

Basic Regular Expressions in R

Cheat Sheet

Character Classes

<code>[[:digit:]]</code> or <code>\\d</code> Digits; [0-9]
<code>\\D</code> Non-digits; <code>[^0-9]</code>
<code>[[:lower:]]</code> Lower-case letters;
<code>[[:upper:]]</code> Upper-case letters;
<code>[[:alpha:]]</code> Alphabetic characters;
<code>[[:alnum:]]</code> Alphanumeric characters [A-z0-9_]
<code>\\w</code> Word characters; [A-z0-9_]
<code>\\W</code> Non-word characters
<code>[[:xdigit:]]</code> or <code>\\x</code> Hexadecimal digits;
<code>[[:blank:]]</code> Space and tab
<code>[[:space:]]</code> or <code>\\s</code> Space, tab, vertical tab, newline, form feed, carriage return
<code>\\S</code> Not space; <code>[^[:space:]]</code>
<code>[[:punct:]]</code> Punctuation characters; <code>!"#\$%&'()*+,-./:;<=>?@[\\]^_`{ }~</code>
<code>[[:graph:]]</code> Graphical characters;
<code>[[:alnum:]][[:punct:]]</code>
<code>[[:print:]]</code> Printable characters;
<code>[[:alnum:]][[:punct:]][\\s]</code>
<code>[[:cntrl:]]</code> or <code>\\c</code> Control characters; <code>\\n, \\r</code> etc.

Special Metacharacters

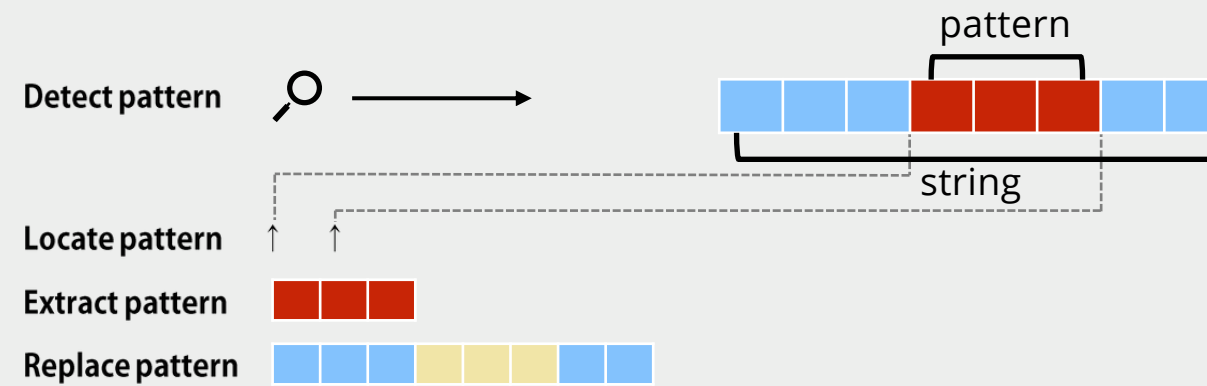
<code>\\n</code>	New line
<code>\\r</code>	Carriage return
<code>\\t</code>	Tab
<code>\\v</code>	Vertical tab
<code>\\f</code>	Form feed

Lookarounds and Conditionals*

<code>(?=)</code>	Lookahead (requires PERL = TRUE), e.g. <code>(?=yx)</code> : position followed by 'xy'
<code>(?!)</code>	Negative lookahead (PERL = TRUE); position NOT followed by pattern
<code>(?<=)</code>	Lookbehind (PERL = TRUE), e.g. <code>(?<=yx)</code> : position following 'xy'
<code>(?<!)</code>	Negative lookbehind (PERL = TRUE); position NOT following pattern
<code>?0 ifthen</code>	If-then-condition (PERL = TRUE); use lookaheads, optional char. etc in if-clause
<code>?0 ifthenelse</code>	If-then-else-condition (PERL = TRUE)

