Lecture 3: Editors (Vim)

Modal editing

Vim has multiple operating modes:

- **Normal**: for moving around a file and making edits (Pressing `<ESC>` the escape key to switch from any mode back to normal mode)
- **Insert**: for inserting text (enter insert mode with `i`)

- **Replace**: for replacing text (replace mode with `R`)
- **Visual** (plain, line, or block) mode: for selecting blocks of text (visual mode with `v`, visual line mode with `V`, visual block mode with `<C-v>`)
- **Command-line**: for running a command (command-line mode with `:`)

Basics

Buffers, tabs, and windows

Vim maintains a set of open files, called "buffers". A Vim session has a number of tabs, each of which has a number of windows (split panes). Each window shows a single buffer.

By default, Vim opens with a single tab, which contains a single window.

Command-line

- `:q` quit (close window)
- `:w` save ("write")
- `:wq` save and quit
- `:e` {name of file} open file for editing
- `:ls` show open buffers
- `:help {topic}` open help
  - `:help :w` opens help for the :w command
  - `:help w` opens help for the w movement

Vim's interface is a programming language

Movement

- Basic movement: `hjkl` (left, down, up, right)
- Words: `w` (next word), `b` (beginning of word), `e` (end of word)
- Lines: `0` (beginning of line), `^` (first non-blank character), `$` (end of line)
- Screen: `H` (top of screen), `M` (middle of screen), `L` (bottom of screen)

- Scroll: `Ctrl-u` (up), `Ctrl-d` (down)
- File: `gg` (beginning of file), `G` (end of file)
- Line numbers: `:{number}<CR>` or `{number}G` (line {number})
- Misc: `%` (corresponding item)
- Find: `f{character}`, `t{character}`, `F{character}`, `T{character}`
  - find/to forward/backward {character} on the current line
  - `,` / `;` for navigating matches
- Search: `/{regex}`, `n` / `N` for navigating matches

Edits

- `i` enter insert mode
  - but for manipulating/deleting text, want to use something more than backspace
- `o` / `O` insert line below / above
- `d{motion}` delete {motion}
  - e.g. `dw` is delete word, `d$` is delete to end of line, `d0` is delete to beginning of line
- `c{motion}` change {motion}
  - e.g. `cw` is change word
  - like `d{motion}` followed by `i`
- `x` delete character (equal do `dl`)
- `s` substitute character (equal to `xi`)
- visual mode + manipulation
  - select text, `d` to delete it or `c` to change it
- `u` to undo, `<C-r>` to redo
- `y` to copy / "yank" (some other commands like `d` also copy)
- `p` to paste
- `.` repetes previous changes

- Lots more to learn: e.g. `~` flips the case of a character

Counts

- `3w` move 3 words forward
- `5j` move 5 lines down
- `7dw` delete 7 words

Modifiers

You can use modifiers to change the meaning of a noun. Some modifiers are `i`, which means "inner" or "inside", and `a`, which means "around".

- `ci(` change the contents inside the current pair of parentheses
- `ci[` change the contents inside the current pair of square brackets
- `da'` delete a single-quoted string, including the surrounding single quotes

Customizing Vim

Vim is customized through a plain-text configuration file in `~/.vimrc` (containing Vimscript commands).

Extending Vim

- ctrlp.vim: fuzzy file finder
- ack.vim: code search
- nerdtree: file explorer
- vim-easymotion: magic motions

Check out Vim Awesome for more awesome Vim plugins.

Advanced Vim

Search and replace

`:s` (substitute) command (documentation).

- `%s/foo/bar/g`

    - replace foo with bar globally in file
- `%s/\[.*\](\(.*\))/\1/g`
    - replace named Markdown links with plain URLs

Multiple windows

- `:sp` / `:vsp` to split windows
- Can have multiple views of the same buffer.

Macros

- `q{character}` to start recording a macro in register `{character}`
- `q` to stop recording
- `@{character}` replays the macro
- Macro execution stops on error
- `{number}@{character}` executes a macro {number} times
- Macros can be recursive
    - first clear the macro with `q{character}q`
    - record the macro, with `@{character}` to invoke the macro recursively (will be a no-op until recording is complete)
- Example: convert xml to json (file)
    - Array of objects with keys “name” / “email”
    - Use a Python program?
    - Use sed / regexes
        - `g/people/d`
        - `%s/<person>/{/g`
        - `%s/<name>\(.*\)<\/name>/"name": "\1",/g`
        - ...
    - Vim commands / macros
        - `Gdd`, `ggdd` delete first and last lines
        - Macro to format a single element (register `e`)

    - Go to line with `<name>`
    - `qe^r"f>s": "<ESC>f<C"<ESC>q`
  - Macro to format a person
    - Go to line with `<person>`
    - `qpS{<ESC>j@eA,<ESC>j@ejS},<ESC>q`
  - Macro to format a person and go to the next person
    - Go to line with `<person>`
    - `qq@pjq`
  - Execute macro until end of file
    - `999@q`
  - Manually remove last `,` and add `[` and `]` delimiters

Resources

- `vimtutor` is a tutorial that comes installed with Vim
- Vim Adventures is a game to learn Vim
- Vim Tips Wiki
- Vim Advent Calendar has various Vim tips
- Vim Golf is code golf, but where the programming language is Vim's UI
- Vi/Vim Stack Exchange
- Vim Screencasts
- Practical Vim (book)