Writing and editing text files with VIM

These slides were made using VIM

UCLouvain

What is VIM ?

VIM is a terminal-based text editor:

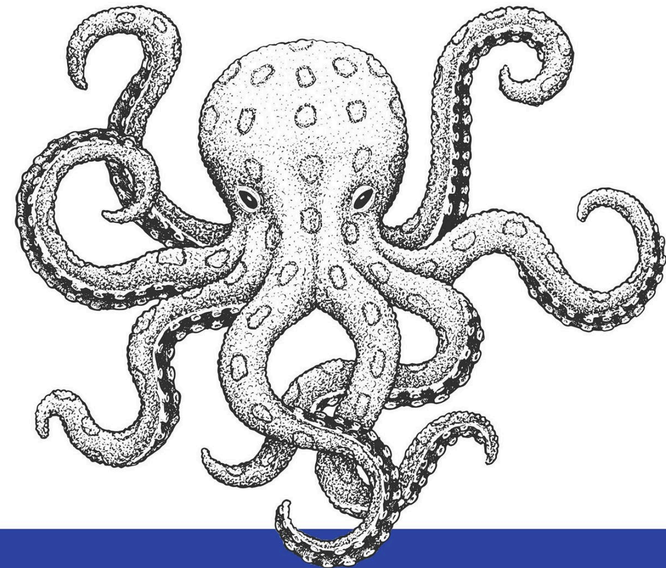
- Born in 1991, successor of VI (1977)
- Written in C and in VIM-script
- Free and free
- Actively maintained and developed
- One of the most popular terminal-based text editors

Why should I use it ?

- Lightweight
- Available on all Linux distributions
- Powerful commands system
- Easy configuration
- Plugin system
- Commands system is available in many other editors (VSC, Sublime, ...)

Getting in and out

- Opening a file from the terminal:
  `$ vim my_file.py`
- Saving and quitting
  `:wq`
- Quitting without saving:
  `:q!`
- Opening another file in VIM:
  `:e other_file.py`



Getting a file to play with

You are encouraged to tests the commands during the tutorial.

You can use the `cthulhu.txt` file linked to the indico page:

Copy link address -> go to terminal and enter:

```
wget https://indico.cism.ucl.ac.be/event/149/contributions/164/attachments/240/507/cthulhu.txt
```

and then:

```
vim cthulhu.txt
```

Modes*

VIM is a **modular** editor. It has three (main) modes:

- *normal* : default mode, where you enter commands
- *edition* : where you type text
- *visual* : where you select portions of text with a visual highlight to apply commands on them

Start typing !

To start adding text you have to go from **normal** to **insertion mode**.

- `i/I` (insert): go to insertion mode at cursor/ beginning of line
- `a/A` (append): go to insertion mode after cursor/end of line
- `o/O` (open line) : start a new line in insertion mode below/above cursor
- `R` : go to **replace mode** (insert on top of current text)

Note: lots of VIM commands have meaningful upper/lowercase variations.

When you've typed the text you wanted, remember to use the **"esc"** key to get back to normal mode. **This should always be your default mode**.

Movement commands

VIM allows to navigate quickly through text using short commands in addition to the usual arrows, home, end, … keys.

- word-based (capitals ignore special characters):
  - `w/W` : first character of next word
  - `e/E` : end of word
  - `b/B` : beginning of word (previous word if already at beginning)
- sentence:
  - `(/)` : sentence start/end
- paragraphs/code blocks:
  - `{/}` : paragraph start/end

Movement commands

VIM allows to navigate quickly through text using short commands.

- lines:
  - `0`: beginning of **line**
  - `$`: end of **line**
  - `^`: beginning of **text in line**

Movement commands

VIM allows to navigate quickly through text using short commands.

- files:
  - `gg` : beginning of **file**
  - `G` : end of **file**
- others:
  - `gf` : open file with name under cursor
  - `gd` : go to the definition of the current object/variable

Movement commands

VIM allows to navigate quickly through text using short commands.

- Matching symbols (parentheses, brackets, ...):
  - `%` brings you to the matching sign
  - if not on a symbol, brings you to the next one in file
- Finding generic characters:
  - `f/F<x>`: place the cursor on the next/previous appearance of "x" in line
  - `t/T<x>`: place the cursor just before/after the next/previous "x" in line

Quiz !

