Reading 03

☆ Book: The Linux Command Line ☆

☆ Chapter 19: Regular Expressions

- Simply put, regular expressions are symbolic notations used to identify patterns in text.
- The main program we will use to work with regex is "grep" (global regular expression print)
  - ↳ The grep program accepts options and arguments this way, where regex is a regular expression:

    grep [options] regex [file...]

    ↳ Table 19-1 describes the commonly used `grep` options.

Table 19-1: `grep` Options

Option	Long Option	Description
`-i`	`--ignore-case`	Ignore case. Do not distinguish between uppercase and lowercase characters.
`-v`	`--invert-match`	Invert match. Normally, `grep` prints lines that contain a match. This option causes `grep` to print every line that does not contain a match.
`-c`	`--count`	Print the number of matches (or non-matches if the `-v` option is also specified) instead of the lines themselves.
`-l`	`--files-with-matches`	Print the name of each file that contains a match instead of the lines themselves.
`-L`	`--files-without-match`	Like the `-l` option, but print only the names of files that do not contain matches.
`-n`	`--line-number`	Prefix each matching line with the number of the line within the file.
`-h`	`--no-filename`	For multi-file searches, suppress

↳ Ex: `[me@linuxbox ~]$ grep bzip dirlist*.txt`

↳ Searches for substring "bzip" in the contents of any file whose name takes this form.

↳ Ex: `[me@linuxbox ~]$ grep -l bzip dirlist*.txt`

↳ returns just the names of the files that contain substring.

↳ Ex: `[me@linuxbox ~]$ grep -L bzip dirlist*.txt`

↳ returns just the names of the files that DO NOT contain the substring.

Metacharacters and Literals

- Literals are plain vanilla matches like 'b', 'z', 'i', 'p' in the example above
- Metacharacters are used for more complex matches. They consist of the following:

```
^ $ . [ ] { } - ? * + ( ) | \
```

- When you pass regex functions containing metacharacters, ALWAYS use quotes

The Any Character

- dot character → used to match any character.

↳ Ex:

```
[me@linuxbox ~]$ grep -h '.zip' dirlist*.txt
bunzip2
bzip2
bzip2recover
gunzip
gzip
funzip
gpg-zip
```

matches any line in a file that takes this form which contains substring ".zip" as long as it's preceded by at least one character

- Anchors

↳ these are the caret (^) and dollar sign ($) characters

↳ they cause the match to occur only if the regular expression is found at the beginning of the line (^) or at the end of the line ($)

↳ Ex:

```
[me@linuxbox ~]$ grep -h '^zip' dirlist*.txt
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
[me@linuxbox ~]$ grep -h 'zip$' dirlist*.txt
gunzip
gzip
funzip
gpg-zip
preunzip
prezip
unzip
zip
[me@linuxbox ~]$ grep -h '^zip$' dirlist*.txt
zip
```

↳ Note: the regular expression "^$" will match blank lines.

↳ Ex: `[me@linuxbox ~]$ grep -i '^..j.r$' /usr/share/dict/words`

looking for 5-letter words whose 3rd letter is a 'j' and 5th letter is 'r'

Bracket Expressions and Character Classes

- Using bracket expressions we can also match a single character from a specified set of characters
  - Ex Searching for any line that contains "bzip" or "gzip":

```
[me@linuxbox ~]$ grep -h '[bg]zip' dirlist*.txt
bzip2
bzip2recover
gzip
```

- Most metacharacters loose their meaning in brackets.
  - However the caret (^) and dash (-) hold a meaning.
    - If the 1st character in a bracket expression is a caret (^), the remaining characters are taken to be the set of characters that must not be present at the given character expression.
  - Ex:

```
[me@linuxbox ~]$ grep -h '[^bg]zip' dirlist*.txt
bunzip2
gunzip
funzip
gpg-zip
preunzip
prezip
prezip-bin
unzip
unzipsfx
```

- Note: the caret (^) only has this meaning if it's the 1st character in the set.
- The dash character allows you to enter ranges

Ex matching all lines (within files) ^ that start with letters or numbers:

```
[me@linuxbox ~]$ grep -h '^[A-Za-z0-9]' dirlist*.txt
```

- Note: if you put the dash to be the first character, then the special meaning is taken away.

