File I/O and Processing

EE 107S: Introduction to Linux

Lecture 2

Display input as output: `echo`

- `echo` takes a string argument and outputs it
  - `echo 'Hello, world!'`
- Doesn't sound very useful, but it will be…we'll use it soon

Create/update a file: `touch`

- Updates the date modified of a file
- If the file doesn’t exist, creates it

Wildcards

- The * character matches anything in the CWD
  - Given files `file1` and `file2`: `file*` would match both names
- * can be used for directories too
  - Given files `dir1/fileA` and `dir2/fileA`: `dir*/fileA` would match both files

Searching for files/dirs: `find`

- Lists all files/directories at the path given as a parameter (including nested files/directories)
- Specify whether to show just files or directories with `` `-type f` `` or `` `-type d` ``, respectively

Searching for files/dirs: `find`

- Use `-name` argument with wildcards to search for files/directories
  - `find . -name ‘fullname’ -type d`
  - `find . -name ‘*part*’ -type f`

Exercise

Find all directories in your home directory that have the word ‘dir’ in them

Possible Solution

```
cd ~
find . -name ‘*dir*’ -type d
```

Viewing files: `cat`

- Accepts a parameter of a file name
  - This parameter is optional, in which case it can accept input from `stdin` (more on this later)
- Outputs the contents of the file

Viewing files: `less`

- Works similar to `cat`, but has a scrollable view
- Very useful when displaying lots of output
- Press ‘q’ to exit, like `man`

Viewing parts of a file: `head`/`tail`

- `head` outputs the first 10 lines of a file
- `tail` outputs the last 10 lines of a file
- Both commands accept the `-n` flag, which specifies the number of lines to show

I/O == `stdin`/`stdout`

- Input is captured from ‘standard input’ (`stdin`)
- Output is displayed to ‘standard output’ (`stdout`)
- Input and output can be redirected
  - Files can be fed into input, instead of typing them
  - Outputs can be sent to a file instead of to the terminal

Redirecting output

- The output of a command can be sent to a file by adding the '>' character and then specifying a file name at the end of the command (overwrites the file)
  - `ls -l > files.txt`
- '>>' can be used to append to an output file
  - `echo 'End of file' >> files.txt`

Redirecting input

- A file can be used as input instead of the keyboard by adding the ‘<‘ character and then specifying a file name at the end of the command
  - `cat < files.txt` (same result as `cat files.txt`)
  - `less < files.txt` (same result as `less files.txt`)

Pipes

- Takes the output of one command and sends it to the input of the next
- Allows you to chain commands together
  - May not be immediately obvious, but this is extremely powerful
- Separate commands with the `|` character
  - `cat files.txt | less`

Intro to text processing: `tr`

- Translate one character to another
- Delete characters
- Compress consecutive characters into a single character

Delimeters

- A character that separates parts of a line
- Common delimeters are ‘ ‘ and ‘,’
- Many of the text processing tools allow you to specify alternate delimeters

Bonus command: `curl`

Used to interact with webpages

```
curl http://a-dev.me/107s/gradebook.txt > gradebook.txt
```

Splitting by columns: `cut`

- Separates each line into columns by delimeter
  - Default delimeter is tab character
- Accepts a range of columns to output
  - Given as ’1-3’, ‘-3’, ‘3-’, etc.
  - Ranges are inclusive, column numbers start at 1

Exercise

Print out a column of first names from the gradebook file

Possible solution

```
cut -d',' -f2 gradebook.txt | cut -d' ' -f1
```

Regular expressions

- A more flexible way of matching strings
  - Allow a lot more control than wildcards
- '[0-9]+' matches a string of one or more numbers
  - 'ab0123cde' and '4ef' match, but 'abcde' doesn't
- '[A-z]*' matches a string of zero or more letters (upper case and lower case)
  - 'abCdef1234', '1bc', and '1234' all match
    - Why does 1234 match? Because * means **zero** or more letters

Regular expressions

- Structured as ’character classes’ followed by number of instances
  - Character classes can be in square braces like [0-9]
  - Any character in the range matches, regardless of order
- Number of instances given by +, *, ?, etc.
  - +: one or more
  - *: zero or more
  - ?: exactly zero or one

Regular expressions

- . will match any single character
  - Therefore .* will match every single line
- Character classes can be combined together into a large regular expression

Regular Expression examples

- Date: [0-9][0-9]/[0-9][0-9]/[0-9]{2}
- Hex number: x?[a-fA-F0-9]+
- Email address: [A-z0-9]+@[a-z]+\.com

Exercise

Write a regular expression that would match a phone number, where the area code may or may not have parenthesis, and the remaining digits may or may not be hyphenated

Possible solution

\(?[0-9]{3}\)? ?[0-9]{3} ?-? ?[0-9]{4}

Filtering output: `grep`

- Reads input (file or `stdin`) line by line
- Extracts lines that match regular expression
- Useful for searching through many files

Pretty much everything else: awk

- Reads input (file or `stdin`) line by line
- Format of an awk operation is '`regex1 { operation }; regex2 { operation } ...`'
- Common operation is `print`
  - There are special variables `$0`, `$1`, … that can be used for columns
- The operations are similar to C code, which means you can have variables!

Exercise

Swap the A3 and A4 columns in the gradebook

Possible solution

```
awk ‘/.*/ {temp=$6; $6=$5; $5=temp; print $0;}’ gradebook.txt
```

Assignment

- Will be posted some time tonight
- It will exercise your text processing skills

Tutorials

- Regular expressions
  - Regular expression ‘debugger’
- sed
- awk