You're writing your thesis, your cursor is **in the middle of a paragraph** and you want to **start a new paragraph** directly following the current one, what sequence would work ?

What if you want to add it at the beginning ?

Cut, copy, paste

In VIM, copying is called "yanking" and cutting is "deleting". The commands are:

- `d` for deleting/cutting
- `y` for yanking/copying
- `p` for pasting

You must specify what you want to delete/yank. The simplest are:

- `dd` : delete current line
- `yy` : yank current line
- `x` : delete current character

You can then use `p` anywhere to paste

Undo and redo*

- To undo last change, go to normal mode and use `u`
- To redo last undone changes, use `ctrl+r`

Note: by doing "undoes" and "redoes" you are creating a tree of states. You can go back and forth in time with `g-` and `g+`

Combination commands

Typical VIM commands are built like this:

```
<action><number?><movement>
```

For instance:

- `dw` will delete text until next word
- `y}` will copy the text until the end of the current paragraph

One can also use a number to extend the movement:

- `d5↓` will delete five lines going down

Quiz !

You have a csv file with lines like this:

```
"data1";"brol 1";"value 1";
"more data 2";"something 2";"value 2";
"test data 3 and some";"other thing 3";"value 3";
```

How would you exchange the content of the first and second column on one line ?

Changing text

Changing with `c`:

- `cc`: delete line and go to insertion mode
- `c5w`: delete 5 words and go to insertion mode

Inside and around (very useful):

- `ci)`: delete text inside parentheses and go to insertion mode
- `ya[`: yank text around (including) brackets

Note: with `i` and `a`, signs like `(` are **symbols, not movement commands**. To change a paragraph you just need to use `{c}` or `cap` (use `p` for "paragraph" or `s` for "sentence")

Quiz !

With your cursor inside a word, how would you move this word after the next word ?

You made a typo and exchanged two characters, how to you exchange them again (the cursor is on the first one) ?

Indentation

VIM will try to guess the proper indentation for your code.

If you need to change it you can use the `<` and `>` commands to de-indent and indent. As usual:

- `>>` will indent the current line
- `>3↓` will indent 3 lines going down
- `<}` will de-indent all lines until the end of the block/paragraph

You can also use the `=` command to apply auto-indentation:

- `gg=G` will go to the beginning and try to auto-indent until the end of the file

Useful options

Show line numbers:

```
:set number
```

Paste mode (bypass automatic indentation)

```
:set paste
```

Incremental search:

```
:set incsearch
```

These and many others can be set permanently in the `.vimrc` file.

Macros*

To repeat sequences of commands you can use macros:

- enter `recording` mode with `q<macro_id>` where `macro_id` can be any letter
- type the sequence of commands
- stop recording with `q`
- use the macro with `@<macro_id>`
- you can repeat macros with `<number>@<macro_id>`

Note: If you just want to repeat the last command, you can simply use `.`

Quiz !

You have an emails list file with 200 lines, looking like this:

```
Pierre Bieliavsky pierre.bieliavsky@uclouvain.be
Giacomo Luca Bruno giacomo.luca.bruno@uclouvain.be
Eduardo Cortina Gil eduardo.cortina@uclouvain.be
```

How would you use macros to make a csv from this file in te following shape ?

```
"Pierre Bieliavsky"; "pierre.bieliavsky@uclouvain.be"
"Giacomo Luca Bruno"; "giacomo.luca.bruno@uclouvain.be"
"Eduardo Cortina Gil"; "eduardo.cortina@uclouvain.be"
```

Searching

The search command is:

```
/<search_pattern>
```

Where the pattern can contain **regular expressions**. You can then use

- `n` to go to the next match
- `N` to go to the previous match

If your cursor is on the word you want to search (for instance a variable name), you can use `*` to start a search on it.

