GREP

Notebook:	Linux Command		
Created:	28/02/2018 11:32	**Updated:**	24/07/2018 12:31
Author:	nabilafr		

Tool for matching regular expressions against a text file, multiple files or a stream of input.

REGULAR EXPRESSIONS

A regular expression is a pattern.

Basic	Treats special characters as simple symbol. ?, +, {, |, (,) are treated like alphanumeric chars
Extended	Treats special characters as **operators.** Has to be preceeded by backslash \ to be treated as simple symbol. The "Period" (.) can match any single character.
perl	Has more extensive pattern functionality

Character Classes and Bracket Expr:sessions

[*ty0px%***]**	Match all individual characters enclosed in bracket in specified files. Here you will search for any occurances of t, y, 0, p, x and %.
[^*abcd***]**	Match any individual characters **NOT** enclosed in bracket in specified files. Here you will search for any character **other than** a, b, c and d.
[*a-d***], [**1-6**]**	Use hyphen to **match range of characters** a,b,c,d or 1,2,3,4,5,6 to file contents.
[***]abcd-[***]	Match [and]
[*abcd-***]**	Match the hyphen also
[*ab^cd***]**	Match the ^ also. All other punctuation not treated specially in this bracket.

Predefined Classes

Requires two brackets.

[[:alnum:]]	All Alphanumeric characters [0-9A-Za-z]
[[:alpha:]]	All alphabetic characters [A-Za-z]
[[:blank:]]	Blank characters "space" and "tab"
[[:cntrl:]]	Characters without symbols such as delete
[[:digit:]]	All digits [0-9]
[[:graph:]]	All Graphical characters + Punctuation
[[:lower:]]	All lower case alphabets [a-z]
[[:print:]]	All printable characters [:alnum:] [:punct:]
[[:punct:]]	All punctuation ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~.
[[:space:]]	All space characters tab, newline, vertical tab, form feed, carriage return, and space
[:upper:]	All upper case alphabets [A-Z]
[:xdigit]	Only match Hexadecimal digits [0-9A-Fa-f]

Anchoring

^	Defines start of a line E.g **grep –E '^Th' file.txt** Finds lines starting with Th: there was a, they were, they are
$	Defines end of a **line** (Requires text to be in EOL format) E.g. **grep –E 'ble.$' file.txt** Finds lines ending with "ble." i.e. He was capable. I am able. He lost his marble.

The Backslash Character and Special Expressions

\<	Defines **start** of a **word** E.g. **grep –E '\<b[aeiou]?' file.txt** Finds words starting with **b** followed by a vowel like boots, books, but not robot
\>	Defines end of a **word** E.g **grep –E '[aeiou]t\>' file.txt** Finds words ending with **t** preceeded by a vowel like boot, but **not** boots, robot but not robotic
\b	Defines **start or end** of a **word** E.g **grep –E '\b**b[aeiou]k**b' file.txt** Finds words starting with b and ending with k, result can be bark, book, bizmark, etc
\B	\B prefix means preceeded by graphical character (gc) not space, as suffix means followed by gc not space. **grep –E '\Bk\B' file.txt** Finds words with k in the middle of the word: broken, maker, taker **grep –E '\B**k' **file.txt** Finds words with k in the middle or the end of a word: book, broken, maker, taker, but NOT kite, Karen **grep –E** 'k**\B' file.txt** Finds words beginning or the middle of words e.g. broken, kite, Karen, maker but NOT book, cook
\w	matches all alphanumeric characters [[:alnum:]]
\W	matches all EXCEPT alphanumeric characters [^[:alnum:]]

Repetition

A regular expression may be followed by one of several repetition operators:

