Working With File Data

Linux System Administration
COMP2018 Fall 2019

Standard Input and Output

- Regular files contain bytes, and the ways we work with them are very consistent regardless of what program wants to work with their data, or what it wants that data for
- UNIX and UNIX-like systems implement the concept of treating streams of data as files
- Input coming from the keyboard is a stream of data that has the special name stdin (standard input)
- Output going to the screen is a stream of data that has the special name stdout (standard output)
- Error messages going to the screen is a stream of data that has the special name stderr (standard error)

I/O Independence

- Commands that can use data from stdin don't generally care how the data gets to stdin
- Commands that send data to stdout don't generally care where stdout actually goes, the same is true of stderr
- Using this approach means programs can work with data in the same way, no matter where it comes from or is going to
- Programs can override this, but rarely do - the passwd program is an example of one which overrides this behaviour to force input to come from an actual keyboard

Output Redirection

- Since the output that goes to the screen is treated like a file, it is simple to use the shell special character > to tell it to send that output to a regular file instead of displaying it on a screen

- It takes one of the following forms:
  - command > file - overwrites the file with stdout from the command
  - command >> file - appends stdout from the command to the end of the existing file data
  - command >& file - overwrites the file with stderr from the command
  - command >>& file - appends stderr from the command to the end of the existing file

- In both cases, bash will create the output file if it did not already exist

- The special file /dev/null used as an output file will discard output

cat

- cat is used to take input from various sources, including regular files, and send it to your output, typically your screen
- Viewing a file can be as simple as cat file
- cat can use multiple files as the source, as in cat a b c
- cat without file(s) to get input from will take input from stdin
- The keyboard is the default stdin for many commands, input from the keyboard is terminated by typing ^D (Control-d) on a line by itself (corresponds to the ascii code for end of file)

Input Redirection

- bash supports the special character < to tell it to get input from wherever we want instead of commands having to get their input from the keyboard or be coded to accept filenames on the command line
- The 3 forms for this are:
  - command < file - gets stdin from file
  - command << TEXT - bash gets stdin from the keyboard as input until it sees TEXT on a line by itself, then runs the command and feeds it your input text as if you typed it out while the command was running - this is called a HERE document and is mostly used for scripting
  - command <<< "Some text" - bash feeds Some text to the command when it runs the command, as if you had typed it in while the command ran - acts like echo "Some text" | command and is mostly used for scripting

more

- If the output printed on your screen is more than a screenful, the cat command might not be your best choice to view the output
- The more command was written to solve this by displaying a screenful of text at a time, allowing the user to decide how to proceed at the end of each screenful
- The more command has very similar syntax to the cat command
- Pressing space shows the next screenful (or page), b shows the previous page where possible, /, and n are used to search for text, h shows help, and q immediately terminates the more command
- When the last screenful is displayed, the more command automatically terminates - the less command was created to change this behaviour
- more will get input from stdin if you don't give it one or more filenames to use, making it useful for pipelines

head And tail

- If you only want the first lines from a file of text, or the last ones, cat and more (or less) can be unwieldy
- The head command is used to display the first lines of text from one or more files, showing 10 lines from each by default
- The tail command is used to display the last lines of text from one or more files, showing 10 lines from each by default
- Both support displaying any specific set of lines from their input
- They both will use stdin for input data if you do not give them files to work on, making them useful in pipelines

grep

- grep is the global regular expression printer program
- It is used to find text in files
- grep pattern file(s) will display the lines of text matching the pattern you gave from the file(s) you specified
- grep supports multiple input files or using stdin for source data
- Interesting options include -i, -s, -v, -n, -H, -r, -c, and -l

Regular Expressions

- The pattern grep expects is in the form of a regular expression (or regex)
- Regular expression patterns are used by many commands that can use pattern matching
- Simple regular expressions include the ., *, [ ] characters similar to file name globbing, but not the same
- There are also extended regular expressions which add more special characters for enhanced pattern matching
- The command fgrep ignores special characters in the pattern, egrep allows extended regular expressions

Command Line Pipes

- bash supports connecting the stdout from a command to the stdin of another command, creating a pipeline between them

- bash allows multiple pipes on the command line enabling the creation of powerful command line combinations

- bash uses the vertical bar (|) to create the pipe(s)

sort

- A common task when working with text is to put the output in some order such as numeric or alphabetic
- The sort command does this for us
- sort supports many options for sorting and can work on multiple files or use stdin for source data
- Interesting options include -r, -n, -h, -k, -f, -u, and -t
- -k does not necessarily sort what you might expect

nano

- Modifying text files is done using a text editing program
- In a GUI environment, you might use a tool like textedit, gedit, atom, or brackets for this purpose
- On the Linux command line, you will likely use either nano or vim (a.k.a. vi)
- nano is designed to be more intuitive for windows users with the start typing anywhere and use arrow keys to move around approach
- Control-key combinations provide control for the editor to do things like read, write, search, and quit
- There are many good references for nano, with https://www.howtogeek.com/howto/42980/the-beginners-guide-to-nano-the-linux-command-line-text-editor/ as an example tutorial

vim

- vim (vi improved) is an older editor that operates in modes, with many more commands and options for file editing than nano

- It has a steeper learning curve than nano but is universally available in the UNIX and UNIX-like market

- It came from the ex editor which supports complex editing of files on non-cursor-addressable terminal devices

- The manual is the definitive reference for using vim, but https://blog.interlinked.org/tutorials/vim_tutorial.html is also a good concise introduction

sed

- sed is the stream editor
- it is based on the original ed editor which provided editing of text files on non-cursor-addressable terminal devices including teletypes
- Using sed involves creating a regular expression to match what you want to change in a file, and then supplying the editing commands to make the changes you want
- sed is used to edit files from within scripts, as well as modify data passing through a command pipeline
- There is an excellent sed tutorial at http://www.grymoire.com/Unix/Sed.html

awk

- awk is a command line tool for manipulating text
- It has pattern matching and output generation scripting syntax that allows you to perform simple substitutions or generate complex reports
- awk scripts can be saved to files, allowing for complex scripts without having to memorize how to get things done with awk
- http://www.grymoire.com/Unix/Awk.html is a good introduction to using awk

Comparing Files

- Comparing file contents can be done with cmp which will tell you if 2 files have distinct content

- Use cmp when the files being compared are binary or you are simply testing if they differ and do not care what the differences are

- If you are comparing text files, you can use diff to see not only if they differ, but what the differences are