Lecture 3

Tonight we dine in shell

Hands-On Unix System Administration DeCal
2012-09-17

Review

- $1, $2, ...; $@, $*, $#, $0, $?
- environment variables
- env, export
- $HOME, $PATH
- $PS1=n\[\e[0;31m\]\u\[\e[m\]@\[\e[1;34m\]\w \[\e[2;90m\]\@\n\[\e[m\]\[\e[0;35m\]\h\[\e[m\]\[\e[0;31m\]\$\[\e[m\]\[\e[0;32m\]
- quotes, ’ and ”
- aliases
- globbing
- backticks (`)
- pipes (|)

tee

- essentially a pipe
- mostly used to do

```
$ sudo <command> | sudo tee <file>
```

find

- google search on steroids for file system
- regexes, depths, mtimes, types, groups, users, etc.

```
$ find -L /usr/ports/packages -type l -exec rm -- {} +
```

```
$ find / -newer ttt -user wnj -print
```

```
$ find a b -type f ! -name 'crazy' -printf '%f\n'
```

xargs

- most used after a find:

```
  $ find . -name 'randomcrapfile' | xargs rm
```

```
  $ find . -name 'filetobemoved' | xargs -I {} mv
 {} somefolder
```

-print0

  used with find to print NUL character generally for xargs

-0

used in xargs in conjunction with -print0 for `find' for files with spaces

locate

- cached google search for file system
- precompiled database
- faster than `find`, but not as detailed in search

for and while loops

- built into shell

Structure:

```
for {something}
do
   somecommand
   someothercommand
done
```

One-liner, with semi-colons:

```
 $ for {something}; do somecommand;
 someothercommand; done
```

`for` and `while` loops

Structure

```
while {something}
do
    somecommand
    someothercommand
done
```

One-liner, with semi-colons:

```
 $ while {some expression}; do somecommand;
 someothercommand; done
```

awk

- full programming language
- generally used to do (simple) regular expressions on files
- More info at: https://en.wikipedia.org/wiki/Awk, http://www.grymoire.com/Unix/Awk.html

Moar shell-fu

- `grep`
- `sed`
- `cut`
- `tr`
- `wc`
- `sort`
- `head`
- `tail`

tr

```
tr [OPTION]... SET1 [SET2]
```

- SET1 and SET2 define ordered sets of characters (characters of input that ‘tr’ operates on)
- Function is to replace, squeeze, remove characters from its input
  - No filenames to provide as arguments
- Reads stream from stdin, writes to stdout

tr

```
tr [OPTION]... SET1 [SET2]
```

- Examples

```
$ echo “the quick brown fox” | tr abc def
```

  - Replace characters in SET1 with corresponding characters in SET2

    can use ranges (e.g, a-z A-Z)

```
$ echo “the quick brown fox” | tr a-z A-Z
```

- Commonly used options

    -d, --delete

    -s, --squeeze-repeats

sort

```
sort [OPTION]... [FILE]...
```

- Useful options:

  -d, --dictionary-order

    - Consider blanks and alphanumeric characters

  -n, --numeric-sort

    Sort by numerical value

  -r, --reverse

    Reverse the result of comparisons

  -f, --ignore-case

  -k(column), -nk2 means sort column 2 numerically

cut

```
cut [OPTION]... [FILE]...
```

- Print selected parts of lines from each FILE (or stdin) to stdout
- Useful options:

  -d, --delimter=DELIM

  -f, --fields=LIST

head/tail

- View first/last parts of file
- Useful for viewing logs
- Default: view first/last 10 lines
- Common options
  - -n, --lines=N
    - Output first/last N lines,
- tail -f <file>
  - “follow” the file, output appended data as <file> grows
- tail -n +N <file> or tail --lines=+N <file>
  - Starting from N, output N to rest of file
- head --lines=-N <file>
  - View everything but the last N lines in <file>

wc

- Word count
- Print newline, word, and byte counts
- wc -l, print newline count (count lines)
- wc -w, print word count

Regular Expressions (regex)

- String matching
  - Set of metacharacters let you search for text that fits criteria
- Text editors, utilities, programming languages
  - grep, sed, awk, vi(m)
  - Perl, Ruby, etc.
- Many flavors, POSIX BRE
- Regex ≠ globs/wildcards
  - Different sets of metacharacters used for different purposes
    - Filename expansion vs. string matching
  - The shell itself does not recognize RE's. It is the commands and utilities, that do.

Basic Regex

\ (backslash) turn off special meaning of following character (escaping)

. (period) match any single character

[..] (bracket expression)

- Matches ONE of any of the enclosed characters
- Hyphens indicate a range of characters (a-z, A-Z, 0-9)

* (a quantifier) match any number or none of preceding character

- e.g, a* matches 'abc', 'bc'
- aa* matches 'abc' but not 'bc'

Anchors (regex)

Specify where matching text should be

^ Match following regex at beginning of line

$ Match preceding regex at end of line

Examples

Regex	Matches
tolstoy	tolstoy, anywhere
^tolstoy	tolstoy, beginning of line
tolstoy$	tolstoy, end of line
^tolstoy$	a line containing exactly 'tolstoy' and nothing else
[Tt]olstoy	Either Tolstoy or tolstoy
tol.toy	tol, followed by any character, followed by toy
Tol.*toy	tol, any sequence of 0 or more characters, followed by toy

grep

```
grep [OPTIONS] PATTERN [FILE...]
```

- Match text (PATTERN can be w/ or w/o regex)
- Grep (BRE), egrep/grep -E (ERE), fgrep/grep -F (match fixed strings)
- Can search with fixed strings, or with regexes
- Common options

  -i case insensitive search

  -l list names of files instead of printing the actual matching lines

  -v print lines that DON'T match the pattern

  -e <pattern>

  - Use multiple -e options to search with multiple patterns

sed

```
sed [OPTIONS] 'COMMAND' [FILE...]
```

- Stream editor for filtering and transforming text on an input stream (file or input from pipeline)
- Commonly used to perform text substitution in a pipeline
- 'COMMAND' is often substituting, appending, inserting, deleting text

SUBSTITUTION:

```
sed 's/old value/new value/(flags)' <file>
```

'old value' can be a regex

sed substitution

```
sed 's/old value/new value/(flags)' <file>
```

- Common flags
  - n – replace nth instance of pattern with replacement
  - g – replace ALL instances of pattern with replacement
  - Without flags, sed replaces first instance of 'old value' with 'new value' in each line

```
$ echo “I hate this decal” | sed 's/hate/love/'
```

```
$ echo “hi hi hi” | sed 's/hi/bye/'
```

sed: deletion

```
sed '{what to find} d' <file>         # deletion
```

- {what to find} can be:
  - Range of lines: sed '1,3d' <myfile.txt>
  - Regex: sed '/#/d' (delete comments maybe?)

Other sed commands include insertion (i) and appending text (a)

Common options

- Many commands share some common options:

```
-h/--help
-v/--verbose
-d/--debug
-f/--force or file input
-R recursive
```