POSIX Character Classes

- to check language setting of the program use `echo $LANG`
- POSIX character classes:

Table 19-2: POSIX Character Classes

Character Class	Description
[:alnum:]	The alphanumeric characters. In ASCII, equivalent to: [A-Za-z0-9]
[:word:]	The same as [:alnum:], with the addition of the underscore (_) character.
[:alpha:]	The alphabetic characters. In ASCII, equivalent to: [A-Za-z]
[:blank:]	Includes the space and tab characters.
[:cntrl:]	The ASCII control codes. Includes the ASCII characters 0 through 31 and 127.
[:digit:]	The numerals 0 through 9.
[:graph:]	The visible characters. In ASCII, it includes characters 33 through 126.
[:lower:]	The lowercase letters.
[:punct:]	The punctuation characters. In ASCII, equivalent to: [-!"#$%&'()*+,./:;<=>?@[\\\]_`{|}~]
[:print:]	The printable characters. All the characters in [:graph:] plus the space character.
[:space:]	The whitespace characters including space, tab, carriage return, newline, vertical tab, and form feed. In ASCII, equivalent to: [\t\r\n\v\f]
[:upper:]	The uppercase characters.
[:xdigit:]	Characters used to express hexadecimal numbers. In ASCII, equivalent to: [0-9A-Fa-f]

- POSIX character classes can be used for the regex and the shell performing pathname expansion.

POSIX Basic vs. Extended Regular Expressions:

- In Basic Regular Expressions (BRE) the following metacharacters are recognized: ^ $. [ ] *
- In Extended Regular Expressions (ERE) the following metacharacters are added: ( ) { } ? + |
- However, the "(", ")", "{", "}" characters are treated as metacharacters in BRE if they are escaped with a backslash, whereas in ERE, preceding any metacharacter with a backslash causes it to be turned literal.

- You can use ERE metacharacters with "grep" by adding the "-E" flag.

Alternation

- You can use a pipe to basically perform an "OR" match.

↳ Ex:

```
[me@linuxbox ~]$ echo "AAA" | grep -E 'AAA|BBB'
AAA
[me@linuxbox ~]$ echo "BBB" | grep -E 'AAA|BBB'
BBB
[me@linuxbox ~]$ echo "CCC" | grep -E 'AAA|BBB'
[me@linuxbox ~]$
```

- Note you can keep adding pipes and substrings

↳ Ex:

```
[me@linuxbox ~]$ grep -Eh '^(bz|gz|zip)' dirlist*.txt
```

This expression will match the filenames in our lists that start with either `bz`, `gz`, or `zip`. Had we left off the parentheses, the meaning of this regular expression changes to match any filename that begins with `bz` *or contains* `gz` *or contains* `zip`:

```
[me@linuxbox ~]$ grep -Eh '^bz|gz|zip' dirlist*.txt
```

Quantifiers

"?" - Match an element 0 or 1 times. (basically make it optional)

- Ex if we wanted to check phone number validity and only considered these two forms to be valid:

(nnn) nnn-nnnn
nnn nnn-nnnn
} where 'n' is a number

we could use this regex expression: `^\(?[0-9][0-9][0-9]\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$`

- '\' used to escape special meaning of '(' and ')' in ERE
- '?' used to make parentheses optional

- Ex

```
[me@linuxbox ~]$ echo "(555) 123-4567" | grep -E '^\(?[0-9][0-9][0-9]
\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$'
(555) 123-4567
[me@linuxbox ~]$ echo "555 123-4567" | grep -E '^\(?[0-9][0-9][0-9]\)
? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$'
555 123-4567
[me@linuxbox ~]$ echo "AAA 123-4567" | grep -E '^\(?[0-9][0-9][0-9]\)
? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$'
[me@linuxbox ~]$
```

- "*" Matches an element zero or more times

- Ex

```
[me@linuxbox ~]$ echo "This works." | grep -E '^[[:upper:]][[:upper:]
[:lower:] ]*\.'
This works.
[me@linuxbox ~]$ echo "This Works." | grep -E '^[[:upper:]][[:upper:]
[:lower:] ]*\.'
This Works.
[me@linuxbox ~]$ echo "this does not" | grep -E '^[[:upper:]][[:upp
er:][:lower:] ]*\.'
[me@linuxbox ~]$
```

The expression matches the first two tests, but not the third, since it lacks the required leading uppercase character and trailing period.

- the [[:upper:][:lower:]]* means there can be a sequence of any length as long as it contains either upper or lower case letters.

- The characters are also optional again.

- "+" matches an element one or more times

↳ same as "*" but requires at least one instance of the preceeding element to cause a match.

