94- Regular expressions

Table of Contents

Introduction:....2
Table of Metacharacters in Basic and Extended REs:....2
Basic....2
Ext....2
Brief description:....2
Normal String matches:....2
Empty string matches:....2
Item Repetitions:....2
Logical Operators....2
Basic regular expressions....3
. 3
* 3
^....3
$ 3
\<....3
\>....3
[...]....3
\....3
Extended Regular Expressions....4
(...)....4
{...}....4
?....4
+....4
|....4
POSIX Character Classes....5
[:class:]....5
Notes:....5
Backslashed characters....5

Introduction:
A regular expression (RE) is a string of characters of which their interpretation is above and beyond their literal meaning. The are called **Metacharacters**.
The main uses for REs are text searches and string manipulation. A RE matches a single character or a set of characters (a substring or an entire string).
There are two types of regular expressions:

- Basic (older) expresions: like the ones used by `grep` and `sed`.
- Extended expression: like the ones used by `egrep`, `awk`, and Perl language.

Table of Metacharacters in Basic and Extended REs:

	.	*	^	\<	\>	\b	\B	$	[..]	\	(..)	{..}	+	?	|
Basic	x	x	x	x	x	x	x	x	x	x	\x	\x	\x	\x	\x
Ext	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

In Basic REs the metacharacters **? + { | (** and **)** lose their special meaning; instead use the backslashed versions **\? \+ \{ \| \(** and **\)**
eg. `\{1,5\}` in Basic REs is the same meaning as `{1,5}` in Extended REs

Brief description:
Normal String matches:

. matches any single character
[abt] matches one character only: either a or b or `t` and nothing else
[a-z] matches one character only: either a to `z` and nothing else
[^A-Z] matches one character only: any character but NOT A to Z
(hallo) matches the word 'hallo' as one Item (an atom). Normally used for repeats.

Empty string matches:

^ matches the beginning of a line
$ matches the end of a line
\< matches the beginning of a word
\> matches the end of a word
\b matches either the beginning or end of a word
\B matches NOT the beginning or end of a word

Item Repetitions: (item = character or an atom) Note: use `\{....\}` for grep & {....} for egrep

? The preceding item is optional and matched at most once.
***** The preceding item will be matched zero or more times.
+ The preceding item will be matched one or more times.
{n} The preceding item is matched exactly n times.
{n,} The preceding item is matched n or more times.
{n,m} The preceding item is matched at least n times, but not more than `m` times.

Logical Operators

| Allow to specify multiple REs that may match. OR operator.

Basic regular expressions(detailed)**:**

. **(dot) matches any one character, except a newline.**
eg. **13.** matches 13 + at least one of any character (including a space):
1133, 11333, but not 13 (additional character missing).

***** (asterisk) matches any number of repeats of the character string or Atom RE preceding it, including zero times.
eg. **1153*** matches 115 + none or one or more 3's + possibly other characters after. In this case it matches 115, 1153, 11151zF, and so forth.

^ (caret) matches the beginning of a line, but sometimes, depending on context, negates the meaning of a set of characters in an RE.
eg1. **^Hallo** matches `Hallo` appearing at the beginning of a line.
eg2. **[^0-9]** matches any one character that is NOT a digit from 0 to 9

$ (dollar sign) at the end of an RE matches the end of a line.
eg1. **barkley$** matches the word barkley at the end of a line.
eg2. **^$** matches blank lines.

\< (escaped smaller than) matches the beginning of a word
\> (escaped greater than) matches the end of a word
eg. `\<hallo\>` matches the words `hallo du` but not `hallodu`

[...] (brackets) enclose a set of characters to match in a single RE.
eg. **[xyz]** matches the char. x, y, or z.
[c-n] matches any of the char. in the range c to n.
[B-Pk-y] matches any of the char. in the ranges B to P and k to y.
[a-z0-9] matches any lowercase letter or any digit.
[^b-d] matches all char. except those in the range b to d.
This is an instance of ^ negating or inverting the meaning of the following RE (taking on a role similar to **!** in 'C')

Combined sequences of bracketed characters match word patterns.
eg1. **[Yy][Ee][Ss]** matches yes, Yes, YES, yEs, and so forth.
eg2. `[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}`
Matches network IP number.
`192.168.45.67` or `12.18.149.0 etc`

**** (backslash) escapes a special character(metacharacter), which means that character gets interpreted literally.
Theses characters and then said to be 'escaped'
eg1. **\$** reverts back to its literal meaning of "$", rather than its RE meaning of end of line.
eg2. **** has the literal meaning of "\"

Extended Regular Expressions. Used in `egrep`, `awk`, and Perl language:

`(...)` (parenteses) Declares its content as an 'Atom'. An atom is considered as one unit only, just like a single character. Normally used to match repeats.
eg. H(allo)* matches `H`, `Hallo`, `Halloallo`, `Halloalloallo` ect.

