EE2510 - Productivity Tools for IC Design. Spring 2024-25.
IIT Hyderabad

==

vi editor commonly used commands

- i - insert
- a - append (go to insert mode from next character to the cursor)
- A - append at the end of the line
- d - delete
- y - yank/copy
- p - put/paste/print next to the current line
- P - put/paste/print before the current line
- o - open a new blank line below the cursor in insert mode
- O - open a new blank line above the cursor in insert mode
- x - delete single character under cursor
- X - delete single character before cursor
- h, j, k, l - can be used to move the cursor left, down, up, and right. Sometimes the arrow keys do not work and also using these letters are sometimes faster in terms of finger movements
- u - undo a change
- U (or Ctrl+R) - redo a change
- r - replace a character under the cursor
- R - enter the replace mode and the characters will get replaced as we type. Esc takes us out.
- Esc Esc - pressing escape twice brings to the command mode
- cw - change word. Deletes the word under cursor and goes into insert mode
- dw - deletes the word under cursor
- dd - delete the current line
- ^ or 0 - move to beginning of a line
- $ - move to end of a line
- gg - move to top of file
- G - move to bottom of file

Add a number before any command to repeat the command that many number of times. For example, 10dd will delete 10 lines.

- :sp filename - open another file filename in horizontal split window
- :vsp filename - open file filename in vertical split window
- Ctrl+w+w - switch between different split windows in a round-robin manner
- :q - quit
- :w - write
- :num - go to line num
- :+num - go ahead num numbers of line
- :-num - go back num numbers of line
- :e filename - open the file filename in the buffer

- :n - when multiple files are open in the same buffer, switch to the next file
- :r filename - append the content of file filename into current file
- :term - open a terminal in a split pane
- :s - substitute (or replace). See details below.

Search/Substitute/replace commands

- /expression - search for expression (any string or regular expression) in the file
- * - search word under cursor in forwards
- # - search word under cursor in backwards
- :%s/expression/string/ - replace the expression with string throughout the file
- :%s/expression/string/g - replace the expression with string, even for multi-occurrences in a line
- :%s/expression/string/c - replace the expression with string, confirm before each replacement
- :%s/expression/string/i - replace the expression (case insensitive match) with string
- A combination of qualifiers g,i,c can be used with the substitute/replace command above

Settings commands

- :set ic or noic - turn on or off the case insensitive search
- :set nu or nonu - turn on or off the line number view in the editor
- :set wrap or nowrap - turn on or off the text wrap for longer lines
- :set hls - highlight search
- Default desired settings could be configured in a file named .vimrc in your home directory.

Vim registers: numbered:

- Register 0 - copied line
- Register 1 - deleted line
- Registers 2-9 - rotates the previous deleted lines
- Register a-z - can be used for independent clipboards to store copied text or keystrokes

- "REGyy - copy the line into register REG (could be any letter)
- "REGp - paste the line from register REG
- :registers - show the contents of registers

Keystroke recording:

- qREG - start recording into register REG, pressing ESC q again stops the recording
- @REG - replay the keystrokes from register REG
- 100@REG - replay 100 times

Misc:

- vi could also be used as default editor inside vscode through some configuration changes in vscode
- There is also a graphical version of vi named gvim

- Visual block: Ctrl+v
    - Ctrl+v followed by arrow-keys can help select a portion of the text for copy/paste, column-wise selection of text, etc.
    - It can also be used to copy-paste a column before or after another region of text
    - It can be used to insert some prefix before a particular position in each line (insert a new column of text)
-

Regular Expressions (regex)

In the context of editors or pattern searching, regular expressions denote any string consisting of a sequence of characters (alphabetic, numeric, special). The entire expression could be specified as a specific sequence or using certain wildcard symbols to allow a more generic representation. Here, we try to discuss regular expressions in general, but use vim regex as a specific example.

Character class: A character class defines a set of characters which we wish to match during our search. In vim, a character class is specified within square brackets. e.g., [abc12] will match any of a, b, c, 1, 2. Some generic (pre-defined) character classes are as follows:

- \ℓ - [a-z] - any lower case alphabet letter
- \u - [A-Z] - any upper case alphabet letter
- \a - [a-zA-Z] - any lower or upper case alphabet letter
- \d - [0-9] - any decimal digit
- \w - [0-9a-zA-Z_] - any alphanumeric character or underscore
- \s - any whitespace character (space or tab)
- . - any character

Apart from these, any custom class/set can be used by enclosing them in the brackets. For example, [aeiou] will match any vowel; [1,3,5,7,9] will match any odd digit, [b-f] will match letters b, c, d, e, f.

