Bash Oneliners

07-131 Great Practical Ideas in CS

Instructors: Eduardo Feo-Flushing & Giselle Reis

Course website: https://web2.qatar.cmu.edu/cs/07131/

01.

Unix Process Comm.

...

Unix process interacts with the world

- stdin
- args
- env
- fs
- network

- stdout
- stderr
- exitcode
- fs
- network

We can script some of these

- **stdin**
- **args**
- **env**
- fs
- network

- **stdout**
- **stderr**
- **exitcode**
- fs
- network

Input & Output (streams)

- stdin - standard input (file descriptor = 0)
  - raw_input, scanf
- stdout - standard output (file descriptor = 1)
  - print, printf
- stderr - standard error (file descriptor = 2)
  - fprint(stderr)

Arguments

- args - command line arguments
- Scripts can access arguments with
  - `$#` = number of arguments given to the script (*different from "argc" in C, which includes program name*)
  - `$1` = first argument, `$2` = second argument, ...
- e.g. `echo Hello World`
  - `$# = 2`
  - `$1 = Hello`
  - `$2 = World`
  - `$0 = echo`

Environment Variables

- A list of key-value pairs
- Essentially the shell's global variables
- Any program can use these variables
  - Access a VAR by `$VAR`
- `printenv` - prints currently set environment variables

Exit Code

- All programs exit with some code
- This is determined by the programmer
- In general, exit 0 means success
- Anything else indicates some error/failure
- Process can access last executed program’s exit code

02.

Scripting I/O (Redirect)

...

Redirection

	Input	Output	Error
Append	*	`[cmd]>>[file]`	`[cmd]2>>[file]`
Read/ Overwrite	`[cmd]<[file]`	`[cmd]>[file]`	`[cmd]2>[file]`

Redirection Tricks

- Redirect one stream to another
  - `[cmd] 2>&1`
- Ignore a stream - redirect stdout to the “null device”
  - `[cmd] > /dev/null`
- Ignore any output from a program (both stdout and stderr)
  - `[cmd] > /dev/null 2>&1`
  - (alternatively) `[cmd] 2> /dev/null 1>&2`
  - (alternatively) `[cmd] > /dev/null 2> /dev/null`

03.

Scripting More

...

Unix Pipes - Intro

- Pipes connect processes by linking **`stdout`** of first process to **`stdin`** of second

- Think of it like function composition:

`g(f(x))` ↔ **`x | f | g`**

Unix Pipes - Syntax

```
<cmd> [ARGS] [REDIRECTS] | <cmd> [ARGS] [REDIRECTS]
```

- The pipe character is ⇧ Shift + \ (i.e. the character above ↵ Return)

Unix Pipes - Warnings

- A few things to keep in mind using pipes:
  - Programs in a series of pipes are run in PARALLEL
    - i.e. if your future programs are dependent on the previous running to completion before starting, don’t use pipes
  - At pipe boundaries, results are buffered
    - i.e. if your future programs cannot handle buffered input, don’t use pipes

Scripting More Things

- You can easily change the scripting environment by setting environment variables before a command:

```
VAR1=value1  VAR2=value2 <cmd> [args]
```

- You can get the exit code of a program easily too:

```
$?
```

- Plus, Bash has all the features of a scripting language, including conditionals, functions, loops, and processing tools
  - But, it's hard to write correct code easily (see: pitfalls)
  - Bash is great for automation and it can make your life (a lot!) easier

Command Substitution

- You've already seen how to match file arguments easily with globs
- It's pretty easy to use command substitution to get the output of a command as a single argument: use $(command)

An example:

```
touch myfile-$(date +%s).txt
```

This creates a file with the current timestamp inputted in the name!

An Aside: The Parable of Knuth and McIlroy

- Jon Bentley, a famous person who improved the speed of quicksort, challenged Donald Knuth to “show off” literate programming with a solution to a sorting problem and asked Doug McIlroy to critique it
- Knuth is famous -- he wrote The Art of Computer Programming, among other things
- McIlroy is also famous -- he literally invented pipes
- Knuth wrote a 10+ page Pascal program -- McIlroy wrote a well-explained 6-line Bash script

Find the original story here

The Parable of Knuth and McIlory (Continued)

```
# Split text into words by replacing non-word characters with newlines
tr -cs A-Za-z '\n' |
# Convert uppercase to lowercase
tr A-Z a-z |
# Sort so that identical words occur adjacently
sort |
# Count occurrences of each line
uniq -c |
# Sort numerically by decreasing number of word occurrences
sort -rn |
# Quit after printing the K specified number of words
 sed ${1}q
```

find

- We use **`grep`** to search through file *contents*
- **`find`** does the same thing for file *names* for deep recursive file system searches
  - Walk a file hierarchy and do something for each things that matches

```
find <directory> -regex '<regex>'

find <directory> -name '<glob>'
```

xargs

- Read input from **`stdin`** and execute argument command with arguments constructed from **`stdin`**

```
xargs <command>
```

Examples

- Find all my uses of **`find`**

```
history | grep -E "find .*"
```

- Find all my shell scripts, add permissions, and execute them:
  - -t flag is to also print the commands run
  - -n1 flag specifies run command per line of input (one at a time)

```
find . -name '*.sh' | xargs -t chmod +rwx
find . -name '*.sh' | xargs -t -n1 bash
```

curl

- Make a network request to return a file or webpage located at the argument URL

```
curl <URL>
```

Oneliner Tips

- The best way to get a fully working oneliner is to keep building iteratively
  - Try each step one at a time and see what happens when it runs
- Figure out what you think the steps to do what you want should be, and *then* try to write the script
- You stand on the shoulders of all the programmers before you
  - Use Google/StackOverflow as resources to try and figure out if there's an easy way to do what you want
  - Use man pages, they're made to teach people how to use a tool

Useful resources

Bash One-Liners Explained

A multi-part guide to various bash oneliners explained in detail! (Also has different articles on sed, awk, Perl, among others!)

Bash-Oneliner

A gigantic list of oneliners that will probably have what you want to do with Bash!

Bash Scripting How-To

An introductory article to a wide list of features Bash offers.

PipeLab

Helpful commands for pipelab:

- **Curl** - pulls content from an url
- **Sed** - Edits text (stream editing) (input can be supplied through stdin)
- **Xargs** <command> - Transformed newline separated text in stdin to arguments for the given command
- **Test locally first! Construct iteratively!**

Small secret:

- `./driver/driver` is a bash script
- Wow! (you can hack it if you want
- But it's probably easier to do the lab...)