**Regular Expression Syntax:**

|  |  |
| --- | --- |
| Syntax | Description |
| \\d | Digit, 0,1,2 ... 9 |
| \\D | Not Digit |
| \\s | Space |
| \\S | Not Space |
| \\w | Word |
| \\W | Not Word |
| \\t | Tab |
| \\n | New line |
| ^ | Beginning of the string |
| $ | End of the string |
| \ | Escape special characters, e.g. \\ is "\", \+ is "+" |
| | | Alternation match. e.g. /(e|d)n/ matches "en" and "dn" |
| • | Any character, except \n or line terminator |
| [ab] | a or b |
| [^ab] | Any character except a and b |
| [0-9] | All Digit |
| [A-Z] | All uppercase A to Z letters |
| [a-z] | All lowercase a to z letters |
| [A-z] | All Uppercase and lowercase a to z letters |
| i+ | i at least one time |
| i\* | i zero or more times |
| i? | i zero or 1 time |
| i{n} | i occurs n times in sequence |
| i{n1,n2} | i occurs n1 - n2 times in sequence |
| i{n1,n2}? | non greedy match, see above example |
| i{n,} | i occures >= n times |
| [:alnum:] | Alphanumeric characters: [:alpha:] and [:digit:] |
| [:alpha:] | Alphabetic characters: [:lower:] and [:upper:] |
| [:blank:] | Blank characters: e.g. space, tab |
| [:cntrl:] | Control characters |
| [:digit:] | Digits: 0 1 2 3 4 5 6 7 8 9 |
| [:graph:] | Graphical characters: [:alnum:] and [:punct:] |
| [:lower:] | Lower-case letters in the current locale |
| [:print:] | Printable characters: [:alnum:], [:punct:] and space |
| [:punct:] | Punctuation character: ! " # $ % & ' ( ) \* + , - . / : ; < = > ? @ [ \ ] ^ \_ ` { | } ~ |
| [:space:] | Space characters: tab, newline, vertical tab, form feed, carriage return, space |
| [:upper:] | Upper-case letters in the current locale |
| [:xdigit:] | Hexadecimal digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f |

## Finding Regex Matches in String Vectors

The **grep** function takes your regex as the first argument, and the input vector as the second argument. If you passvalue=FALSE or omit the value parameter then grep returns a new vector with the indexes of the elements in the input vector that could be (partially) matched by the regular expression. If you pass value=TRUE, then grep returns a vector with copies of the actual elements in the input vector that could be (partially) matched.

> grep("a+", c("abc", "def", "cba a", "aa"), perl=TRUE, value=FALSE) [1] 1 3 4

> grep("a+", c("abc", "def", "cba a", "aa"), perl=TRUE, value=TRUE) [1] "abc" "cba a" "aa"

The **grepl** function takes the same arguments as the grep function, except for the value argument, which is not supported. grepl returns a logical vector with the same length as the input vector. Each element in the returned vector indicates whether the regex could find a match in the corresponding string element in the input vector.

> grepl("a+", c("abc", "def", "cba a", "aa"), perl=TRUE) [1] TRUE FALSE TRUE TRUE

The **regexpr** function takes the same arguments as grepl. regexpr returns an integer vector with the same length as the input vector. Each element in the returned vector indicates the character position in each corresponding string element in the input vector at which the (first) regex match was found. A match at the start of the string is indicated with character position 1. If the regex could not find a match in a certain string, its corresponding element in the result vector is -1. The returned vector also has a match.length attribute. This is another integer vector with the number of characters in the (first) regex match in each string, or -1 for strings that didn't match.

**gregexpr** is the same as regexpr, except that it finds all matches in each string. It returns a vector with the same length as the input vector. Each element is another vector, with one element for each match found in the string indicating the character position at which that match was found. Each vector element in the returned vector also has a match.length attribute with the lengths of all matches. If no matches could be found in a particular string, the element in the returned vector is still a vector, but with just one element -1.

