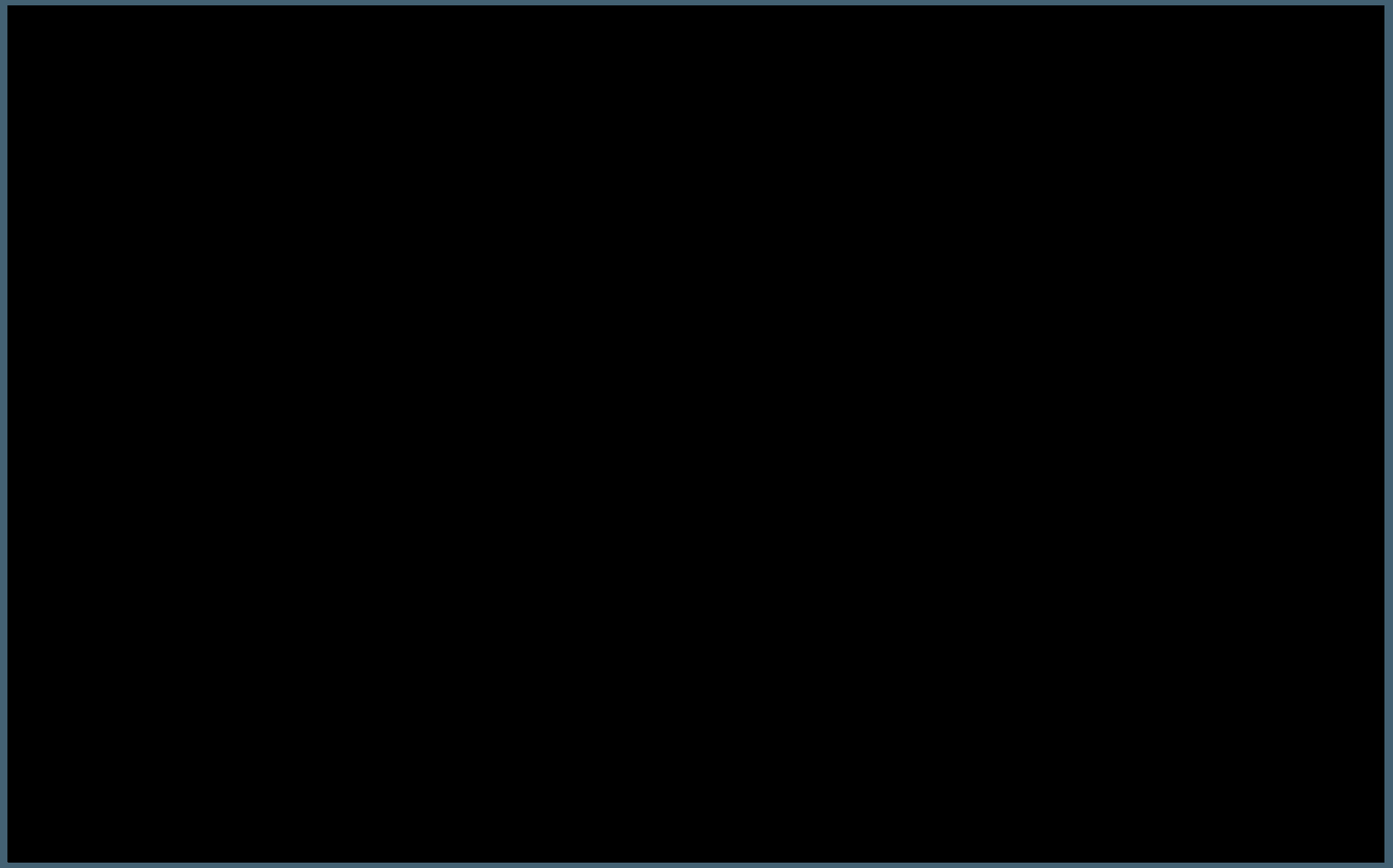
Bash Scripting: Do This, Not That.

Ryan Manly

@rmanly

commodore
207.
* THE NEW INTRO * 30 MIN. INT. *
* RAW SCROLLY! * SUUUUPEER VAUS *
* FANTASY * KRAKOUT C. SET *
commodore 64
CLR HOME
INST DEL
RESTORE
RETURN
SHIFT
CRSR
CTRL
SHIFT LOCK
BLK
WHT
RED
CYN
PUR
GRN
BLU
YEL



Why are we here?