.	Matches any single character. Does not have to have regular expression defined
?	The preceding item is optional and matched 0 or 1 time. e.g. **grep 'Par[[:alpha:]]?'** Will highlight: **part**, **par**, **para** and only **para** in **para**graph
*****	The preceding item will be matched 0-∞ until word ends. e.g. **grep 'Par[[:alpha:]]*'** Will find: part, par, partner, paragraph, pariah, but not "part of" as the word ended after "part"
+	The preceding will be match at least once. Same as * but it expects to find at least 1 match or gives error
{n}	The preceding matched exactly n times. e.g. **grep 'Par[[:alpha:]]{2}'** Will find: par**ty**, par**ks**, par**ma**, but NOT park, par, pariah, because you are looking for exact 2 alphabets in this case

	after par.
{n,}	The preceding is matched at least n times e.g. **grep 'Par[[:alpha:]]{2,}'** Will highlight: **party**, **parks**, **parma**, **pariah**, **paragraph** but not part, park or par because you want two or more alphabets in this case.
{,m}	The preceding is matched at most m times e.g. **grep 'Par[[:alpha:]]{,2}'** Will match: par**ty**, par**ks**, par**ma**, par, part but NOT pariah, paragraph because you want 2 or less alphabets after par.
{n,m}	The preceding item is matched at least n times, but not more than m times. e.g. **grep 'Par[[:alpha:]]{2,4}'** Will match: par**ty**,par**ties** par**ks**, par**ma**, par**iah** but NOT part,par, paragraph because no. alphabets following Par has to be between 2 and 4

Concatenation

RE1.*RE2.*RE3.*RE4 **,** **Grep -E RE1 file.txt| grep -E RE2**	Show matches containing RE1 AND RE2 AND RE3 AND RE4 in the same line in that order and anything in between. This is done by joining regular exp by **.*.** Using separate grep commands joined by | will only highlight last grep RE. E.g. **grep –E 'Manager.*Sales' exployee.txt** This will find all lines where staff is a sales related manager

Alternation

RE1|RE2|RE3|RE4	Show matches from either expression of RE1 or RE2 or RE3 or RE4. This is done by joining regular exp by infix **operator |** E.g. **grep –E 'boo*|mac*|pi*' file.txt** This will find books,boots,boot, Boolean, machine, macbook,macaroni,pit, pi,pint etc

Precedence

Repetition > concatenation > alternation
Can be overridden by enclosing whole expression in parenthesis

Back References and Subexpressions

\n	The match you got from the nth () enclosed regular expression. The back-reference \n, where n is a single digit, matches the regular subexpression previously matched by the nth parenthesized subexpression. e.g. **grep –E '(camel)\1{2}'**

	Here you will match for the word camel first, then you will look for two other "camel" attached to first camel. Essentially you are looking for camelcamelcamel.

Matcher Selection

-E, **--extended-regexp**	Interpret PATTERN as an extended regular expression
-F, **--fixed-strings**	Searches for fixed strings. All characters are treated literally (nothing is special). You cannot have fancy regular expressions in –F
-G, **--basic-regexp,** **grep**	Default pattern matching behaviour (redundant). Interpret a basic regular expression where meta characters are already assumed to be literal \{.
-P, **--perl-regexp**	Interpret PATTERN as a Perl regular expression. This is highly experimental and grep -P may warn of unimplemented features.

Matching Control

-e PATTERN, **--regexp=PATTERN**	Use special characters as string, Search multiple patterns. e.g. **grep -e '–--foo' -e 'mary' -e '{dog}'** will search for the word ---foo AND mary AND {dog}
-f FILE, **--file=FILE**	Match pattern from file. Use one pattern per line. For regular expression, use **grep -E -f**
-i, **--ignore-case**	Ignore upper/lower case distinctions
-v, **--invert-match**	Look for everything BUT the input pattern
-w, **--word-regexp**	Match whole words only.
-x, **--line-regexp**	Mat .ch entire line only

General Output Control

-c, --count	Print count of lines that have matches. Hide normal output
--color[=WHEN], --colour[=WHEN]	Color the matched strings “always” or “auto” or “never”
-L, --files-without-match	Print name input files that have no matches. Hide normal output
-l, --files-with-matches	Print name input files that have one or more matches. Hide normal output
-m N, --max-count=N	Only find first N matches
-o, --only-matching	Print the matched string only. Show one matched string per line.
-q, --quiet, --silent	Do not show any output
-s, --no-messages	Hide error messages and missing/unreadable files

