Vim

editing, regex, text

<panel title="Macros">

Record a macro into register a:

qa

When done with adding actions to the macro stop the recording:

q

Execute the macro with

@a

Execute the previously executed macro again with

@@

</panel>

<panel title="Regex">

==== Capitalize the first letter of every word ====

s/\<./\u&/g

==== Lookahead / Lookbehind ====

|\@<= | positive lookbehind | |\@<! | negative lookbehind | |\@= | positive lookahead | |\@! | negative lookahead |

Positive lookahead

foo bar
foo blubb

To substitue only *foo* that is followed by ' **blubb**' with *bla*

:%s/foo\(blubb\)\@=/bla/

negative lookahead

foo bar
foo blubb

To substitue *foo* that is not followed by ' **blubb**' with *bla*

:%s/foo\(blubb\)\@!/bla/

Positive lookbehind

To substitute *bar* that is preceded by '**foo** ' with *blubb*

foo bar
blubb bar

:%s/\(foo \)\@⇐bar/blubb/

Negative lookbehind

To substitute *bar* that is not preceded by '**foo** ' with *bla*

foo bar
blubb bar

:%s/\(foo \)\@<!bar/bla/

==== substitute() ==== It is possible to call vim's substitute function. In contrast to the substitute command the function can be used to modify a register's content.

let @a='This is a test for substitute' let @a=substitute(@a, '$', '()',) `The first` *`let`* `instruction saves the string into register` **`a`**`. The second` *`let`* `instruction shows` *`substitute()`*`'s usage. It needs 4 arguments: - The expression the function is applied to - The pattern - The substitution - and flag, which can be one of` **`'g'`** `or` **`''`** `In this case the expression was register a, pattern was the end of line` **`$`** `that was substituted with` **`()`** `so if you now instruct vim to` **`echo @a`** `then "`*`This is a test for substitute()`*`" gets displayed. </panel> <panel title="Settings"> ==== Syntax highlighting ==== Syntax highlighting in general can be controlled by` **`:set syntax=[on|off]`** `temporarely on vim's cmdline or vimrc file. In shell scripts variables are sometimes dark blue for example. Depending on several circumstances this can be very hard to read. It is possible to change vim's behavior of highlighting things with the` **`hi[light]`** `command. Variables's look can be controlled with the` **`PreProc`** `hightlight group and the groups arguments. :hi PreProc ctermfg=28 Would set the color of the PreProc group to green - 28 in terms of 256 color table. * :help hi * :help attr-list * :help cterm * :help term <note tip> Sometimes (especially in the case that you use vim in tmux) it can be very helpful to configure` **`set t_Co=256`** [1] `</note> </panel> <panel title="Snippets"> ==== Create directories for backups, swapfiles and undofiles ==== <code vim> for f in ['backups', 'swapfiles', 'undofiles'] if !isdirectory($HOME . "/.vim/" . f) call mkdir($HOME . "/.vim/" . f , "p", 0700) endif endfor </code> This - combined with <code vim> set backupdir=$HOME/.vim/backups set directory=$HOME/.vim/swapfiles set undodir=$HOME/.vim/undofiles </code> ensures that vim collects backupfiles, swapfiles and undofiles in single locations if backups, swapfiles and undofiles are` **`not`** `deactivated by <code vim> set noundofile set nobackup set noswapfile </code>`

This can lead to filename collisions

</panel> </accordion>

1) this is a matter of bce: background color erase. You may wish to read this thread