Using a ^ symbol as the first character in the character set negates the selection and will match everything except the characters in the class. e.g., [^0-9] will match everything except digits; [^A-Z0-9] will match everything except capital letters or digits. Some generic (pre-defined) negated character classes are as follows:

- \L - [^a-z] - except any lower case alphabet letter
- \U - [^A-Z] - except any upper case alphabet letter
- \A - [^a-zA-Z] - except any lower or upper case alphabet letter
- \D - [^0-9] - except any decimal digit
- \W - [^0-9a-zA-Z_] - except any alphanumeric character or underscore
- \S - everything except any whitespace character (space or tab)

Examples:

- A capital letter followed by small letter - \u\ℓ or [A-Z][a-z]
- A small letter followed by a number - \ℓ\d or [a-z][0-9]
- A 3-digit number followed by a space - \d\d\d\s
- …

Pattern quantifiers: They can be used to indicate the count of a particular character or a class to be matched.
e.g., If we want to match all 10-digit telephone numbers, one option is to match 10-digits in sequence - \d\d\d\d\d\d\d\d\d\d. But this might be inconvenient and error prone to write. Instead, we have qualifiers to specify the count during matching. Following qualifiers are used in vim:

- * - zero or more occurrences of the previous character
- \+ - one or more occurrences of the previous character
- \? - zero or one occurrences of the previous character
- \{min, max\} - min to max occurrences of the previous character, both inclusive
  - By default, if min is not specified, it is 0. e.g., \{,max\} matches from 0 to max
  - By default, if max is not specified, it is infinity. e.g., \{min,\} matches from min to infinity
- \{num\} - match exactly 'num' number of times.

The same sequence to match 10 digit telephone number can now be written as \d\{10\}
Examples:

- Match any UG student of ee23 batch - ee23btech\d\{5\}
- Match any EE'22 or EE'23 student of IITH - ee2[23]btech\d\{5\}
- Match any CS course of IITH - CS\d\{4\}
- Match any word starting with capital letter followed by 1 or more small letters - \u\l\+
- Match any non-blank line - \w\+

Anchors/Positions: These can be used to additional qualify the position of the search pattern. Following are some common anchor positions:

- ^ - match starting of line
- $ - match end of line
- \< - match the beginning of a word boundary
- \> - match the ending of a word boundary

Matching multiple expressions:

- Using \|
- e.g., search foo or bar - foo \| bar

Grouping:

- The pattern quantifiers apply only to previous character. What if we want to check the repetition of more than one character. e.g., search a sequence abababababab…
- or, Let's say we want to match a string where any two lowercase letters are repeated. E.g., aa or bb or cc … zz
- These things can be handled through grouping of expressions. Any sub-expression inside a set of parentheses form a group. e.g., \(abc\)def searches for abcdef, but since abc is within parentheses, it is a group and can be quantified using the quantifiers. For example, to search for abcabcdef, we can write \(abc\)\{2\}def

- Depending on the order in which these groups appear in the expression, they can be referred to using \1, \2, \3, … where \1 refers to the match within the first group, and so on. \1, \2, etc. are called back-references.
- If we want to match repeated occurrences of the matched term, we can use such back-references. e.g., to match aa, bb, … zz; we can use \([a-z]\)\1. The term [a-z] will match any lower case letter and \1 refers to the matched letter within the parentheses. Thus, the expression indicate repeated characters and will match aa, bb, … zz.
- Back-references can also be used during replace - e.g., we want to duplicate all the characters in a file like abcd will become aabbccdd
    - :%s/\([a-z\)/\1\1/g
- Please note that the true formal definition of regular expressions has more restrictions than what is supported in vi. Essentially, they DO NOT support back-references.

Regular expressions are also useful in other frameworks like grep, sed, awk. Python also has an in-built support for regular expressions. Most of them follow similar kind of notation as we discussed, but there might be small variations. However, the general concept still holds and is very useful.