Substitude - Command

```
:range s[ubstidate]/pattern/string/cgiI
```

- `c` confirm each substitution
- `g` global: replace all occurrences in the line (without g only first)
- `i` ignore case for search pattern
- `I` don't ignore case for the pattern

The substitude (search and replace) and the search command are the commands where regular expressions can be used from which the substitude command might be used more frequently.

Let's describe the `range` option of the `:s[ubstitude]` command more in detail (the `[` and `]` brackets mean that this part of the command name can be omitted. Typing `:s` is just enough).

range

number	an absolute number
.	the current line (default)
$	the last line in a file
%	the whole file, same as 1,$
't	position of mark t
/pattern[/]	next line where pattern matches
?pattern[?]	previous line where pattern matches
\/	the next line where the previously used search pattern matches
\?	the previous line where the previously used search pattern matches
\&	the next line where the previously used substitute pattern matches.

If no line range is specified the command will operate on the current line only. Each specifier can be followed by + or - and an optional number. This number is added or subtracted from the preceding identifier. If the number is omitted, 1 is used.

`:10,20` from line 10 to line 20

`:/Section 1/+,/Section 2/-`

all lines between Section 1 and Section 2, non-inclusive

The `/pattern[/]` and `?pattern[?]` may be followed by another address separated by a semicolon. This tells vim to find the first pattern and then continue searching for the second pattern from that on.

`:/section 1/;/subsection/+,/subsection/-`

find Section 1 then perform the operation from one line after the first occurrence of subsection (starting from Section 1) to the line before the following occurrence of subsection.

`:/section/+ y` this will search for section and yank one line after into the yank register

`://normal p` this will reuse the previous search pattern and put (paste) the text from the default register into a new line

Pattern

This section is a description of the possibilities to describe a search pattern.

Anchors: Anchors are usefull when the search pattern is bound to the boundaries of a word or an indention. P. ex. we want to replace every occurrence of `vi` with `VIM`. The approach

```
:s/vi/VIM/g
```

might do the job but it also changes navigator to navVIMgator.

We need a way to tell vim that our search starts at the beginning of a word and ends at the end of a word or maybe a line. The following anchors are available for this task:

`\<`	beginning of a word
`\>`	end of a word
`^`	beginning of a line
`$`	end of a line

So for our example

```
:s/\<vi\>/VIM/g
```

does what we want.

Metacharacters Some Metacharacters, also called "Escaped Characters" represent a group of characters. That makes regular Expressions in certain ways shorter.

.	any character except new line
\s	whitespase
\S	non - whitespace
\d	digid ([0-9])
\D	non - digit
\x	hex digit
\X	non - hex digit
\o	octal digit
\O	non - octal digit
\h	head of word character ([a-z A-Z _])
\H	non - head of word
\p	printable character
\P	like \p but excluding digits
\w	word character ([0-9 a-z A-Z])
\W	non - word character

\a alphabetic character ([a-z A-Z])

\A non-alphabetic character

\e lowercase character ([a-z])

\L non-lowercase character

\u uppercase character ([A-Z])

\U non-uppercase character

To match a Date like 09/05/1995 you can use

/\d\d\/\d\d\/\d\d\d\d/

Quantifiers Sometimes a character can appear multiple times in a row. Maybe we even don't know how often it appears exactly or it might be a character that can appear but doesn't have to. In this kind of chases quantifiers can help out, creating a RegEx that covers this in its specifications.

`*` matches any number of occurrences of the preceding character, range or metacharacters.

`.*` matches everything, even an empty line.

`\+` matches 1 or more of the preceding characters

`\=` matches 0 or more of the preceding characters

`\{n,m}` matches from n to m of the preceiding characters

`\{n}` matches exactly n times of the preceiding characters

`\{,m}` matches at most m (including 0) of the preceding characters

`\{n,}` matches at least n of the preceiding characters

To match a word that begins with an uppercase letter followed by 5 lowercase letters we can use

```
\u\e{5}
```

The problem that comes with these quantifiers is that they are greedy: they try to match as much text as possible. If we want to match some text in between brackets we cannot use

```
/(.*)/
```

because the dot will also match the closing bracket and everything that follows. For this case there are also non-greedy versions of the quantifiers: they match only as much as necessary.

`\{-}` matches 0 or more of the preceeding atom, as few as possible.