Man accidentally 'deletes his entire company' with one line of bad code

‘I feel sorry to say that your company is now essentially dead,’ one person on a coding forum advised Marco Marsala

Andrew Griffin | @_andrew_griffin | Thursday 14 April 2016 | 327 comments

Agenda

PEMDAS!

Not really

Order of Expansions!

Word Splitting!

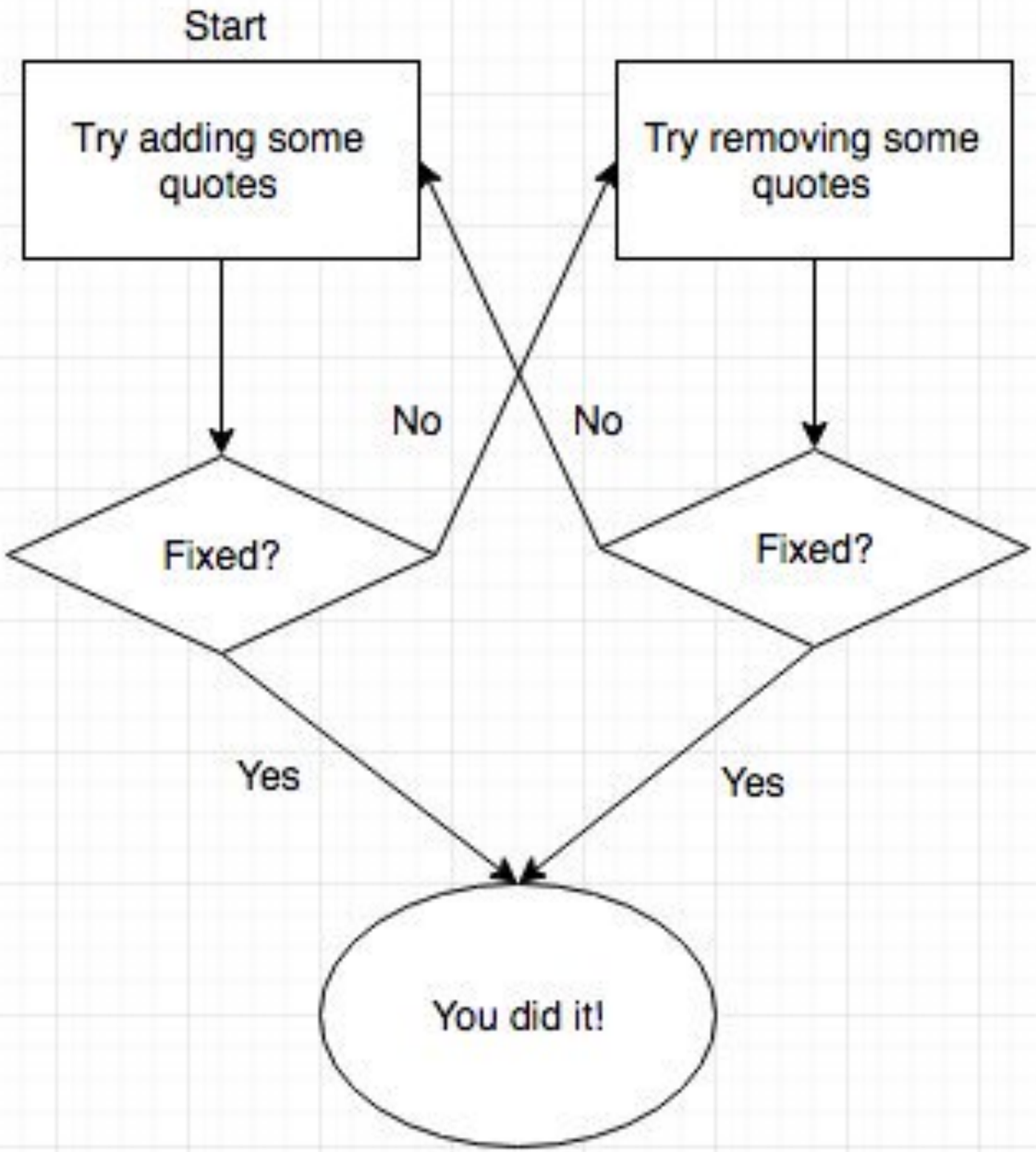
Quotes!