Searching is a **movement command**. You can do `d/vim` to delete everything until the first match of `vim`

Searching

Some useful regular expressions:

- `.` : any character
- `[a-z]`, `[1-4]` : ranges
- `(expression)+` : given expression once or more
- `(expression)*` : given expression zero times or more
- `(expression)?` : given expression zero or one time
- `(expression){m,n}` : given expression m to n times

Notes:

- Special characharers should be escaped with `\` as in `\.` for a literal `.`
- Parenthesis can be used to gather expressions if needed

Visual mode*

Visual mode allows you to **highlight text to perform actions on it**. There are three different visual modes:

- `v` : character mode (I almost never use it)
- `V` : line mode
- `ctrl+v` : block mode

Once in visual mode, you can use:

- movement commands to change the selection
- action commands to apply them on the selection (and go back to normal mode)

Block mode*

Block mode behaves a little differently than the other modes:

- `c` replaces the text on all lines of the block by copies of the same text
- `I` can be used to insert on all lines

Changes are only applied to lines when `esc` is hit. It can be used to easily move whole columns:

1. Highlight the desired column with `ctrl+v` + movement commands
2. Use `x` to delete the column and put it in the clipboard
3. Go to te desired location and use `p` to paste it.

Search and replace*

Substitution can be done using the following:

```
:s<scope>/<pattern>/<replacement>
```

Where `<scope>` is the area in which the substitution should happen. The simplest are:

- `%` : replace everywhere in file
- `3,5`: replace on lines 3 to 5 included

The easiest way is usually to

1. Do a visual selection of the scope
2. Start the `:s` command without a scope to substitute only in the selected area

Search and replace

The Pattern can contain regular expressions.

The commands allow options:

- `/g` : replace all occurrences, not only the first one
- `/c` : ask confirmation for each replacement
- `/i` : ignore case

These options can be combined:

```
:s<scope>/<pattern>/<replacement>/gc
```

Quiz !

You are working on your markdown notes and you realize that you have used `*` instead of `- [ ]` in the 20 lines of your todo-list: how do you fix that ? find one solution with visual blocks and one with search/replace.

```
* task 1 **very important**
* another task
```

To

```
- [] task 1 **very important**
- [] another task
```

Completion

VIM provides intelligent completion:

- `ctrl+p` : complete current word based on words in the current file.
- `ctrl+x → ctrl+l` : complete current line
- `ctrl+x → ctrl+f` : complete current word using existing file names

Once in completion mode, you can either continue typing to reduce options or use arrows to select among the proposed solutions. Use the `Enter` key to select an option

Bash commands

You can apply bash commands using:

```
:<scope>! <command> <arguments>
```

The scope is the same as for the substitute command.

This will run the command on the selected scope and replace the content by the output of the command. Examples:

- `:%! sort`: sort the whole file
- `:%! grep -v <pattern>`: remove lines matching <pattern>
- `:%! grep -o '[a-z\.\-]\+@[a-z\.]\+'`: only keep emails

Split-screen

You can open two files side-by-side with:

```
$ vim -O file_1.txt file_2.txt
```

Then you can go to the left/right part of the window with `ctrl+w ←/→`
If already in VIM, you can split the current editor window with:

- `:split other_file.txt` for horizontal split
- `:vsplit other_file.txt` fo vertical split

If you are in verical split, option `:set scrollbind` forces the pars to scroll together.

Special cases

You can edit files over ssh using:

```
vim scp://<user>@<server>/<relative path to file>
```

You can also view the content of zip files:

```
vim my_file.zip
```

You can also easily compare files:

```
vimdiff file_1.txt file_2.txt
```

Press "pause"

If you need to quit VIM and get back to the terminal temporarily, instead of quitting VIM you can pause it by using

```
ctrl+Z
```

It will suspend the VIM process and put it in th background. Once you want to get it back, simply use bash command

```
%
```

To bring it to the foreground.

Note: Be careful if you run other commands in the backround, your VIM session might not be the one you will bring back !

Plugins

VIM can be further enhanced with plugins. The easiest way to manage them is to use a dedicated system such as:

- Vundle (https://github.com/VundleVim/Vundle.vim)
- Pathogen (https://github.com/tpope/vim-pathogen)

These provide commands to install and update plugins.

Plugins

Plugins can help in several aspects:

- Provide better completion
- Manage "tags" over a larger coding project
- Check code for syntax errors
- Provide (even) better navigation/action commands
- Integrate with code versioning systems
- Open files at last position
- ...

Thank you for your attention !