undo info – only contains text if you put it there
- Each of the 26 buffers is referenced by letter a-z

Numbered Buffers

- Numbered 1-9
- Read only
- Contain most recently deleted chunks of data greater than one line long
- You can paste (put) from these buffers and use them for undoing deletes

yank

- Copies lines of text
- To yank a line, use `yy`
- Or use `Y` – it's shorter
- To yank multiple lines, place cursor on the first line and use *n*`Y`, where *n* is the number of lines to yank

yank con’t

- By default it yanks text to the General Purpose Buffer
- To place in a named buffer, precede the yank command with double quotes and the letter of the buffer you wish to use
- Use lowercase letter to overwrite, upper case to append
- Ex: `"c5Y` would yank 5 lines to buffer c

put

- Pastes text from a buffer into the Work Buffer
- Use `p` to put below current line
- Use `P` to put above current line
- Again, if using a named buffer, precede with double quotes and the letter

vim

- Just barely scratching the surface
- Hundreds of commands
- Command modifications
- We'll cover searching and substituting in Appendix A