EVEN

MORE

QUOTES!

Beginner's Guide to Fixing Bash Syntax Errors



QUOTING RULE OF THUMB (newbie version):

If you are NEW to bash scripting…

1. **after** this talk go through the BashGuide
2. quote every expansion (use arrays when *for* breaks) 😉

QUOTING RULE OF THUMB (grizzled version):

If you are a wizened veteran of Bash…

1. go through the BashGuide
2. go through BashPitfalls

Order of Expansions

8 kinds of expansion

1. brace expansion
2. tilde expansion
3. parameter and variable expansion
4. command substitution
5. arithmetic expansion
6. word splitting
7. pathname expansion
8. process substitution

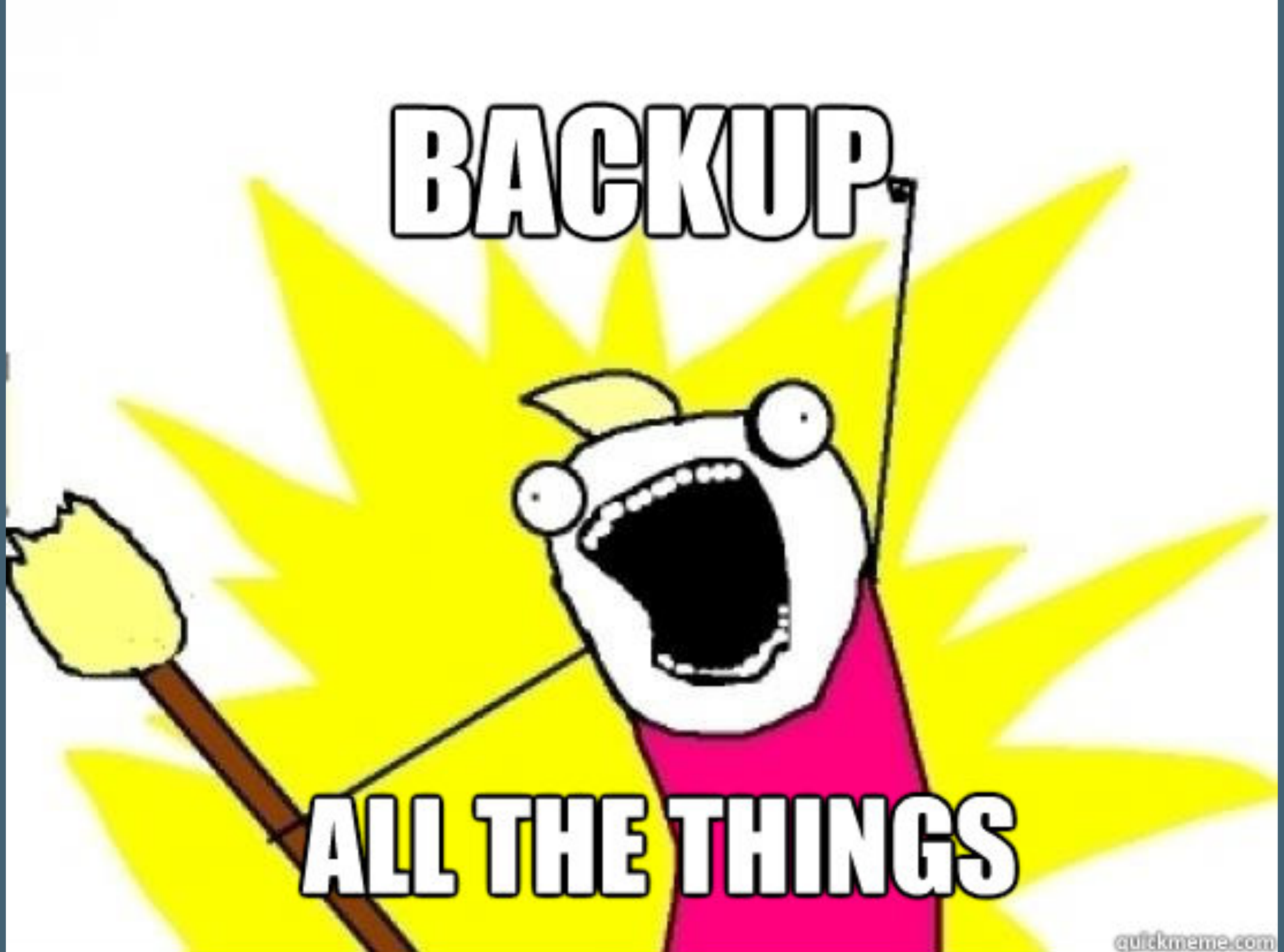
Expansion Order

1. brace expansion
2. tilde expansion
3. parameter, variable, arithmetic expansion, command and process substitution
4. word splitting
5. pathname expansion

