Vim

<< Parent: N/A || Nexus || Child: N/A >>

Tags: #vim #tool #text-editor #linux #institute-course #data-processing

Created: `= this.file.ctime`
Modified: `= this.file.mtime`

Introduction:

From vim.org:
"Vim is a highly configurable text editor built to make creating and changing any kind of text very efficient. It is included as "vi" with most UNIX systems and with Apple OS X.

Use Case:

Functionality:

- stable and maintained
- persistent, multi-level undo tree
- extensive plugin system
- support for hundreds of programming languages and file formats
- powerful search and replace
- integrates with many tools

Advantages & Disadvantages:

Great hands-on-keyboard text editor with excellent features for quickly editing files and quick, keyboard based navigation, search, find and replace, and more.

Not great for managing a tree of files.

Related:

List similar tools that can be used as an alternative:

- nano
- VS Code
- mousepad

Breakdown:

Basic:

Initial Setup

`vim --version`

- make sure `+clipboard` is included for linking system clipboard and vim clipboard functions
  - If not enabled, install `vim-gtk3` or search for how to add it for your repo

Key Terms

Mode
Vim operates in multiple modes. These modes have unique commands and features to programmatically handle the processing of data and text

Buffer
Vim uses buffers of files instead of direct read/write access. Practically, a buffer is a file in use by vim.

Register
A vim-internal clipboard that can be linked to the system's clipboard.

Modes:

Command/Normal Mode
Summary: for reviewing text, making edits, and using commands for vim itself
Command: `:`

Insert Mode
Summary: for composing text
Command: `i`

Visual Mode
Summary: for highlighting text and manipulating portions of text by selection
Command: `v`
Note: There is a visual block mode available with `V`

Pressing `esc` will return you to **Command Mode** from any other mode.

Command Mode:

Most navigation and text editing commands can be prepended with a number. For instance if you wish to delete to the end of a word `de` can be used. To delete from cursor to the end of the 4th word (array starting at 1 here), `4de` can be used.

Common Commands:
Note: all commands begin with `:`
`:w` writes current file from stash to file
`:q` quits vim
`:x` writes current file from stash to file *only if the file has been modified* and quits
`:w filename` writes current file from stash to file located at path to $filename, stays in the same stash

`!` forces command
ex: `:wq!` will write, then force quit the editor

`:help` or `:h` opens help in a new file, with a split display
`:help $` searches the help file for `$`

Navigation Commands

`hjkl` - default arrow keys for navigation in vim, many implementations
`h` - move left
`j` - move down to next line (based on line breaks by default)
`k` - move up to previous line (based on line breaks by default)
`l` - move right

`G` - move to end of document
`gg` - move to top of document
`#gg` - navigate to the prepended line number
`e` - move to end of word, including last character (punctuation included)
`b` - move to beginning of word
`(` and `)` - navigate sentences
`{` and `}` - navigate paragraphs
`^` - navigate to the beginning of a line
`$` - navigate to the end of a line
`w` - moves to the start of the next word, excluding the first character

Text Editing Commands

Deletion
`x` or `del` - delete character selected
`u` - undo previous change
`D` - delete to end of line
`dd` - delete entire line

Insertion/Creation
`p` - pastes the text [from register x] after the cursor
`P` - pastes the text [from register x] before the cursor
`y` - "yank" - copies text to the register [register x, usually]
`Y` - yanks the entire line

Managing Buffers

Buffers distinguish between the file currently opened by vim and the original file. It is not the file until the buffer is saved to the file.

Note: Buffers are managed from Command Mode

`e file2` - Opens `file2` as another buffer within vim
`buffers` - shows what buffers are currently opened in the current instance of vim
- Contain id numbers for each buffer
- Contain special symbols:
- `#` current buffer
- `%` alternate buffer

`ls` - shorthand for `buffers`
`buffer 1` - Switches to buffer with id of `1`
`badd` - opens another buffer in the background

Managing Registers

Based on the kind of data removed, the data will be stored in different registers (`help registers` for more information).

Registers have specific names, but not id numbers (like buffers).

Examples of commonly used registers:
`""` Unnamed buffer that serves as the default buffer
`"0` through `"9` These buffers are historical buffers of yanks moved from the unnamed registers to these named registers

Read Only Registers
`".`, `"%`, `":` and `"#` are the read-only registers
`".` is the most recently inserted text
`"%` is the current file path, starting from the directory vim was first opened in
`":` is the most recently executed command or what is saved with
`"#` is the name of the alternate file go to `:h alternate-file` for more detail

Insert mode

Allows the typing of text.

