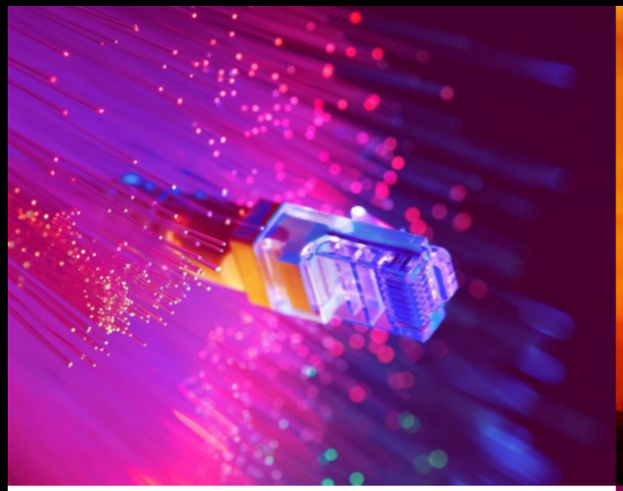

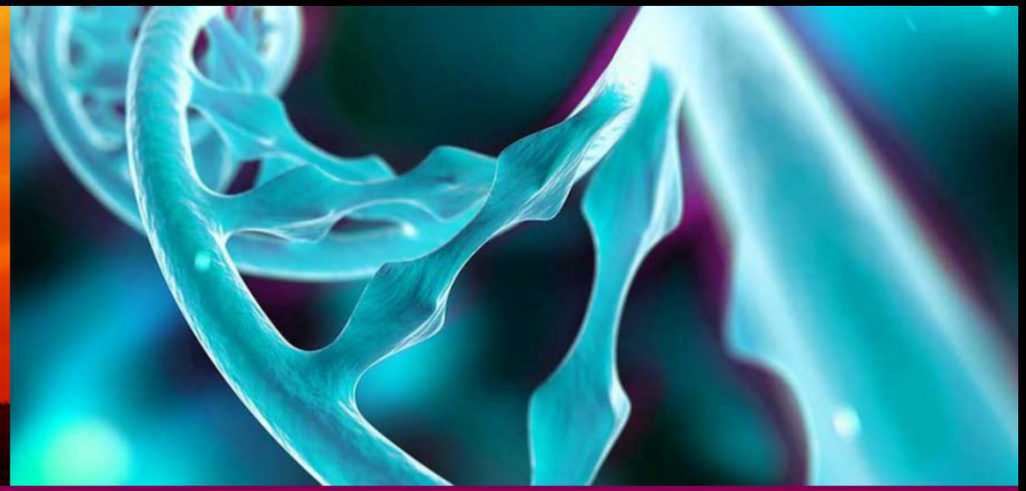

Regular expressions — a matching game

CSC Training, 2019-12


CSC – Finnish expertise in ICT for research, education and public administration

Matching text

- A number of Unix text-processing utilities let you search for, and in some cases change, text strings.
  - These utilities include the editing programs `ed`, `ex`, `vi` and `sed`, the `awk` programming language, and the commands `grep` and `egrep`.
- *Regular expressions* — or *regexes* for short — are a way to match text with patterns.
- Regular expressions are a pattern matching standard for string parsing and replacement.

The most simple regex

- In it’s simplest form, a regular expression is a string of symbols to match “as is”.

Regex	Matches
abc	**abc**def
234	1**234**5

```
$ grep '234'
```

Quantifiers

- To match several characters you need to use a quantifier:
  - `*` matches any number of what's before it, from zero to infinity.
  - `?` matches zero or one of what's before it.
  - `+` matches one or more of what's before it.

