Technische Universität Berlin

Advanced CLI use – filtering and manipulating text streams

Training for Afghan System Administrators

Filter and manipulate text files and streams

A typical routine task for any admin often involves filtering (finding patterns) and manipulating text files or output streams. In most situations, looking through files by hand is not feasible, so we need to learn how to use different tools to find and then manipulate files or strings in files.

Imagine typical situations like:

- Search the mail server log for all mails from a known spam domain and mark them as spam
- Search the auth log for multiple failed logins from the same IP and use iptables to ban it
- Renaming lots of files that match a certain pattern

Useful tools for text filtering and manipulating (part 1)

- cat / tac

  # print file contents to stdout / tac prints in reverse order
- head / tail

  # show the beginning / end of a text file
- less

  # a pager utility to view files
- cmp

  # compare files byte by byte
- diff

  # compare file line by line
- wc

  # count lines, words, characters
- cut

  # cut (print) columns in a file to stdout or file

Useful tools for text filtering and manipulating (part 2)

- grep <PATTERN>
  # find lines that contain <PATTERN>
- sort
  # sort by different options
- uniq
  # find / delete identical lines in a text file
- tee
  # print stdout/stderr on screen AND save to file (esp. useful with redirectors < and >)
- xargs
  # can run a command on a list of elements but “treats” each element separately
- sed
  # stream editor that reads files or stdin and processes them (non-interactively)

Matching text strings with (simple) regular expressions

A regular expression is a sequence of characters that define a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations. The concept arose in the 1950s, when the American mathematician Stephen Kleene formalized the description of a regular language, and came into common use with the Unix text processing utilities ed, an editor, and grep (global regular expression print), a filter.

Unfortunately, different programs / utilities use different a syntax and pattern matches. For example, besides the grep utility, there are egrep / grep -E (extended grep) and fgrep / grep -F (fixed grep). While the former has much more sophisticated pattern matching techniques, the latter tries to match more literal and knows less meta characters for searching.

Meta characters to use in regular expressions

- ^

  # matches the beginning of a line
- $

  # matches the end of a line
- \<

  # matches the beginning of a word
- >\

  # matches the end of a word
- \

  # meta character escape sequence; the following meta character will be interpreted literally
- [abc]

  # matches a, b, c
- [^abc]

  # matches if not a, b, c

Wildcards in regular expressions

In regular expressions, similar to BASH - wildcards can be used to match zero, one or more characters. Compared to BASH heir meaning is different though, so **make sure not to confuse BASH wildcards with regexp wildcards**!

- *
  # matches **zero or more** occurrences of the preceding character
- ?
  # matches **zero or one** occurrences of the preceding character
- +
  # matches **at least one** occurrence of the preceding character
- .
  # matches **any single character**

Some grep examples

- grep 'Word[1-9]*/>' /path/to/file
  # finds “Word”, “Word1”, “Word42”, “Word9999”
- grep 'Word[1-9]?\>' /path/to/file
  # finds “Word”, or “Word1”, “Word2”, but NOT “Word12” or “Word999”
- grep '^Dec 10' /var/log/syslog
  # finds all lines that start with “Dec 10”
- grep '1\{3,5\}' /path/to/file
  # finds occurences of 3, 4 or 5 repeated “1” in the file
- Grep '[0-9][0-9]' /path/to/file
  # finds all lines that contain (at least) 2 digits in a row

Some sed examples

- sed 's/Hello/Hi/' input.txt > output.txt
  # replace the first “Hello” with “Hi” and save to output.txt
- sed -i 's/Hello/Hi/' file.txt
  # replaces the first occurrence of “Hello” with “Hi” in file.txt (interactive mode!)
- sed -i 's/Hello/Hi/g' file.txt
  # replaces all occurrences of “Hello” with “Hi” in file.txt (interactive mode!)
- sed -i '1,50d' file.txt
  # delete every line from line 1 to line 50 in file.txt (interactive mode!)
- sed -f script_file input.txt > output.txt
  # uses script_file to process input.txt and write the processed file to output.txt
  # script_file that contains single sed operations, each on one line; every line ends with “;”

Exercises

Download copy of Moby Dick (UTF8; raw text) from https://www.gutenberg.org/cache/epub/2701/pg2701.txt

Save the file to moby.txt (use this for questions 1-10) and make another copy moby2.txt (for question 11)

1) How many lines, words, characters does the book have?
2) Delete all quotation marks from the file
3) Delete lines 1-30, 36-535 and 21750-22108 (pro-tip: start from the end or you need to recalculate the line numbers!)
4) Count lines that contain the word “whale” (case sensitive; only whale, not whales / whaleman / etc.)
5) Count lines that contain the word “Ahab.” (case sensitive; with a dot at the end!)
6) Count lines that contain “Whale” (case sensitive; only if at the beginning of a line)
7) Count lines that contain a 4 digit number
8) Change every occurrence of “CHAPTER” to “Chapter”
9) Count lines that contain words with at least 2 capital letters in a row
10) Replace every occurrence of the word “Ahab” with “Luke Skywalker”, “harpoon” with “light saber”, “whale” with “sith lord” and “Moby Dick” with “Darth Vader”
11) Do steps 2, 3, 8, 10 again with moby2.txt but this time use a script file for sed!
12) If you are done with everything, take the regex quiz: http://regexone.com/

Help:

Tutorial: http://gnosis.cx/publish/programming/regular_expressions.html

Interactive regex validators: http://www.regexr.com/ and http://rubular.com/