System Commands & RegEx

Table of Contents

Introduction

When using the system and especially the abstractions of the shell, we do not want to hard code all of the small programs that we can use into the shell. That would make maintaining and adding programs difficult. Thus, in the usr/bin in path, we have a ton of small tools of the system that accomplish tasks really well. The components are then connected to make them fast, maintainable and well structured.

Grep

If `grep` finds a line with the PATTERN, it outputs it. So it is similar to cat, but more selective. In fact, if you use a pattern that matches every character, grep can behave identically to cat: `grep '.*' filename`

```
 NAME
       grep, egrep, fgrep, rgrep - print lines that match patterns

SYNOPSIS
       grep [OPTION...] PATTERNS [FILE...]
       grep [OPTION...] -e PATTERNS ... [FILE...]
       grep [OPTION...] -f PATTERN_FILE ... [FILE...]

DESCRIPTION
       grep  searches  for  PATTERNS  in  each  FILE.  PATTERNS is one or more
       patterns separated by newline characters, and  grep  prints  each  line
       that  matches a pattern.  Typically PATTERNS should be quoted when grep
       is used in a shell command.

       A FILE of “-”  stands  for  standard  input.   If  no  FILE  is  given,
       recursive  searches  examine  the  working  directory, and nonrecursive
       searches read standard input.

       In addition, the variant programs egrep, fgrep and rgrep are  the  same
       as  grep -E,  grep -F,  and  grep -r, respectively.  These variants are
       deprecated, but are provided for backward compatibility.

OPTIONS
   Pattern Syntax
       -E, --extended-regexp
              Interpret PATTERNS as extended regular  expressions  (EREs,  see
              below).

   Matching Control
       -e PATTERNS, --regexp=PATTERNS
              Use  PATTERNS  as the patterns.  If this option is used multiple
              times or is combined with the -f (--file) option, search for all
              patterns  given.   This  option can be used to protect a pattern
              beginning with “-”.

       -f FILE, --file=FILE
              Obtain patterns from FILE, one per line.  If this option is used
              multiple  times  or  is  combined with the -e (--regexp) option,
              search for all patterns given.  The  empty  file  contains  zero
              patterns, and therefore matches nothing.
```

Regular Expressions

Basic Regular Expressions

---

grep actually uses Basic regular expressions (BREs), a simpler form of the more familiar regex, extended regular expressions

(EREs). It also only matches against single lines at a time.

- empty regular expression will match the empty string
- `[]` matches a single character but only those that are included in the set enclosed in the square brackets
    - within a set, if you want a `-` sign, put it at the end
    - `[@-~]` matches the ascii characters
- `^` matches the start of a the line
- `$` matches the end of the line
- `.` matches every single character
- `ABC*` any or more characters of C
    - binds as tight as possible so only to C
    - `\(abc\)*` searches for abcabcabc (how to escape tight binding)
- `\` must be escaped

Quoting

- If we use the following command `'['"\]'` the system will prompt input because it is looking to finish the command as an open double quote signifies that you are reading a string
- the idea of quoting suggests that you feed the output of programs as quoted inputs to other programs
    - Quoting and little language commands like grep go hand-in-hand.
    - Scripts are evaluated by the shell first before being shipped off as arguments to its child programs
    - for instance the argument of grep, often quoted regex is a small program that you give to GREP.
    - make sure you quote and/or escape everything keeping in mind how your initial string will be resolved as it makes its way through each program
- As described in single quoting, single quotes are often used for grep expressions because they preserve everything literally - what you see within the quotes is exactly what you're giving grep.
- double quotes to be able to interpolate regex fragments into the larger expression.

Special Characters in Quoting for the Shell

- `\\` = \
- `\'` = '
- `\"` = "
- `'what ever @ you wont'` take all of the words inside the apostrophe and treat it as a single word
    - you cannot put apostrophes within this type of quoting
- `"what you like"` = what you like will be treated as a single word
    - `` $ \ ` `` are the special characters in double quotes

**Consider we are searching for: ** `['"\]`

1. `\` encase all things in a backslash. `\[\'\"\\\]`
2. `'` encase all things in single quote, escape single quote `'['\''"\\]'` or `'['\''"\]'`
3. `"` encase all things in double quote, escape double quote `"['"\""\\]"` or `"['\"\\]"`
    - You must `\\` when you put things in shell commands because, shell evaluates stuff, then the mini languages

Extended Regular Expressions

Historically there was another team that came up with alternate syntax to the familiar grep, called egrep. Nowadays you can use the -E flag to specify that the pattern is using the extended syntax instead

> In extended regular expression, you do not need to escape out of `()` like you would `\(word\)` in regular

- `+` quantifier to watch 1 or more occurrences, similar to PP*
- `P{2-5}` would match `P` 2-5 times in a row inclusive. We must quote otherwise, it will glob in the shell
- `(eggert|foo)` matches either the entire string Eggert or the entire string foo

- `[^a-z]` negates the values from the response, removes lines with just a-z responses
- `P?` 0 or one instance of the object
- `|` is the or command
- `()` groups some notations

Examples

```
 []a-z]      # matches either the closing bracket or characters from a-z (must be in quotes)
[^]a-z]     # negation of ] or a-z
[^a-z]      # matches all non a-z lines as the ^ acts as a negation of the set
[a-z^]      # matches either a-z or ^
[a-z*]      # matches a-z or star
grep \q\a\b\c\*\d or 'qabc*d'       # qabcd or qabccd or qabcccd
grep 'q\(abc\)*d'   # qabcd or qabcabcd
grep -E 'q(abc)*d' # gives us qabcd or qabcd (don't need \ in extended)
grep 'A' file | grep 'B' # finds all occurrences of A, then filters out all without B from results of
P{3,5}      # 3-5 instances of p
P{3;}       # 3 to infinity instances of P
^(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9][0-9]|[0-9])$ # find numbers between 0-255
\[\'\"\\\]      # ['"\] every character in the regular expression must have a \ in front of it without
\['\''"'\']     #  [’”\]
['\''"'\']      # would fail, as it does not evaluate the set rather the expression literal [\"']
'[a-zA-Z0-9]+'          # at least 1 alphanumeric character
'\\"([^"\]|\\.)*\\"'    # \"(something)\", where (something) is . OR neither " or \
```