Key Shortcuts
`i` enter Insert Mode
`o` start Insert Mode at the next line
`O` start Insert Mode at the line above
`a` start Insert Mode one character ahead of cursor
`A` start Insert Mode at the end of the current line (does blocks as well)

Note: prepending a number with `i` or `a` will insert a text multiple times.
Example:
`50iA` then the `esc` key will insert 50 occurrences of the letter "A"

Visual Mode

Enter **Visual Mode** with `v`.
***Note:** the selection begins with the highlighted character**
Enter **Visual Line Mode** (selects entire lines) with `V`.
Enter **Visual Block Mode** with `ctrl + v`

Common editing tools:
`U` will uppercase all highlighted text
`u` will lowercase all highlighted text
`~` will change the case of just the selected character

When text is selected in a visual mode, the characters `:'<,'>` will show. This represents the selected text and typical substitutions (like global regex) can be performed on just the selected text.

Novice/Advanced:

Using Regex in Vim

See Regex Notes in KB

Running shell commands from Vim

`:!` signals to vim that you're running shellcode (ex.: `:! ls` will run the `ls` from the cwd)
Note: After running a command, you will be prompted to type `command` or press `Enter` to be returned to vim. This is best for just a quick command.

Reading in files to vim:
`:r file1` reads in the contents of the file1 document into the current buffer of vim
`%! rev` passes the global contents of the buffer (`%`) to the shell command `rev`
`%!xxd` read a file into the vim buffer as a hex table with the `xxd` command
- *Note:* this is best used by opening the file with vim, then passing it to the `xxd` hex editor, then using `%!xxd -r` to reverse it back to the original file.

Read the results of a command into your buffer:
`:r!` this tells vim to read the *output* of the shell command into the buffer.
- `:r! ls -latch` would load the contents of the `ls -latch` command into the current buffer.
- to squash spaces in this output, use `:%!tr -s` (`tr` is the unix *truncate* command with the spaces flag set to standardize spaces to single)
- other commands, such as `cut`, `awk`, etc. can be used from here to trim the data
- Full one-liners can be used (Ex.: `:%!tr -s | cut -d " " -f 1,3`)

Windows for buffer management

Similar to running `:help`, buffers can be opened in multiple windows.

`:split` opens one buffer in two windows
`ctrl` + `w` + *navigation keys* allow you to move between opened windows
`ctrl` + `w` + *capital navigation keys* moves the current window to another location (switches the windows)
`:q` will close the current window
`:split file` will open a second window with `file` opened in it.
`:vsplit` vertical split windows
`ctrl` + `w` + `s` runs the `:split` command and splits the current window horizontally
`ctrl` + `w` + `v` runs the `vsplit` command as above
`ctrl` + `w` `<` or `>` resizes windows horizontally
`ctrl` + `w` `-` or `+` resizes windows vertically
`:resize #` or `:res #` the height of a window to `#` characters tall
`:vertical resize #` sets the width of a window to `#` characters wide
`:resize` or `vertical resize` `+/-#` changes the size of the window relatively

Tabs for buffer management

Tabs function similar to tabs in a browser

`:tab new` or `:tab n` creates a new tab
`:tabprev` or `:tabnext` to cycle through tabs
`gt` and `gT` cycle through tabs
`:tabe file` stands for tab edit then the path to the `file` to edit
`:tabmove [#]` moves the current tab to the array-based index location (reminder: arrays start with 0)
`:tabclose` closes the current tab

Note on Windows and Tabs: Windows are the children of tabs, not the other way around. Tabs will show how many windows are open in each tab with a numeral greater than 1 prepended to the tab.

Customization:

.vimrc

The `.vimrc` (linux) `_vimrc` (windows) file in the user directory stores permanent preference data for vim. Any setting can be changed here.
Example:

```
set number
set spell
set clipboard=unnamedplus
```

This example `.vimrc` file will show absolute numbered line (`set nu` will also work), spellcheck documents (`set nospell` to disable), and shares the vim clipboard with the system clipboard

In order to load the changed `vimrc` file, you will need to:
`:source ~/.vimrc`

Use the resource for building your `.vimrc` in the **Footnotes** section

Plugins

Vim can use plugins that change the behavior of the program. This can include color schemes, tree views, etc.

Plugin Managers

There are many plugin managers for vim. I've only used `VimPlug`, but others exist. In my understanding, having multiple plugin managers for your vim session is unwise due to conflicts.

After installing `VimPlug` the `PlugInstall` command is available for use with installing and configuring plugins, many of which add additional commands and help documentation to vim.

Scripting

This can be added later, but is outside my personal use for vim.

Footnotes:

References:

This content based on the Taggart Technical Institute's Vim for Everyone course for personal use:
https://taggartinstitute.org/p/vim-for-everyone

I found this cheat sheet helpful for specific commands, but it's too messy to be relied on as a good cheat sheet:
https://www.fprintf.net/vimCheatSheet.html

A good write-up on vim registers:
https://www.brianstorti.com/vim-registers/

A useful tool for building your `.vimrc`:
https://vimrc-builder.vercel.app/

A somewhat authoritative list of vim plugins:
https://vimawesome.com/

Knowledge-base templates based on **M-Nelly** and **Alchemer**'s hard work. Stolen with permission from true friends.