BRACE EXPANSION

Brace Expand Backup files

```
$ ls
file.plist

$ cp file.plist{,.bak}

$ ls
file.plist     file.plist.bak

$ echo pre{,X}post
prepost preXpost
```


BACKUP
ALL THE THINGS
quickmeme.com

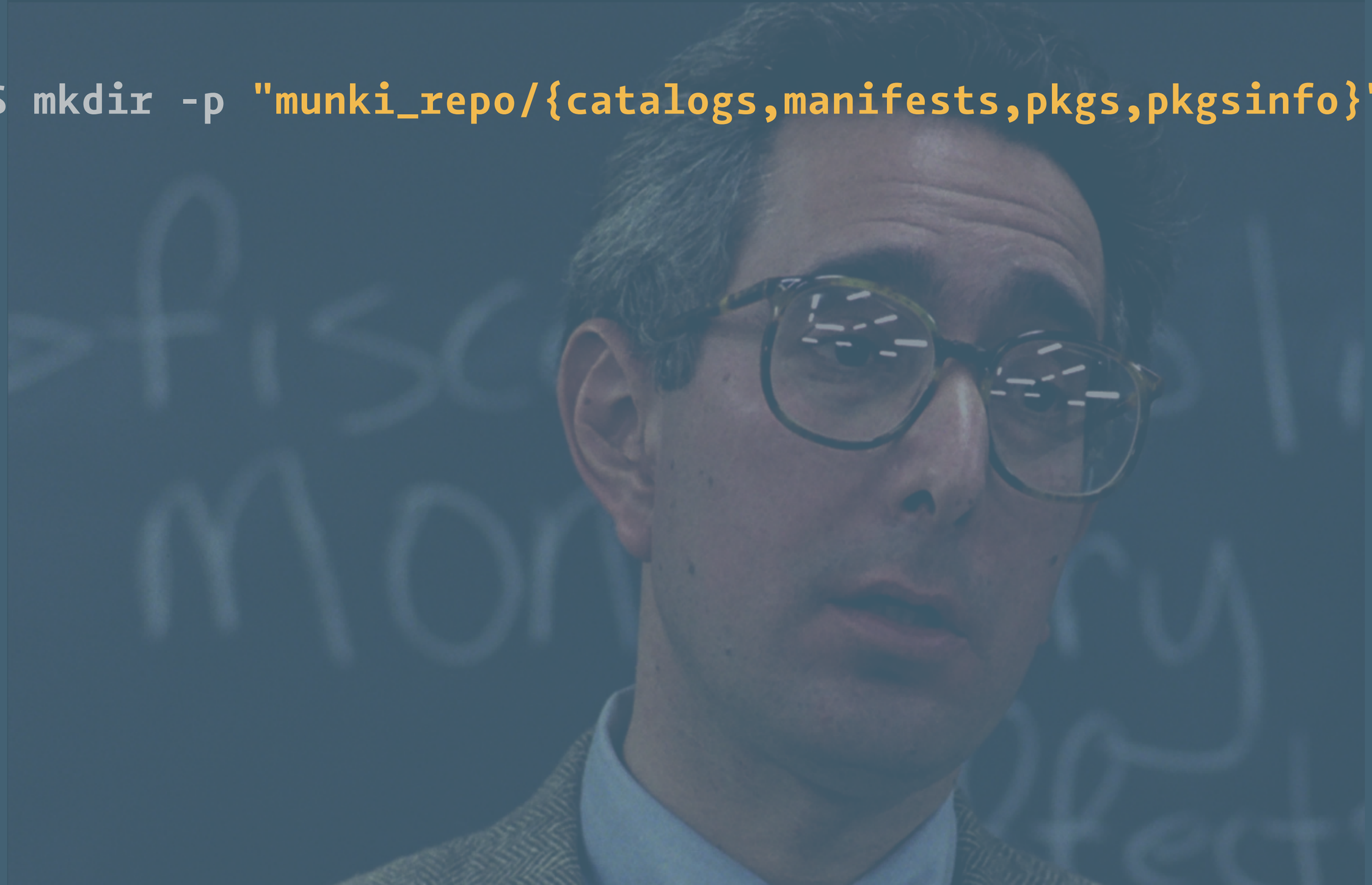
Brace Expand Directory Structure

```
$ mkdir -p munki_repo/{catalogs,manifests,pkgs,pkgsinfo}

$ ls
munki_repo

$ ls *
catalogs  manifests pkgs      pkgsinfo
```