↳ Ex :

```
[me@linuxbox ~]$ echo "This that" | grep -E '^([[:alpha:]]+ ?)+$'
This that
[me@linuxbox ~]$ echo "a b c" | grep -E '^([[:alpha:]]+ ?)+$'
a b c
[me@linuxbox ~]$ echo "a b 9" | grep -E '^([[:alpha:]]+ ?)+$'
[me@linuxbox ~]$ echo "abc  d" | grep -E '^([[:alpha:]]+ ?)+$'
[me@linuxbox ~]$
```

- "{}" matches an element a specified # of times

↳ Table 19-3 : Specifying the Number of Matches :

Specifier	Meaning
{n}	Match the preceding element if it occurs exactly *n* times.
{n,m}	Match the preceding element if it occurs at least *n* times but no more than *m* times.
{n,}	Match the preceding element if it occurs *n* or more times.
{,m}	Match the preceding element if it occurs no more than *m* times.

- So this! `^\(?[0-9][0-9][0-9]\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$`

↳ can be simplified to : `^\(?[0-9]{3}\)? [0-9]{3}-[0-9]{4}$`

- Putting Regex to Work:

↳ Ex:

```
[me@linuxbox ~]$ grep -Ev '^\([0-9]{3}\) [0-9]{3}-[0-9]{4}$'
phonelist.txt
(292) 108-518
(129) 44-1379
[me@linuxbox ~]$
```

↳ "-v" flag initiates inverse match (So only outputs lines in the list that do not match the specified expression)

- 'find' command with regex

↳ find requires an EXACT match with the regular expression.

↳ also need to add "-regex" flag.

- 'locate' command with regex

↳ supports basic regex with "--regexp" flag

↳ supports extended regex with "--regex" flag

- regex with 'less' command and vim

- for both, simply press the '/' key followed by the regex expression to perform a search.

☆ Chapter 20: Text Processing

- 'cat -A my_file' will show non printing characters in the text.

↳ Ex:

```
[me@linuxbox ~]$ cat -A foo.txt
^IThe quick brown fox jumps over the lazy dog.   $
[me@linuxbox ~]$
```

indicates tab

shows true end of line, indicating we have trailing whitespace

↳ "-n" flag numbers lines

↳ "-s" flag supresses multiple blank lines

↳ Ex:

```
[me@linuxbox ~]$ cat > foo.txt
The quick brown fox


jumps over the lazy dog.
[me@linuxbox ~]$ cat -ns foo.txt
     1  The quick brown fox
     2
     3  jumps over the lazy dog.
[me@linuxbox ~]$
```

- 'sort' takes in one or more files and returns lines in sorted order

↳ Ex: making a single sorted file out of many input files:

```
sort file1.txt file2.txt file3.txt > final_sorted_list.txt
```

↳ Common Sort Flags:

Table 20-1: Common `sort` Options

Option	Long Option	Description
`-b`	`--ignore-leading-blanks`	By default, sorting is performed on the entire line, starting with the first character in the line. This option causes `sort` to ignore leading spaces in lines and calculates sorting based on the first non-whitespace character on the line.
`-f`	`--ignore-case`	Make sorting case-insensitive.
`-n`	`--numeric-sort`	Perform sorting based on the numeric evaluation of a string. Using this option allows sorting to be performed on numeric values rather than alphabetic values.
`-r`	`--reverse`	Sort in reverse order. Results are in descending rather than ascending order.
`-k`	`--key=`*`field1`*`[,`*`field2`*`]`	Sort based on a key field located from *field1* to *field2* rather than the entire line. See the following discussion.
`-m`	`--merge`	Treat each argument as the name of a presorted file. Merge multiple files into a single sorted result without performing any additional sorting.
`-o`	`--output=`*`file`*	Send sorted output to *file* rather than standard output.
`-t`	`--field-separator=`*`char`*	Define the field-separator character. By default fields are separated by spaces or tabs.

↳ Use Case:

limits to 10 rows

```
[me@linuxbox ~]$ du -s /usr/share/* | sort -nr | head
509940  /usr/share/locale-langpack
242660  /usr/share/doc
197560  /usr/share/fonts
179144  /usr/share/gnome
146764  /usr/share/myspell
144304  /usr/share/gimp
135880  /usr/share/dict
76508   /usr/share/icons
68072   /usr/share/apps
62844   /usr/share/foomatic
```

↳ use case!

```
[me@linuxbox ~]$ ls -l /usr/bin | sort -nrk 5 | head
```

↳ sorts output of 'ls -l' numerically and reversed based on the 5th field or column.

