File Processing

Class 4

Administrative

- video on
- “Hello” in chat box
- logged on and at bash terminal prompt
- questions?

Some Files to Play With

- we need some common files to use for today's and future work
- cd to the place where you keep your files for this course
- issue the following command (note the period at the end):
  `$ rsync -vruz username@sand.truman.edu:/tmp/states .`
- if you get prompted to trust the authenticity of the host, accept it; if you get prompted for your password, use your Truman password
- you should now have a new directory named "states", which contains 50 files, each of which is named for a state and each of which contains the names of cities

- if you get an error, let me know

Viewing File Contents

- a file is just a sequence of bytes
- when you ask the operating system for some stuff from a file, you just get raw bytes
- there are two main flavors of file
  1. binary files
  2. text files
- every file is a binary file in the sense that it contains bytes
- text files, however, contain only bytes that correspond to ASCII characters
- one of those bytes represents the newline character, interpreted as the end of a line
- thus text files are easy to interpret as a sequence of lines each of which is a sequence of characters

Binary Files

- all files contain bytes
- the bytes encode some information, that we call data
- binary files contain data that is strictly designed to be read by computer programs
- some are open standard formats, e.g., jpeg and pdf for images
- some are proprietary, e.g., xls for spreadsheets and dwg for CAD drawings
- this is not the same as structured text files that conform to a format standard such as csv or xml

Text Files

- text files contain data that can be read either by a human or by a computer program
- most files on a Unix system are text files
- there is a powerful need to be able to process text files
- many tools have been developed for this

Viewing File Contents

- for a small file, use the command cat (concatenate)
  `$ cat Delaware`
- the reason it's called cat is that it concatenates to the screen the contents of multiple command line arguments:
  `$ cat Alaska Hawaii`

- for large files that you wish to browse through, use less:
  `$ less California`
- movement commands
  - `:g` go to top of file
  - `:G` go to bottom of file
  - `:300g` go to line 300
  - PageDown or `:f` to scroll to the next screen
  - 100 PageDown or `:100f` to go forward 100 lines
  - PageUp or `:b` to go back one screen

More Less Magic

- less is an extremely powerful text file viewer
- common commands in less
  - `/string` search for the next occurrence of string
  - `n` go to the next occurrence of same search
  - `:q` quit
- `$ less -N foo` view the file with line numbers — very useful for source code
- `$ less Michigan Missouri Mississippi`
  open Michigan in less
- `:n` now open Missouri in less
- `:p` go back to Michigan
- `:f` show the current file name

File Size

- wc (word count) is specific for file size

```
$ ls -l Nebraska
-rw-rw-r-- 1 jbeck jbeck 175 Aug 23 10:17 Nebraska
$ wc Nebraska
 19  24 175 Nebraska
```

- default output is four columns: number of lines, number of whitespace-separated words, number of characters, and filename from command line
- generates only one number with -l, -w, or -c respectively
- very useful for shell scripts to see how big the result of a command was — it's in filesize.sh

Shell Metacharacters

- many characters other than letters, digits, underscores, and spaces have special meaning to the shell
- these are called shell metacharacters
- cannot be used in shell commands as literal characters without special handling

character	purpose	example
return	submit command to shell	
space	separate elements on command line	`$ ls /etc`
#	start a comment	`$ foo # a comment`
"	quote a string with variable interpolation	`"$foo"`
$	end line, dereference variable, capture output of command	`$ $(wc -l)`

character	purpose	example
&	background execution	`$ emacs foo &`
'	quote a string, no interpolation	'$foo'
( )	execute command in a subshell	`$ (wc -l)`
*	wildcard that matches zero or more characters	`$ ls *`
[]	regular expression grouping	`$ ls [A-Z]`
^	regular expression negation and line-begin	`$ ls [^A-Z]`
|	create a pipe between two commands	`$ ls | wc -l`
< > >>	redirection	`$ wc -l > foo`

character	purpose	example
{ }	group a variable name	`${name}old`
?	wildcard that matches exactly one character	`$ ls -d /e??`
/	root directory and pathname separator	`$ ls /etc/hosts`
\	escape another metacharacter	`$ wc New\ Jersey`

Shell Metacharacters

- metacharacters can be used together to build shell expressions
  `$ ls [A-M]*` # list all files that begin with the letters A through M
- `$ ls [^A-M]*` # list all files that do not begin with A through M
- `$ ls *_*` # list all files that have an underscore in the name

grep

- an incredibly powerful file-searching command
- Global search for Regular Expression and Print results
- three flavors
  1. grep -F: fast or fixed with no pattern matching; use this when you are searching for a literal string
  2. grep: simple pattern matching; use this when you need simple regular expressions
  3. grep -E: slowest, full regular expressions; use this when you need powerful regular expressions
- note: the wildcards and regular expressions in the grep program are somewhat different from those in the shell

grep -F Examples

- `$ grep -F Santa California`
  display all cities in California containing Santa
- `$ grep -nF Santa California`
  display all cities in California containing Santa, with line numbers
- `$ grep -F Santa *`
  search for all cities in all states containing Santa
- `$ grep -lF Santa *`
  list the names of files that contain Santa
- `$ grep -LF Santa *`
  list the names of files that do not contain Santa
- `$ grep -vF Santa California`
  display all cities in California that do not contain Santa

Simple Regular Expressions

- `$ grep -F Bar *`
  displays cities that contain "Bar" anywhere in the name. This includes "Great Barrington" and "Wilkes-Barre"
- `$ grep '^Bar' *`
  displays cities that begin with "Bar". Need quotes to protect the ^ from the shell, which would treat it as a shell metacharacter instead of passing it to the grep program
- `$ grep 'San\>' California`
  displays cities in California that have a word that ends in San
- `$ grep '^Barre$' *`
  displays cities in which the entire line is exactly Barre
- `$ grep 'f..t' *`
  displays cities with f and t separated by exactly two characters

Complex Regular Expressions

- `$ grep -E '[a-z]{8}' West_Virginia`
  displays cities in West Virginia which have at least 8 consecutive lowercase letters
- `$ grep -E '^..[AEIOUaeiou]' Vermont`
  displays cities in Vermont whose third letter is either an uppercase or lowercase vowel
- `$ grep -E '^..[^AEIOUaeiou]' Vermont`
  displays cities in Vermont whose third letter is not either an uppercase or lowercase vowel
- `$ grep -E 'foot|fort' *`
  displays cities that contain either the pattern foot or the pattern fort