*see, e.g. <http://www.regular-expressions.info/lookaround.html>
<http://www.regular-expressions.info/conditional.html>

Functions for Pattern Matching



```
> string <- c("Hiphopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> pattern <- "t.m"
```

Detect Patterns

```
grep(pattern, string)
[1] 1 3
grep(pattern, string, value = TRUE)
[1] "Hiphopopotamus"
[2] "time for bottomless lyrics"
grepl(pattern, string)
[1] TRUE FALSE TRUE
stringr::str_detect(string, pattern)
[1] TRUE FALSE TRUE
```

Split a String using a Pattern

`strsplit(string, pattern)` or `stringr::str_split(string, pattern)`

Locate Patterns

```
regexpr(pattern, string)
find starting position and length of first match
gregexpr(pattern, string)
find starting position and length of all matches
stringr::str_locate(string, pattern)
find starting and end position of first match
stringr::str_locate_all(string, pattern)
find starting and end position of all matches
```

Extract Patterns

```
regmatches(string, regexpr(pattern, string))
extract first match [1] "tam" "tim"
regmatches(string, gregexpr(pattern, string))
extract all matches, outputs a list
[[1]] "tam" [[2]] character(0) [[3]] "tim" "tom"
stringr::str_extract(string, pattern)
extract first match
stringr::str_extract_all(string, pattern)
extract all matches, outputs a list
stringr::str_extract_all(string, pattern, simplify = TRUE)
extract all matches, outputs a matrix
stringr::str_match(string, pattern)
extract first match + individual character groups
stringr::str_match_all(string, pattern)
extract all matches + individual character groups
```

Replace Patterns

```
sub(pattern, replacement, string)
replace first match
gsub(pattern, replacement, string)
replace all matches
stringr::str_replace(string, pattern, replacement)
replace first match
stringr::str_replace_all(string, pattern, replacement)
replace all matches
```

Character Classes and Groups

- `.` Any character except `\\n`
- `|` Or, e.g. `(a|b)`
- `[...]` List permitted characters, e.g. `[abc]`
- `[a-z]` Specify character ranges
- `[^...]` List excluded characters
- `(...)` Grouping, enables back referencing using `\\N` where N is an integer

Anchors

<code>^</code>	Start of the string
<code>\$</code>	End of the string
<code>\\b</code>	Empty string at either edge of a word
<code>\\B</code>	NOT the edge of a word
<code>\\<</code>	Beginning of a word
<code>\\></code>	End of a word

Quantifiers

<code>*</code>	Matches at least 0 times
<code>+</code>	Matches at least 1 time
<code>?</code>	Matches at most 1 time; optional string
<code>{n}</code>	Matches exactly n times
<code>{n,}</code>	Matches at least n times
<code>{n,m}</code>	Matches between n and m times

General Modes

By default R uses *extended* regular expressions. You can switch to *PCRE regular expressions* using `PERL = TRUE` for base or by wrapping patterns with `perl()` for `stringr`.

All functions can be used with literal searches using `fixed = TRUE` for base or by wrapping patterns with `fixed()` for `stringr`.

All base functions can be made case insensitive by specifying `ignore.cases = TRUE`.

Escaping Characters

Metacharacters (`.` `*` `+` etc.) can be used as literal characters by escaping them. Characters can be escaped using `\\` or by enclosing them in `\\Q...\\E`.

Case Conversions

Regular expressions can be made case insensitive using `(?i)`. In backreferences, the strings can be converted to lower or upper case using `\\L` or `\\U` (e.g. `\\L\\1`). This requires `PERL = TRUE`.

Greedy Matching

By default the asterisk `*` is greedy, i.e. it always matches the longest possible string. It can be used in lazy mode by adding `?`, i.e. `*?`.

Greedy mode can be turned off using `(?U)`. This switches the syntax, so that `(?U)a*` is lazy and `(?U)a*?` is greedy.

Note

Regular expressions can conveniently be created using e.g. the packages `rex` or `rebus`.