Regex	Matches
`23*4`	1**24**5, 1**234**5, 1**2334**5
`23?4`	1**24**5, 1**234**5
`23+4`	1**234**5, 1**2334**5

```
$ grep '23*4'
```

Basic regexes vs. extended regexes

- The Basic Regular Expressions or *BRE flavor* standardizes a flavor similar to the one used by the traditional UNIX `grep` command.
  - The only supported quantifiers are `.` (dot), `^` (caret), `$` (dollar), and `*` (star). To match these characters literally, escape them with a `\` (backslash).
  - Some implementations support `\?` and `\+`, but they are not part of the POSIX standard.
- Most modern regex flavors are extensions to the BRE flavor, thus called *ERE flavor*. By today's standard, the POSIX ERE flavor is rather bare bones.
- We will be using extended regexes, so:

```
$ alias grep='grep --color=auto -E'
```

Regexes are hoggish

- By default, regexes are greedy. They match as many characters as possible.

Regex	Matches
2	1**2222**3

- You can define how many instances of a match you want by using ranges:
  - `{m}` matches only *m* number of what's before it.
  - `{m,n}` matches *m* to *n* number of what's before it.
  - `{m,}` matches *m* or more number of what's before it.

Special characters

- A lot of special characters are available for regex building. Here are some of the more usual ones:
  - `.` matches any single character.
  - `\w` matches an alphanumeric character, `\W` a non-alphanumeric.
  - `\` to escape special characters, e.g. `\.` matches a dot, and `\\` matches a backslash.
  - `^` matches the beginning of the input string.
  - `$` matches the end of the input string.

Special character examples

Regex	Matches	Does not match
`1.3`	**123**4, **1z3**, 0**133**	13
`1.*3`	**13**, **123**, **1zdfkj3**	
`\w+@\w+`	**a@a**, **email@oy**.ab	,.-!"#€%&/
`^1.*3$`	**13**, **123**, **1zdfkj3**	x13, 123x, x1zdfkj3x

Character classes

- You can group characters by putting them between square brackets. This way, any character in the class will match any *one* character in the input.
  - `[abc]` matches any of a, b, and c.
  - `[a-z]` matches any character between a and z.
  - `[^abc]` matches anything other than a, b, or c.
    - Note that here the caret `^` at the beginning indicates “not” instead of beginning of line.
  - `[+*?.]` matches any of +, *, ? or the dot.
    - Most special characters have no meaning inside the square brackets.

Character class examples

Regex	Matches	Does not match
`[^ab]`	**c**, **d**, ab**c**, **s**a**dv**b**cv**	a, b, ab
`^[1-9][0-9]*$`	**1**, **45**, **101**	0123, -1, a1, 2.0
`[0-9]*[,.]?[0-9]+`	**1**, **.1**, **0.1**, **1,000**, **0,0,0.0**	

Grouping and alternatives

- It might be necessary to group things together, which is done with parentheses `(` and `)`.

Regex	Matches	Does not match
`(ab)+`	**ab**, **abab**, a**ab**b	aa, bb

  - Grouping itself usually does not do much, but combined with other features turns out to be very useful.

- The OR operator `|` may be used for alternatives.

Regex	Matches	Does not match
`(aa|bb)+`	**aa**, **bbaa**, **aabb**	abab

Subexpressions

- With parentheses, you can also define subexpressions to store the match after it has happened and then refer to it later on.

Regex	Matches	Does not match
`(ab)\1`	**abab**cdcd	ab, abcabc
`(ab)c.*\1`	**abcabc**, **abcdefab**cdef	abc, ababc

Some practical (?) examples

- Check for a valid format for email address:

```
$ grep '[A-Za-z0-9_-][A-Za-z0-9_.-]*[^.]@[A-Za-z0-9][A-Za-z0-9.-]+\.[A-Za-z]{2,}'
```

  - `[A-Za-z0-9_-][A-Za-z0-9_.-]*[^.]` matches a positive number of acceptable characters not starting or ending with dot.
  - `@` matches the @ sign.
  - `[A-Za-z0-9][A-Za-z0-9\.-]+` matches any domain name, incl. dots.
  - `\.[A-Za-z]{2,}$` matches a literal dot followed by two or more characters at the end.

- Check for a valid format for Finnish social security number:

```
$ grep '[0-9]{6}[-+A][0-9]{3}[A-Z0-9]'
```