QUOTING QUOTES

A correctly formed brace expansion must contain unquoted opening and closing braces, and at least one unquoted comma or a valid sequence expression.

```
$ mkdir -p "munki_repo/{catalogs,manifests,pkgs,pkgsinfo}"
```



```
$ ls munki_repo
{catalogs,manifests,pkgs,pkgsinfo}
```

TILDE EXPANSION

```
$ echo ~ladmin
/Users/ladmin

$ echo ~ryan
/Users/ryan

$ echo ~root
/var/root
```

```
$ echo ~+
/Users/ryan/test/brace

$ echo ~-
/Users/ryan/test/brace/munki_repo
```

QUOTING QUOTES

If a word begins with an unquoted tilde character, all of the characters preceding the first unquoted slash are considered a tilde-prefix. If none of the characters...are quoted, the characters...are treated as a possible login name...If the login name is invalid, or the tilde expansion fails, the word is unchanged.

» ~tilde-prefix/"something"

» ~root/"something"

» $ echo ~tilde-prefix/

» ~tilde-prefix/

```
$ foo=~/test/"file name-1.txt"; echo $foo
/Users/ryan/test/file name-1.txt

$ foo="~/test/file name-1.txt"; echo $foo
~/test/file name-1.txt
```

```
$ foo=~"/test/file name-1.txt"; echo $foo
~/test/file name-1.txt

$ foo=~/"test/file name-1.txt"; echo $foo
/Users/ryan/test/file name-1.txt
```

```
$ foo=~root/"test/file name-1.txt"/; echo $foo
/var/root/test/file name-1.txt/
```

PARAMETER EXPANSION

(and variables too)

${parameter}

```
$ echo $foo

$ echo "${foo:=bar}"
bar

$ echo $foo
bar
```

```
$ echo $foo

$ baz="nope"

$ echo "${foo:-$baz}"
nope

$ echo $foo
```

ARITHMETIC EXPANSION

```
total=0
for item in silly example here; do
    total=$(($total + 1))
done


$ echo $total
3
```

"MOVE ALONG."

COMMAND SUBSTITUTION

```
my_username=$(whoami)


$ echo $my_username
ryan
```

A few more words about Command Substitution

STOP

USING

BACKQUOTES!

Why should I stop using backquotes?

1. They are hard to read!

```
echo ``printf "%s\n" "this" "is" "some" "text"``
```

test

Ryan M to me

9:48 PM

echo \`printf "%s\n" "this" "is" "some" "text"\`

Reply

It works! | I got it! | Yes, it is!

2. Escaping nested backquotes is crazy town.

When the old-style backquote form of substitution is used, backslash retains its literal meaning except when followed by $, `, or \. **The first backquote not preceded by a backslash terminates the command substitution.**

When using the $(command) form, *all characters between the parentheses make up the command*; none are treated specially.