`\{-n,m}` matches n to m of the preceeding atom, as few as possible

`\{-n,}` matches at least n or more of the preceeding atoms

`\{-,m}` matches 0 to m of the preceeding characters, as few as possible

So to find anything within brackets we could do this with:

```
/(.\{-})/
```

This works just fine. Just the Reg Ex

```
/.\{-}/
```

on its own is pretty useless because it matches everything, even empty lines, like

```
/.*/
```

does.

Character ranges

Within squared brackets we can define a set of characters of which we want to match one. p. Ex.

```
[012345]
```

matches any digit from zero to five but it only matches a single character out of this list.

`/[1234]/` is not the same as `/1234/` and the order of characters within the brackets doesn't matter at all, `[456]` is equal to `[465]` or `[564]` or anything similar.

Instead of writing out all the single numbers we can use the dash `-` operator to specify a range: `[0-5]` is the same as `[012345]`. This also works with letters in ascii order:

`[a-z]` matches lowercase letters

`[A-Z]` matches uppercase letters

`[a-zA-Z]` matches uppercase and lowercase

Depending on the system locale this even works with special letters like ø, ß, ö and other non-ascii characters.
If you want to include the dash - in your list then you have to put it at the beginning of the list:

```
/[-0-9]/
```

because that's the only place where it won't be treated as the range operator.
Sometimes it's easier to specify a list of characters that shouldn't be included. To do so a ^ at the beginning of the list indicates that the list should be negotiated.

`/[^0-9]/` matches everything that is not a pattern from `[0-9]`.

Let's talk about a useful example: find all sentences that start with a lowercase letter (because this is a grammar mistake):

`/\.\s\+[a-z]/` find all periods followed by one or more blanks and a lowercase word.

Grouping and Backreferencing

To perform substitutions it is usefull to refer back to the results the pattern matched. parts of the pattern can be grouped by putting it in between `\(` and `\)`. These groups can later be referenced as `\n` where `n` is a number from 1 to 9. `\0` always refers to the whole match of the pattern, `\1` to the first group until `\9` to the 9th group. A useful example might be the following one to swap the first two words of the line:

`:s/\(\w\+\)\(\s\+\)\(\w\+\)/\3\2\1/`

where `\1` holds the first word, `\2` any number of spaces in between and `\3` the second word.

`&`, `\0`	the whole matched pattern
`\1` to `\9`	group 1 to 9
`~`	the previous substitute string
`\L`	the following characters are made lowercase
`\U`	the following characters are made uppercase
`\E`, `\e`	end of `\U` and `\L`
`\r`	split line in two at this point
`\l`	next character made lowercase
`\u`	next character made uppercase

All these special characters can be used in the replacement part of the `:s[ubstitute]` command.

Now the search and replace to correct lowercase letters at the beginning of a sentence to uppercase looks like this:

```
:s/\([.!?]\)\s\+\([a-z]\)/\1 \u\2/g
```

space

This regex also replaces a variable number of spaces between sentences with only a single one.

Alternations

Using `\|` several expressions can be combined into one which matches any of its components. The first one matched will be used.

```
\(Date:\|Subject:\|From:\)\(is.*\)
```

will parse various mail headers into `\1` and `\2`. Vim alternations are not greedy, they won't search for the longest match and take the first that matched. Therefore the order of items in an alternation is important.

Operador Precedence

1. grouping `\(` `\)`
2. quantifiers `\=`, `\+`, `*`, `\{n}`, ...
3. metacharacters `\w`, `\d`, `\s`, ...
4. alternation `\|`

Global command

Global search and execution: The global command can search through the buffer and execute a command:

: range g[lobal][!] /pattern/ cmd

-> executes the command cmd everywhere where pattern matches. If pattern with a ! then execute the command only where the match does not occur.

The command searches within the range of lines and marks each as occured. In a second step it executes all the commands. If a line is changed or deleted its mark disappears. The command cmd can only be an Ex command (:s[ubstitute], :co[py], :d[elete], :w[rite], etc.). Non-Ex commands can be executed the following way: (normal mode)

: norm[al] non-ex-cmd

The following example deletes all empty lines:

```
:g/^$/d
```

Other examples:

`:g/^$/,/./-j` reduces multiple blank lines to a single one.

`:'a,'b g/^Error/ . w >> Errors.txt` find all lines starting with „Error“ and append these to Errors.txt. The dot . preceiding the w[rite] command tells it to only write the current file, not the whole buffer, for every error.