- you can sort on multiple keys / fields.
  (fields are defined with whitespace delimeters by default)
  - you can specify delimiter via '-t' flag
    - Ex: `sort -t ':' my_file | head`

- Ex: Sorting by multiple keys

"start at field 1, end at field 1" "field 2 is the sort key and numerically sort"

```
[me@linuxbox ~]$ sort --key=1,1 --key=2n distros.txt
Fedora  5       03/20/2006
Fedora  6       10/24/2006
Fedora  7       05/31/2007
Fedora  8       11/08/2007
Fedora  9       05/13/2008
Fedora  10      11/25/2008
SUSE    10.1    05/11/2006
SUSE    10.2    12/07/2006
SUSE    10.3    10/04/2007
SUSE    11.0    06/19/2008
Ubuntu  6.06    06/01/2006
Ubuntu  6.10    10/26/2006
Ubuntu  7.04    04/19/2007
Ubuntu  7.10    10/18/2007
Ubuntu  8.04    04/24/2008
Ubuntu  8.10    10/30/2008
```

- Ex: `[me@linuxbox ~]$ sort -k 3.7nbr -k 3.1nbr -k 3.4nbr distros.txt`
  - This means sort by the 7th character within the 3rd field
  - then sort by the 1st character in the 3rd field
  - then sort by the 4th character in the 3rd field

- 'uniq' command is often used with 'sort'
  - takes in a sorted file and remove duplicates.
  - 'sort' also now has '-u' flag for this purpose

- Common 'uniq' options:

Table 20-2: Common `uniq` Options

Option	Long Option	Description
`-c`	`--count`	Output a list of duplicate lines preceded by the number of times the line occurs.
`-d`	`--repeated`	Output only repeated lines, rather than unique lines.
`-f n`	`--skip-fields=n`	Ignore *n* leading fields in each line. Fields are separated by whitespace as they are in `sort`; however, unlike `sort`, `uniq` has no option for setting an alternate field separator.
`-i`	`--ignore-case`	Ignore case during the line comparisons.
`-s n`	`--skip-chars=n`	Skip (ignore) the leading *n* characters of each line.
`-u`	`--unique`	Output only unique lines. Lines with duplicates are ignored.

- Ex: using 'uniq' to report the # of duplicates in our text file:

```
[me@linuxbox ~]$ sort foo.txt | uniq -c
      2 a
      2 b
      2 c
```

- 'cut' command extracts a section of text from a line and output the extracted section to stdout.

  ↳ cut options:

Table 20-3: `cut` Selection Options

Option	Long Option	Description
`-c` *list*	`--characters=`*list*	Extract the portion of the line defined by *list*. The list may consist of one or more comma-separated numerical ranges.
`-f` *list*	`--fields=`*list*	Extract one or more fields from the line as defined by *list*. The list may contain one or more fields or field ranges separated by commas.
`-d` *delim*	`--delimeter=`*delim*	When `-f` is specified, use *delim* as the field delimiting character. By default, fields must be separated by a single tab character.
	`--complement`	Extract the entire line of text, except for those portions specified by `-c` and/or `-f`.

↳ It can cut:

- fields using 'tab' as the natural delimeter

  ↳ Ex:

```
[me@linuxbox ~]$ cut -f 3 distros.txt
12/07/2006
11/25/2008
06/19/2008
04/24/2008
11/08/2007
10/04/2007
10/26/2006
```

  ↳ you can specify delimeter as well? using '-d' flag

```
[me@linuxbox ~]$ cut -d ':' -f 1 /etc/passwd | head
```

- or character ranges

  ↳ Ex:

```
[me@linuxbox ~]$ cut -f 3 distros.txt | cut -c 7-10
2006
2008
2008
2008
```

- 'paste' command does the opposite of cut.

  ↳ it adds one or more columns of text to a file.

- 'join' command

  ↳ files must be sorted on the key field for 'join' to work properly

  ↳ does an inner join.

  ↳ Ex:

```
join -1 1 -2 3 file1 file2 > some_file
```

the key in the first file is the first field

the key in the second file is the 3rd field

redirect output to some_file

- 'tac' command
  - ↳ works like 'cat' but only in reverse
  - ↳ concatonates files in reverse order
  - ↳ useful for re-ordering log files.

- 'rev' command
  - ↳ reverses the characters in a string
  - ↳ often used with 'cut'
  - ↳ Ex:

```
[me@linuxbox ~]$ tail /var/log/backup.log | rev | cut -c 2- | rev
Wed Aug 14 06:00:32 AM EDT 2025: backup (0.6) of linuxbox started
Wed Aug 14 06:08:01 AM EDT 2025: backup of linuxbox finished

Thu Aug 15 07:36:30 AM EDT 2025: backup (0.6) of linuxbox started
Thu Aug 15 07:45:57 AM EDT 2025: backup of linuxbox finished

Fri Aug 16 07:37:57 AM EDT 2025: backup (0.6) of linuxbox started
Fri Aug 16 07:44:11 AM EDT 2025: backup of linuxbox finished
```

- 'comm' command
  - ↳ compares two text files and displays the lines that are unique to each one.
  - ↳ Ex:

```
[me@linuxbox ~]$ comm file1.txt file2.txt
a
        b
        c
        d
    e
```

lines unique to 1st file

lines unique to 2nd file

lines shared by both files

  - ↳ if you used '-1' the 1st column would be suppressed
  - ↳ if you used '-2' the 2nd column would be suppressed
  - ↳ etc.

