Week 2: Regular Expressions

grep,linux,regex

- notation that lets you search for text with a particular pattern
  - *"starts with the letter a, ends with three uppercase letters, etc."*
- test regex expressions
  - http://regexpal.com

Character	BRE / ERE	Meaning in a pattern
\	Both	Usually, turn off the special meaning of the following character. Occasionally, enable a special meaning for the following character, such as for \(...\) and \{...\}.
.	Both	Match any single character except NUL. Individual programs may also disallow matching newline.
*	Both	Match any number (or none) of the single character that immediately precedes it. For EREs, the preceding character can instead be a regular expression. For example, since . (dot) means any character, .* means "match any number of any character." For BREs, * is not special if it's the first character of a regular expression.
^	Both	Match the following regular expression at the beginning of the line or string. BRE: special only at the beginning of a regular expression. ERE: special everywhere.

$	Both	Match the preceding regular expression at the end of the line or string. BRE: special only at the end of a regular expression. ERE: special everywhere.
[...]	Both	Termed a bracket expression, this matches any one of the enclosed characters. A hyphen (-) indicates a range of consecutive characters. (Caution: ranges are locale-sensitive, and thus not portable.) A circumflex (^) as the first character in the brackets reverses the sense: it matches any one character not in the list. A hyphen or close bracket (]) as the first character is treated as a member of the list. All other metacharacters are treated as members of the list (i.e., literally). Bracket expressions may contain collating symbols, equivalence classes, and character classes (described shortly).
\{*n*,*m*\}	BRE	Termed an *interval expression*, this matches a range of occurrences of the single character that immediately precedes it. \{*n*\} matches exactly n occurrences, \{*n*,\} matches at least n occurrences, and \{*n*,*m*\} matches any number of occurrences between n and m. n and m must be between 0 and RE_DUP_MAX (minimum value: 255), inclusive.
\(\)	BRE	Save the pattern enclosed between \(and \) in a special *holding space*. Up to nine subpatterns can be saved on a single pattern. The text matched by the subpatterns can be reused later in the same pattern, by the escape sequences \1 to \9. For example, \(ab\).*\1 matches two occurrences of ab, with any number of characters in between.
\n	BRE	Replay the nth subpattern enclosed in \(and \) into the pattern at this point. n is a number from 1 to 9, with 1 starting on the left.
{*n*,*m*}	ERE	Just like the BRE \{*n*,*m*\} earlier, but without the backslashes in front of the braces.
+	ERE	Match one or more instances of the preceding regular expression.
?	ERE	Match zero or one instances of the preceding regular expression.
|	ERE	Match the regular expression specified before or after.
( )	ERE	Apply a match to the enclosed group of regular expressions.

Matching Multiple Charcters with One Expression

*	Match zero or more of the preceding character
\{*n*\}	Exactly n occurrences of the preceding regular expression
\{*n*,\}	At least n occurrences of the preceding regular expression
\{*n*,*m*\}	Between n and m occurrences of the preceding regular expression

Examples

- **tolstoy**
  - the sevel letters, **tolstoy**, anywhere on a line
- **^tolstoy**
  - the seven letters, **tolstoy**, at the beginning of the line
- **tolstoy$**
  - the seven letters, **tolstoy**, at the end of the line
- **^tolstoy$**
  - a line containing exactly the seven letters, **tolstoy**, and nothing else
- **[Tt]olstoy**
  - either the seven letters, **Tolstoy**, or **tolstoy**, anywhere on a line
- **tol.toy**
  - the three letters, **tol**, any character, and the three letters **toy** anywhere on a line
- **tol.*toy**
  - the three letters **tol**, any sequence of zero or more characters, and the three letters **toy**, anywhere on a line
- **o\{5\}**
  - don't need the backslashes if using `grep -E`
  - five consecutive **o**'s anywhere on a line

Searching for Text

`grep`: use basic regular expressions (BRE)

- meta-characters (?, +, {, |, (, and)) lose their special meaning
    - used the backslashed versions
- **o{5}** would search for that literally
    - use **o\{5\}** instead

egrep (`grep -E`)

- uses extended regular expressions (ERE)
- no backslashes needed
- **o{5}** would search for 5 o's

fgrep (`grep -F`)

- matches fixed strings instead of regular expressions

POSIX Bracket Expressions

Class	Matching characters	Class	Matching characters
[:alnum:]	Alphanumeric characters	[:lower:]	**Lowercase characters**
[:alpha:]	Alphabetic characters	[:print:]	Printable characters
[:blank:]	**Space and tab characters**	[:punct:]	Punctuation characters
[:cntrl:]	Control characters	[:space:]	Whitespace characters
[:digit:]	Numeric characters	[:upper:]	**Uppercase characters**
[:graph:]	Nonspace characters	[:xdigit:]	Hexadecimal digits

Simple `grep` example

```
$ who                       # Who is logged on?
tolstoy tty1 Feb 26 10:53
tolstoy pts/0 Feb 29 10:59
tolstoy pts/1 Feb 29 10:59
tolstoy pts/2 Feb 29 11:00
tolstoy pts/3 Feb 29 11:00
```

```
tolstoy pts/4 Feb 29 11:00
austen pts/5 Feb 29 15:39 (mansfield-park.example.com)
austen pts/6 Feb 29 15:39 (mansfield-park.example.com)

$ who | grep -F austen      # Where is austen logged on?
austen pts/5 Feb 29 15:39 (mansfield-park.example.com)
austen pts/6 Feb 29 15:39 (mansfield-park.example.com)
```

`sed` (stream editor)

- used to replace parts of text
- `sed 's/Hello/World/g'`
  - replaces all instances of “Hello” with “World”
  - the [g] means globally and causes **every** instance of the regular expression to be replaced by the replacement text
- `echo $PATH | sed 's/:.*//'`
  - remove everything after and including the first colon from the $PATH variable