Command substitutions may be nested. To nest when using the backquoted form, **escape the inner backquotes with backslashes.**

```
$ ls -la
total 24
drwxr-xr-x  6 ryan  staff  204 Jun 26 22:35 .
drwx------+ 9 ryan  staff  306 Jun 26 22:28 ..
-rw-r--r--  1 ryan  staff    8 Jun 26 22:29 files-01.txt
-rw-r--r--  1 ryan  staff    8 Jun 26 22:35 files-02.txt
-rw-r--r--  1 ryan  staff    8 Jun 26 22:29 files-03.txt
-rw-r--r--  1 ryan  staff    0 Jun 26 22:28 files-04.txt
```

```
$ cat files-01.txt
10.11.5
```

```
$ cat files-02.txt
10.11.4
```

```
$ cat files-03.txt
10.11.5
```

```
$ cat files-04.txt
```

```
$ tar -czvf foo.tar.gz $(grep -F -l -R $(sw_vers -productVersion) .)
a ./files-01.txt
a ./files-03.txt
```

```
$ grep -F -l -R $(sw_vers -productVersion) .
++(:1): sw_vers -productVersion
+(:59): grep -F -l -R 10.11.5 .
./files-01.txt
./files-03.txt
```

```
$ grep -F -l -R "$(sw_vers -productVersion)" .
++(:1): sw_vers -productVersion
+(:60): grep -F -l -R 10.11.5 .
./files-01.txt
./files-03.txt
```

```
$ tar -czvf foo.tar.gz $(grep -F -l -R "$(sw_vers -productVersion)" .)
+++(:1): sw_vers -productVersion
++(:1): grep -F -l -R 10.11.5 .
+(:62): tar -czvf foo.tar.gz ./files-01.txt ./files-03.txt
a ./files-01.txt
a ./files-03.txt
```

```
$ tar -czvf foo.tar.gz "$(grep -F -l -R "$(sw_vers -productVersion)" .)"
+++(:1): sw_vers -productVersion
++(:1): grep -F -l -R 10.11.5 .
+(:61): tar -czvf foo.tar.gz './files-01.txt
./files-03.txt'
tar: ./files-01.txt
./files-03.txt: Cannot stat: No such file or directory
tar: Error exit delayed from previous errors.
```

QUOTING QUOTES

If the substitution appears within double quotes, word splitting and pathname expansion are not performed on the results.

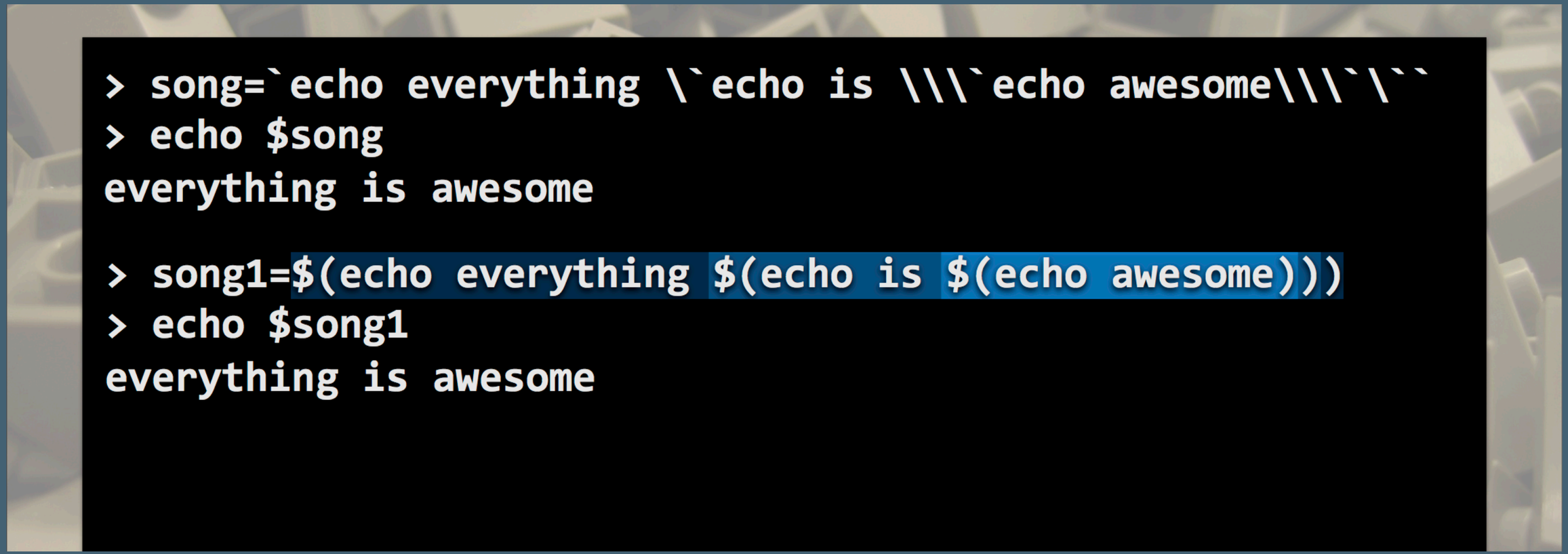
```
$ tar -czvf foo.tar.gz "$(grep -F -l -R "$(sw_vers -productVersion)" .)"
+++(:1): sw_vers -productVersion
++(:1): grep -F -l -R 10.11.5 .
+(:61): tar -czvf foo.tar.gz './files-01.txt
./files-03.txt'
tar: ./files-01.txt
./files-03.txt: Cannot stat: No such file or directory
tar: Error exit delayed from previous errors.
```

