

Oneliners

Jake Zimmerman

November 25, 2016

Review

We can define variables in bash

```
# set my_variable to the string "hello"  
# (no spaces around the '=')  
$ my_variable="hello"
```

```
# get the value of my_variable and print it  
$ echo $my_variable  
hello
```

```
# print another_var surrounded by other text  
$ another_var="some string"  
$ echo lone${another_var}s  
lonesome strings
```

```
# Sometimes using {...} is important:  
$ echo lone$another_vars  
lone
```

Quotes can be used to group arguments

- ▶ No quotes
 - ▶ spaces separate arguments
- ▶ Single or double quotes
 - ▶ entire quoted argument is one argument
 - ▶ spaces inside don't break it up

Quotes are optional sometimes

```
# Unquoted strings are still strings
```

```
$ echo hello
```

```
hello
```

```
# Quoted strings are strings
```

```
$ echo 'hello'
```

```
hello
```

Quotes aren't optional with special characters

Unquoted special characters are shell expanded

```
$ echo *  
file1.txt folder
```

Quoted special characters aren't expanded

```
$ echo "*"
*
```

...unless the special character is a '\$'

```
$ echo "$my_variable"  
hello
```

...in which case we can use single quotes

```
$ echo '$my_variable'  
$my_variable
```

We don't *have* to quote; we can escape

We can use '\ ' to escape special characters

```
$ echo \*
```

```
*
```

Escaping and quoting compound on each other

(Sometimes this is what we want)

```
$ echo "\*"
```

```
\*
```

Understand the difference between globs and regex

```
grep homework_problem(.*) homework.py
```

```
#
```

```
^
```

```
# This is a special shell character!
```

```
# We need to quote or escape it.
```

```
grep 'homework_problem(.*)' homework.py
```

```
#
```

```
^
```

```
# It's quoted now--we're good.
```


Input & Output

`stdin`, `stdout`, `stderr`

- ▶ Each process...
 - ▶ can listen for text input on `stdin` (standard input)
 - ▶ can output "normal" text on `stdout` (standard output)
 - ▶ can output "error" text on `stderr` (standard error)

Redirection

- ▶ Normally `stdin` is the keyboard, and `stdout` & `stderr` are the terminal
- ▶ We can change this
 - ▶ We Have The Technology™

Syntax	Meaning
<code>command < file.txt</code>	<code>stdin</code> from <code>file.txt</code>
<code>command > file.txt</code>	<code>stdout</code> to <code>file.txt</code> (overwrite)
<code>command >> file.txt</code>	<code>stdout</code> to <code>file.txt</code> (append)
<code>command 2> file.txt</code>	<code>stderr</code> to <code>file.txt</code> (overwrite)
<code>command 2>> file.txt</code>	<code>stderr</code> to <code>file.txt</code> (append)

`/dev/null` is a “black hole” file

- ▶ Anything sent to `/dev/null` is thrown away
- ▶ Anything read from `/dev/null` is empty

Pipes send `stdout` of one command to `stdin` of another

Disclaimer: this is a toy example.

We normally run grep like this:

```
grep TODO *
```

But if we give grep no arguments, it

will search on stdin. So we can do

this equivalent command

```
cat * | grep TODO
```

Oneliners

Oneliners are chains of pipes

- ▶ We start with some sort of data
- ▶ Then we filter it down

Example

```
$ du -h d1 | sort -hr
```

```
# ^           ^
```

```
# |           |
```

```
# |           `– And feed it to this command (filter)
```

```
# `– Take the stdout of this command (initial data)
```

Useful commands

- ▶ Old:

- ▶ `sed`
- ▶ `grep`

- ▶ New:

- ▶ `find`
 - ▶ `-name`
 - ▶ `-regex`
- ▶ `curl`
- ▶ `xargs`

Examples

```
# Open all PDF files not named written.pdf
```

```
find . -name "*pdf" \  
  | grep -v "written.pdf" \  
  | xargs open
```

```
# Get 100 random lowercase dictionary words
```

```
shuf /usr/share/dict/words \  
  | head -n 100 \  
  | tr '[A-Z]' '[a-z]' \  
  | sort
```

```
# Count how many times it says "Vim" on a page
```

```
curl https://jez.io \  
  | grep --only-matching Vim \  
  | wc -l
```

Recap

Tips for Writing Oneliners

- ▶ Construct oneliners iteratively!
 - ▶ Try the first command, see what it outputs
 - ▶ Try the first two commands, see what they output
 - ▶ ...
- ▶ Many tools do the same thing
 - ▶ Choose what you're familiar with
- ▶ Some tools are subtly different
 - ▶ For example, not all commands have the same regex syntax

More resources

- ▶ Google is great for finding the *filtering* commands
 - ▶ “Strings that don’t match...”
 - ▶ “Sum a list of numbers”
 - ▶ “Replace character with newline”
- ▶ ... but don’t just run what people tell you!
 - ▶ `man` pages
 - ▶ <http://explainshell.com>