Output Line Prefix Control

-b, **--byte-offset**	Show character/byte count that the match is offset by as Prefix. If there are 2 lines of 40 characters that exists before the match, it will show “40 <match string>”.
-H, **--with-filename**	Show file name for each match as prefix. On by default when there is more than one file to search. e.g. “file.txt <match sting>
-h, **--no-filename**	Prevent prefixing of file names on output.
--label=LABEL	Display input actually coming from standard input as input coming from file LABEL. This is especially useful when implementing tools like zgrep, e.g., gzip -cd foo.gz | grep --label=foo -H something. See also the -H option.
-n, **--line-number**	Show line number of each match as prefix
-T,	

--initial-tab	Introduce tabs so prefixes and match line up nicely
-u, **--unix-byte-offsets**	Report Unix-style byte offsets. This switch causes grep to report byte offsets as if the file were a Unix-style text file, i.e., with CR characters stripped off. This will produce results identical to running grep on a Unix machine. This option has no effect unless -b option is also used; it has no effect on platforms other than MS-DOS and MS-Windows.
-Z, **--null**	Output a zero byte (the ASCII NUL character) instead of the character that normally follows a file name. For example, grep -lZ outputs a zero byte after each file name instead of the usual newline.

Context Line Control

-A N, **--after-context=N**	Print N number of lines that exist AFTER matching lines. Separate match+N with –-(group separator)
-B N, **--before-context=N**	Print N number of lines that exist BEFORE matching lines. Separate match+N with –- (group separator)
-C N, **-N,** **--context=NUM**	Print N number of lines that exist BEFORE and AFTER matching lines. Separate match+N with –- (group separator)
--group-separator=SEP	Use SEP as a group separator. By default SEP is double hyphen (--).
--no-group-separator	Use empty string as a group separator.

File and Directory Selection

-a, **--text** **--binary-files=text**	Process a binary file as if it were text; this is equivalent to the
--binary-files=TYPE	If the first few bytes of a file indicate that the file contains binary data, assume that the file is of type TYPE. Default=binary
-D ACTION, **--devices=ACTION**	If an input file is a device, FIFO or socket, use ACTION to process it. By default, ACTION is read, which means that devices are read just as if they were ordinary files. If ACTION is skip, devices are silently skipped.

-d ACTION, --directories=ACTION	If an input file is a directory, use ACTION to process it. By default, ACTION is read, i.e., read directories just as if they were ordinary files.If ACTION is skip, silently skip directories. If ACTION is recurse, read all files under each directory, recursively, following symbolic links only if they are on the command line. This is equivalent to the -r option.
--exclude=GLOB	Skip files whose base name matches GLOB (using wildcard matching). A file-name glob can use *, ?, and [...] as wildcards, and \ to quote a wildcard or backslash character literally.
--exclude-from=FILE	Skip files whose base name matches any of the file-name globs read from FILE (using wildcard matching as described under --exclude).
--exclude-dir=DIR	Exclude directories matching the pattern DIR from recursive searches.
-I	Process a binary file as if it did not contain matching data; this is equivalent to the --binary-files=without-match option.
--include=GLOB	Search only files whose base name matches GLOB (using wildcard matching as described under --exclude).
-r, --recursive	Read all files under each directory, recursively, following symbolic links only if they are on the command line. This is equivalent to the -d recurse option.
-R, **--dereference-recursive**	Read all files under each directory, recursively. Follow all symbolic links, unlike -r.

Other Options

--line-buffered	Use line buffering on output. This can cause a performance penalty.
-U, --binary	Treat the file(s) as binary. By default, under MS-DOS and MS-Windows, grep guesses the file type by looking at the contents of the first 32KB read from the file. If grep decides the file is a text file, it strips the CR characters from the original file contents (to make regular expressions with ^ and $ work correctly). Specifying -U overrules this guesswork, causing all files to be read and passed to

	the matching mechanism verbatim; if the file is a text file with CR/LF pairs at the end of each line, this will cause some regular expressions to fail.This option has no effect on platforms other than MS-DOS and MS-Windows.
-z, --null-data	Treat the input as a set of lines, each terminated by a zero byte (the ASCII NUL character) instead of a newline. Like the -Z or --null option, this option can be used with commands like sort -z to process arbitrary file names.