> regexpr("a+", c("abc", "def", "cba a", "aa"), perl=TRUE) [1] 1 -1 3 1 attr(,"match.length") [1] 1 -1 1 2

> gregexpr("a+", c("abc", "def", "cba a", "aa"), perl=TRUE) [[1]] [1] 1 attr(,"match.length") [1] 1 [[2]] [1] -1 attr(,"match.length") [1] -1 [[3]] [1] 3 5 attr(,"match.length") [1] 1 1 [[4]] [1] 1 attr(,"match.length") [1] 2

Use **regmatches** to get the actual substrings matched by the regular expression. As the first argument, pass the same input that you passed to regexpr or gregexpr . As the second argument, pass the vector returned byregexpr or gregexpr. If you pass the vector from regexpr then regmatches returns a character vector with all the strings that were matched. This vector may be shorter than the input vector if no match was found in some of the elements. If you pass the vector from regexpr then regmatches returns a vector with the same number of elements as the input vector. Each element is a character vector with all the matches of the corresponding element in the input vector, or NULL if an element had no matches.

>x <- c("abc", "def", "cba a", "aa")

> m <- regexpr("a+", x, perl=TRUE)

> regmatches(x, m) [1] "a" "a" "aa"

> m <- gregexpr("a+", x, perl=TRUE)

> regmatches(x, m) [[1]] [1] "a" [[2]] character(0) [[3]] [1] "a" "a" [[4]] [1] "aa"

## Replacing Regex Matches in String Vectors

The **sub** function has three required parameters: a string with the regular expression, a string with the replacement text, and the input vector. sub returns a new vector with the same length as the input vector. If a regex match could be found in a string element, it is replaced with the replacement text. Only the first match in each string element is replaced. If no matches could be found in some strings, those are copied into the result vector unchanged.

Use **gsub** instead of sub to replace all regex matches in all the string elements in your vector. Other than replacing all matches, gsub works in exactly the same way, and takes exactly the same arguments.

You can use the [backreferences](http://www.regular-expressions.info/replacebackref.html" \t "_top) \1 through \9 in the replacement text to reinsert text matched by a [capturing group](http://www.regular-expressions.info/brackets.html" \t "_top). You cannot use backreferences to groups 10 and beyond. If your regex has named groups, you can use numbered backreferences to the first 9 groups. There is no replacement text token for the overall match. Place the entire regex in a capturing group and then use \1 to insert the whole regex match.

> sub("(a+)", "z\\1z", c("abc", "def", "cba a", "aa"), perl=TRUE) [1] "zazbc" "def" "cbzaz a" "zaaz"

> gsub("(a+)", "z\\1z", c("abc", "def", "cba a", "aa"), perl=TRUE) [1] "zazbc" "def" "cbzaz zaz" "zaaz"

You can use \U and \L to change the text inserted by all following backreferences to uppercase or lowercase. You can use \E to insert the following backreferences without any change of case. These escapes do not affect literal text.

> sub("(a+)", "z\\U\\1z", c("abc", "def", "cba a", "aa"), perl=TRUE) [1] "zAzbc" "def" "cbzAz a" "zAAz" > gsub("(a+)", "z\\1z", c("abc", "def", "cba a", "aa"), perl=TRUE) [1] "zAzbc" "def" "cbzAz zAz" "zAAz"

A very powerful way of making replacements is to assign a new vector to the regmatches function when you call it on the result of gregexpr. The vector you assign should have as many elements as the original input vector. Each element should be a character vector with as many strings as there are matches in that element. The original input vector is then modified to have all the regex matches replaced with the text from the new vector.

> x <- c("abc", "def", "cba a", "aa")

> m <- gregexpr("a+", x, perl=TRUE)

> regmatches(x, m) <- list(c("one"), character(0), c("two", "three"), c("four")) > x [1] "onebc" "def" "cbtwo three" "four"

|  |  |
| --- | --- |
| **?** | The question mark indicates there is *zero or one* of the preceding element. For example, colou?r matches both "color" and "colour". |
| **\*** | The asterisk indicates there is *zero or more* of the preceding element. For example, ab\*c matches "ac", "abc", "abbc", "abbbc", and so on. |
| **+** | The plus sign indicates there is *one or more* of the preceding element. For example, ab+c matches "abc", "abbc", "abbbc", and so on, but not "ac". |