`{...}` (curly brackets) indicate the number of occurrences of a preceding RE to match. In Basic REs it is necessary to escape(\) the curly brackets since they have only their literal character meaning otherwise. eg. `\{...\}`

Maximal	**Minimal**	**Allowed Range**
{*n*,*m*}	{*n*,*m*}?	Must occur at least *n* times and max *m* times
{*n*,}	{*n*,}?	Must occur at least *n* times
{*n*}	{*n*}?	Must match exactly *n* times
*	*?	0 or more times (same as {0,})
+	+?	1 or more times (same as {1,})
?	??	0 or 1 time (same as {0,1})

eg. **`[0-9]{5}`** matches at least five consecutive digits:
(characters in the range of 0 to 9).
ie. 13649, 897507, 9866554 but not 1457b9654

Curly brackets are not available as an RE in the "classic" version of **awk**.
However, **gawk** has the **`-re interval`** option that permits them (without being escaped).
eg. **`echo 2222 | gawk -re interval '/2{3}/' 2222`**

? (question mark)matches zero or one of the previous character or atom.
It is generally used for matching single characters.
eg1. **`Hel?o`** matches a 3 or 4 character word like `Heo` and `Helo` but not `Hello`
eg2. **`H(allo)?du`** matches `Hdu`, `Hallodu`, but not `Hallotdu`

+ (plus) matches one or more of the previous character or atom.
It serves a role similar to the *****, but does not match zero occurrences.
eg1. **`hal+o`** Matches `hallo` or `hallllo` but not `hao`
eg2. **`H(all)+o`** Matches `hallo` or `Hallallo` but not `Ho`

GNU versions of `sed` and `awk` can use "+", but it needs to be escaped.
egs. **`echo a111b | sed -ne '/a1\+b/p'`**
`echo a111b | grep 'a1\+b'`
`echo a111b | gawk '/a1+b/'`
All of above are equivalent.

| (logical OR) matches multiple REs in a logical OR fashion.
eg. `hallo | beybey` Matches either `hallo` or `beybey` or both strings.

POSIX Character Classes.

[[:*class*:]] This is an alternate method of specifying a range of char. to match.
[[:alnum:]] Matches alphabetic or numeric characters.
This is equivalent to `[A-Za-z0-9]`.
[[:alpha:]] Matches alphabetic characters. This is equivalent to `[A-Za-z]`.
[[:blank:]] Matches a space or an horizontal tab.
[[:cntrl:]] Matches control characters. Ctrl-a to Ctrl-z
[[:punct:]] Matches any punctuation: all printable chararcters
except `0-9`, `A-Z`, `a-z` or *space*
ie. `°!"§$%&/()=?`´\}][{~+-*#'_.,:;|<>`
[[:digit:]] Matches (decimal) digits. This is equivalent to `[0-9]`.
[[:graph:]] (graphic printable characters). Matches characters in the range of ASCII 33-126. This is the same as `[:print:]`, below, but excluding the *space* character.
[[:lower:]] Matches lowercase alphabetic characters. Equivalent to `[a-z]`.
[[:print:]] (printable characters). Matches characters in the range of ASCII 32-126. Same as `[:graph:]`, above, but adding the *space*.
[[:space:]] matches whitespace characters (space and horizontal tab).
[[:upper:]] matches uppercase alphabetic characters. Equivalent to `[A-Z]`.
[[:xdigit:]] matches hexadecimal digits. This is equivalent to `[0-9A-Fa-f]`

Notes:

- POSIX character classes generally require quoting or double brackets **`[[ ]]`**.
  eg. **`grep [[:digit:]] test.file`**
  `abc=7`**`7`**`23`

- These character classes may even be used with globbing, to a limited extent.
  eg. **`ls -l ?[[:digit:]][[:digit:]]?`**
  `rw-rw-r-- 1 bozo bozo 0 Aug` **`21`** `14:47 a33b`

Backslashed characters

A backslashed letter matches a special character or character class:

Code	Matches
`\a`	Alarm (beep)
`\b`	Space Character
`\n`	Newline
`\r`	Carriage return
`\t`	Tab
`\f`	Formfeed
`\e`	Escape
`\d`	A digit, same as `[0-9]`
`\D`	A nondigit

Code	Matches
\w	A word character (alphanumeric), same as [a-zA-Z_0-9]
\W	A nonword character
\s	A whitespace character, same as [\t\n\r\f]
\S	A non-whitespace character

Note that \w matches a character of a word, not a whole word. Use \w+ to match a word.

- A backslashed single-digit number matches whatever the corresponding parentheses actually matched (except that \0 matches a null character). This is called a *backreference* to a substring. A backslashed multi-digit number such as \10 will be considered a backreference if the pattern contains at least that many substrings prior to it, and the number does not start with a 0. Pairs of parentheses are numbered by counting left parentheses from the left.
- A backslashed two- or three-digit octal number such as \033 matches the character with the specified value, unless it would be interpreted as a backreference.
- A backslashed x followed by one or two hexadecimal digits, such as \x7f, matches the character having that hexadecimal value.
- A backslashed c followed by a single character, such as \cD, matches the corresponding control character.
- Any other backslashed character matches that character.
- Any character not mentioned above matches itself.