Lab 03 - Bash - The Journey Continues

CS 1XA3

Jan. 22, 2018

Hidden Files and Your Bash Profile

- Hidden files begin with .
- They aren't listed by default with ls
- Use ls -a to list all files

Check out what hidden files exist in your HOME directory

Hidden Files and Your Bash Profile

- The hidden file .bash_profile is a script thats loaded everytime you start a terminal session
- Try adding the following to your bash profile

```
alias ll="ls -la"
```

- To load your changes without restarting your session, use

```
source ~/.bash_profile
```

Exercise: try customizing you're ls command the way you like it

The PATH Environment Variable

- Every linux system has a set of Environment Variables
- Of all these variables, PATH is inarguably the most important
- Everything command / program accessible is either in your current working directory or PATH
- See whats in your PATH variable by executing

  ```
  echo $PATH
  ```

- You can see where a command is located by executing

  ```
  which command
  ```

Exercise: Locate some of your favourite commands, are they in your path?

Glob Patterns

Globbing is a wildcard technique used for pattern matching files / directories

- * Matches any quantity of character (including empty)
- ? Matches occurrence of a **single** character
- \ Escapes a special character (like space)
- Matches one occurrence of any character within brackets `[...]`

Special Note: Trickiness with Spaces

- Bash commands use spaces to seperate arguments to a command
- this can cause disruption when piping input that possibly has spaces
- quotations aren't used to specify string values like other languages, use them to stop spaces in names from creating seperate arguments
- whenever you use glob patterns that can expand to something with spaces, it's a good idea to wrap it in quotations

IO Redirection: StdOut and StdErr

- $>$ overwrite / create a file from stdout

  ```
  echo "Hello World" > tmp.txt
  ```

- $>>$ append / create a file from stdout

  ```
  echo "Goodbye Existential Dread" >> tmp.txt
  ```

- Use 2 $>$ or 2 $>>$ to write / append from stderr
- Use & $>$ or & $>>$ to write / append from both stdout and stderr

IO Redirection: Piping

The Pipe Operator — allows you to build Compound Commands, making use of the output of one command for the input of another

- Try finding all files associated with your bash

  ```
  ls -la | grep bash
  ```

- Now narrow things down by extending the pipe

  ```
  ls -la | grep bash | grep profile
  ```

Exercise: try building a command with as many pipes/different commands as you can

Searching file contents with grep

Syntax

```
grep -flags pattern input
```

Important flags to make note of:

- Recursive, iterate through directories / subdirectories

  ```
  grep -r pattern /dir
  ```

- Search **only** files in specified directory, not subdirectories

  ```
  grep -R pattern /dir
  ```

- Reverse grep, only show lines excluding pattern

  ```
  grep -v pattern input
  ```

Add the -i flag to ignore cases when matching a **pattern** and the -l flag to list only the filename (good for piping into xargs)

Word Counting

Syntax

```
wc -flags input
```

The word count command returns a number for

- word count

  ```
  wc -w input
  ```

- line count

  ```
  wc -l input
  ```

- character count

  ```
  wc -m input
  ```

Exercse: try combining wc with grep

Replacing file contents with sed

Syntax

```
sed -flags pattern input
```

The main flag to be concerned with is -i, which specifies to overwrite the input file. Of more interest is the patterns you can specify

- insert a new line after pattern

  ```
  sed '/pattern/a line to insert' input
  ```

- insert a new line before pattern

  ```
  sed '/pattern/i line to insert' input
  ```

- replace a line with pattern

  ```
  sed '/pattern/c line to replace' input
  ```

Note: you can use numbers in place of /pattern/ to specify a line directly

Replacing file contents with sed

Even more patterns!

- substitute one word for another

  ```
  sed 's/old/new/g' input
  ```

- delete a word

  ```
  sed 's/old//g' input
  ```

- delete any line with pattern

  ```
  sed '/pattern/d' input
  ```

Finding files with .... find

Syntax

```
find searchdir -name pattern -flag1 inp1 -flag2 inp2 ...
```

- limit to file (f) or directory (d) with type

  ```
  find dir -name pattern -type f
  ```

- execute a command cm without prompt

  ```
  find dir -name pattern -exec cm {} \;
  ```

- execute a command cm with prompt

  ```
  find dir -name pattern -ok cm {} \;
  ```

- print files starting at current directory

  ```
  find dir -name pattern -print
  ```

Finding files with .... find

When using -exec or -ok flags with format

```
find dir -name pattern -ok cm {} \;
```

- {} specifies the input
- The ending term specifies to execute each command seperately

  ```
  find dir -name pattern -ok rm {} \;
  # executes rm many times
  ```

- The + ending specifies to add all the arguments at once

  ```
  find dir -name pattern -ok rm {} +
  # executes rm file1 file2 ..
  ```

Note: the -ok and -exec directives can be tricky / work unexpectably because of how bash handles white-spaces

More Powerful Piping with xargs

Syntax

```
command1 | xargs -flag command2
command1 | xargs -flag -I label command2 label
```

- Piping takes it's input as a whole, if you want to split the input into manageable chunks you must use xargs
- Try removing a directories contents by piping ls and rm
- Note: the flag -I is sometimes necessary because of how piping assumes the desired input is the final argument to the command

A better find -exec with xargs

- Because of spaces (yes they're a pain), find -exec commands can often go wrong
- Use the flag -print0 to output the result of a find seperated by a special character

```
find dir -name pattern -print0
```

- The results can then be parsed by xargs with the -0 flag so that arguments are seperated by the special character not spaces

```
find dir -name pattern -print0 | xargs -0 command
```

Busy Work

- Clone the github repo

  `git clone https://github.com/vincentarelbundock/Rdatase`

- This repo contains **ALOT** of data, mostly in **csv** tables
- Try finding all #TODO comments and putting them into a file
- Try copying all the **csv** files into a single tmp directory
- Try finding the tables that have to do with (have info about)
  - Income
  - Gender
  - Contain Estimations