Reading 3 Continued

- 'diff' command is very powerful to detect changes!
  - '-c' flag outputs context format
    - ↳ shows the group of changes
    - ↳ *Table 20-5: `diff` Context Format Change Indicators*

Indicator	Meaning
`blank`	A line shown for context. It does not indicate a difference between the two files.
-	A line deleted. This line will appear in the first file but not in the second file.
+	A line added. This line will appear in the second file but not in the first file.
!	A line changed. The two versions of the line will be displayed, each in its respective section of the change group.

  - '-u' flag outputs unified format
    - ↳ Similar to context format but more concise
    - ↳ *Table 20-6: `diff` Unified Format Change Indicators*

Character	Meaning
`blank`	This line is shared by both files.
-	This line was removed from the first file.
+	This line was added to the first file.

- 'patch' command
  - ↳ usually paired with 'diff' command
  - ↳ used for updating files by passing in the output of 'diff'
  - ↳ Ex:

```
[me@linuxbox ~]$ diff -Naur file1.txt file2.txt > patchfile.txt
[me@linuxbox ~]$ patch < patchfile.txt
patching file file1.txt
[me@linuxbox ~]$ cat file1.txt
b
c
d
e
```

- 'tr' command transliterates characters
  - ↳ Ex:

```
[me@linuxbox ~]$ echo "lowercase letters" | tr a-z A-Z
LOWERCASE LETTERS
```

  - ↳ Ex:

```
[me@linuxbox ~]$ echo "lowercase letters" | tr [:lower:] A
AAAAAAAAA AAAAAAA
```

  - ↳ Ex: using the '-s' flag to squeeze (delete) repeated characters

```
[me@linuxbox ~]$ echo "aaabbbccc" | tr -s ab
abccc
```

    - ↳ NOTE: repeated characters must be adjoining

- 'sed' command

  ↳ very powerful command

  ↳ one function of it is to find and replace

  ↳ Ex:

```
[me@linuxbox ~]$ echo "aaabbbccc" | sed 's/b/B/'
aaaBbbccc
```

find → s/b/B/ ← replace

  ↳ Ex: To replace all instances add /g:

```
[me@linuxbox ~]$ echo "aaabbbccc" | sed 's/b/B/g'
aaaBBBccc
```

all instances

find → s/b/B/g ← replace

  ↳ Ex: You can also specify addresses for where to make the change:

```
[me@linuxbox ~]$ echo "front" | sed '2s/front/back/'
front
```

2 → line number address; front → find; back → replace

"So only do this in line 2"

  ↳ Different Address Notations:

Table 20-7: sed Address Notation

Address	Description
n	A line number where *n* is a positive integer.
$	The last line.
/regexp/	Lines matching a POSIX basic regular expression. Note that the regular expression is delimited by slash characters. Optionally, the regular expression may be delimited by an alternate character, by specifying the expression with *\cregexpc*, where *c* is the alternate character.
addr1,addr2	A range of lines from *addr1* to *addr2*, inclusive. Addresses may be any of the single address forms listed earlier.
first~step	Match the line represented by the number *first*, then each subsequent line at *step* intervals. For example 1~2 refers to each odd numbered line, and 5~5 refers to the fifth line and every fifth line thereafter.
addr1,+n	Match *addr1* and the following *n* lines.
addr!	Match all lines except *addr*, which may be any of the forms listed earlier.

- Note: Once again this command is very powerful, entire books have been written on it.

• 'aspell' command

↳ interactive spelling checker

↳ not all Linux systems include aspell by default.

↳ if you do `aspell check myFile.txt`

↳ it will bring up an interactive spell checker

↳ it allows you to replace spelling mistakes but automatically creates the file 'myFile.bak' where the original file will be stored.

↳ if you don't want the backup file you can use the option '--dont-backup'

↳ to check HTML files add the '-H' flag so it ignores HTML tags.

↳ Ex:

```
[me@linuxbox ~]$ aspell -H check foo.txt
```

↑ "foo.txt" contains HTML code

↳ by defaut 'aspell' ignores URLs and email addresses in text.