Unix Shell Programming

Shell Programing Basics
+
Writing Shell Programs

Roshan Chitrakar, PhD
roshanchi@gmail.com

Stream editor - sed

- It is ***more efficient than ed*** for non-interactive edits and can handle larger files.
- The ed commands are done only on the current line, unless we specify otherwise. ***The sed commands are done on every line***, unless we specify otherwise.

  ed followed by ***s/He/She/***

  $ ***sed 's/He/She/' cars***
- The quotation marks around the editing command were not necessary, but it is good idea to wrap REs in quotation marks so that the shell does not interpret the metacharacters and spaces.
- ***cars*** is just an input file. Its ***contents are not altered***. The edited text is only sent to the standard output, not put back into the file.

  $ ***sed 's/He/She/' cars > newcars*** redirects the output into a file.

Multiple commands

- each editing command is preceded by the -e option

  $ ***sed -e 's/car/auto/' -e 's/petrol/gas/' cars > newcars***

- We could split a long command over several lines

  $ ***sed -e 's/car/auto/' ***

  > ***-e 's/petrol/gas/' ***

  > ***-e 's/tickets/fines/' cars > newcars***

- Or, we could use only one set of quotation marks :-

  $ ***sed***

  > ***'s/car/auto/***

  > ***s/petrol/gas/***

  > ***s/tickets/fines/' cars > newcars***

$ ***sed '/Man/d***

> ***/car/s//auto/***

> ***/petrol/s//gas/***

> ***/tickets/s//fines/' cars > newcars***

Quiet operation - -n

- The -n option and sed 's p command allow us to display only certain lines of a file.

  $ ***sed -n '4p' cars*** displays line 4 only.

- And this example makes sed do the same as grep :

  $ ***sed -n '/model/p' cars***

Standard Input

- sed operating on its standard input:

  $ ***date | sed 's/:/ /g' | wc -w***

  8

cut

- Cuts out various fields of data from a data file or the output of a command
- Syntax : ***cut -cchars file***
  - where ***chars*** specifies what characters you want to extract from each line of file. e.g. ***-c5*** ; ***-c1,13,50*** ; ***-c20-50*** or ***-c5-***
- If file is not specified, cut reads its input from standard input, meaning that you can use cut as a filter in a pipeline.
- Examples : -

  $ ***who | cut -c1-8***

  $ ***who | cut -c10-16 | sort***

The -d and -f Options

- *Syntax :* ***cut -ddchar –ffields file***
  - where d means delimeter (tab is the default) and f stands for field
- Examples: -

$ cut -d: -f1 /etc/passwd

$ cut -d: -f1,6 /etc/passwd

paste

- A kind of the inverse of cut - instead of breaking lines apart, it puts them together.
- Syntax: paste files e.g. paste names addresses
- More than two files can be joined. Files are joined line by line separated by a tab.

The -d Option

- you can specify the -d option with the format -dchars to separate fields by any character other than tab.

  $ paste -d'+' names addresses numbers

The -s Option

- pastes together lines from the same file, not from alternate files.

  $ paste -s names

  $ ls | paste -d' ' -s

tr

- Translates characters from standard input.
- Syntax: tr from-chars to-chars
- $ tr e x < intro
- $ cut -d: -f1,6 /etc/passwd | tr : '\11'
  - here 11 is the octal value of tab character.
- $ tr '[a-z]' '[A-Z]' < intro

The -s Option

- "squeeze" out multiple occurrences of characters in to-chars
- $ tr -s ':' '\11'
- $ tr –s ' ' ' ' < lotsaspaces # each pair of quotes enclose a space

The –d Option

- Deletes single characters from a stream of input
- Syntax : tr -d from-chars
- $ tr -d ' ' < intro

sort

- takes each line of the specified input file and sorts it into ascending order.

  $ sort names

- The -u Option : eliminates duplicate lines from the output.

  $ sort -u names

- The -r Option : Use the -r option to reverse the order of the sort:

  $ sort -r names

- The -o Option : specify the output file

  $ sort names > names # wipes out the input file

  $ sort names -o names

- The -n Option : specifies that the first field on the line is to be considered a number, and the data is to be sorted arithmetically

  $ sort -n data

Skipping Fields

- You could tell sort to skip past the first number on the line by using the option +1n instead of -n. The +1 says to skip the first field.
- The -t Option : sort assumes that the fields being skipped are delimited by space or tab characters. The -t option says otherwise.
- $ sort +2n -t: /etc/passwd

uniq

- Syntax: ***uniq in_file out_file***
- Copies in_file to out_file, removing any duplicate lines

  $ uniq names

  $ sort names | uniq

- **The -d Option** : finds the duplicate entries in a file

  $ sort names | uniq -d

  $ sort /etc/passwd | cut -f1 -d: | uniq -d

Built-in Integer Arithmetic

- The POSIX standard Unix Shell provides a mechanism for performing integer arithmetic on shell variables called arithmetic expansion

  $((expression))

  - expression may contain shell variables and operators. Valid shell variables are those that contain numeric values and valid operators are taken from the C programming language.

- **Examples: -**

  echo $((i+1))

  $ echo $((a = a + 1))

  echo $((i = (i + 10) * j))

  i=$((i * 5))

  result=$((i >= 0 && i <= 100))

  $ i=$((100 * 200 / 10))

  $ j=$((i < 1000)) # If i is < 1000, set j = 0; otherwise 1

  $ echo $i $j

Quotes and Command Substitution

{Refer to ***the practical exercises***

and

Stephen G. Kochan, Unix Shell Programming, p165-187}

Arguments

- Whenever you execute a shell program, the shell automatically stores the first argument in the special shell variable 1, the second argument in the variable 2 ($1, $2), and so on.
- These special variables are called ***positional parameters***.
- The special shell variable $# gets set to ***the number of arguments*** that were typed on the command line.
- The special variable $* references ***all the arguments*** passed to the program.

Arguments - ${n}

- Used to access ***more than 9 (nine)*** arguments.
- You cannot access the tenth and greater arguments with $10, $11, and so on. If you try to do so, the shell actually substitutes the value of ***$1 followed by a 0***.
- Instead, the format ***${n}*** must be used. So to directly access argument 10, you must write ***${10}***.

shift

- Allows you to effectively left shift your positional parameters.
- ***shift 3*** has the same effect as performing three separate shifts.

{ Refer to the practical lessons for more understanding.}