Back to our regularly scheduled program!

```
$ tar -czvf foo.tar.gz $(grep -F -l -R $(sw_vers -productVersion) .)
a ./files-01.txt
a ./files-03.t
```

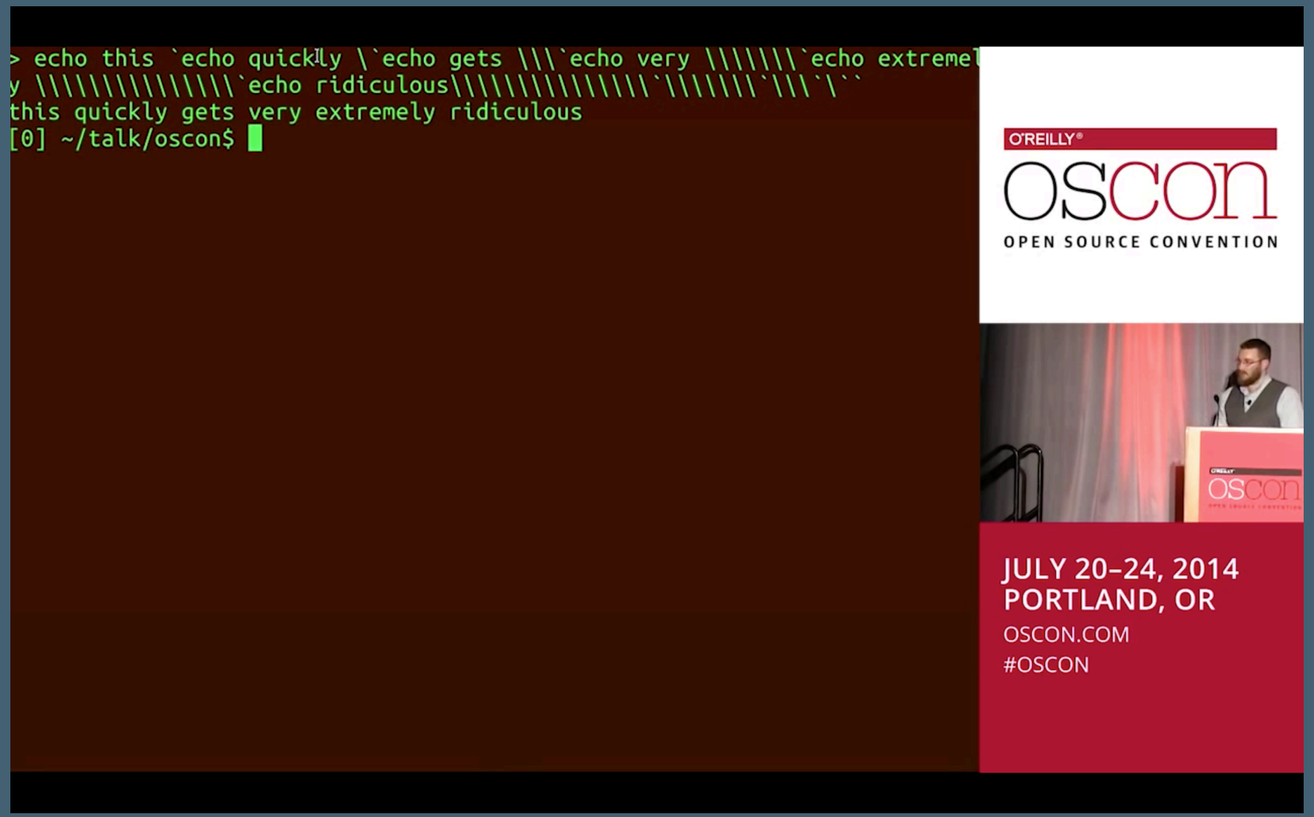
```
$ tar -czvf foo.tar.gz `grep -F -l -R \`sw_vers -productVersion\` .`
a ./files-01.txt
a ./files-03.txt
```

THAT'S NOT ENOUGH
WE HAVE TO GO DEEPER
quickmeme.co



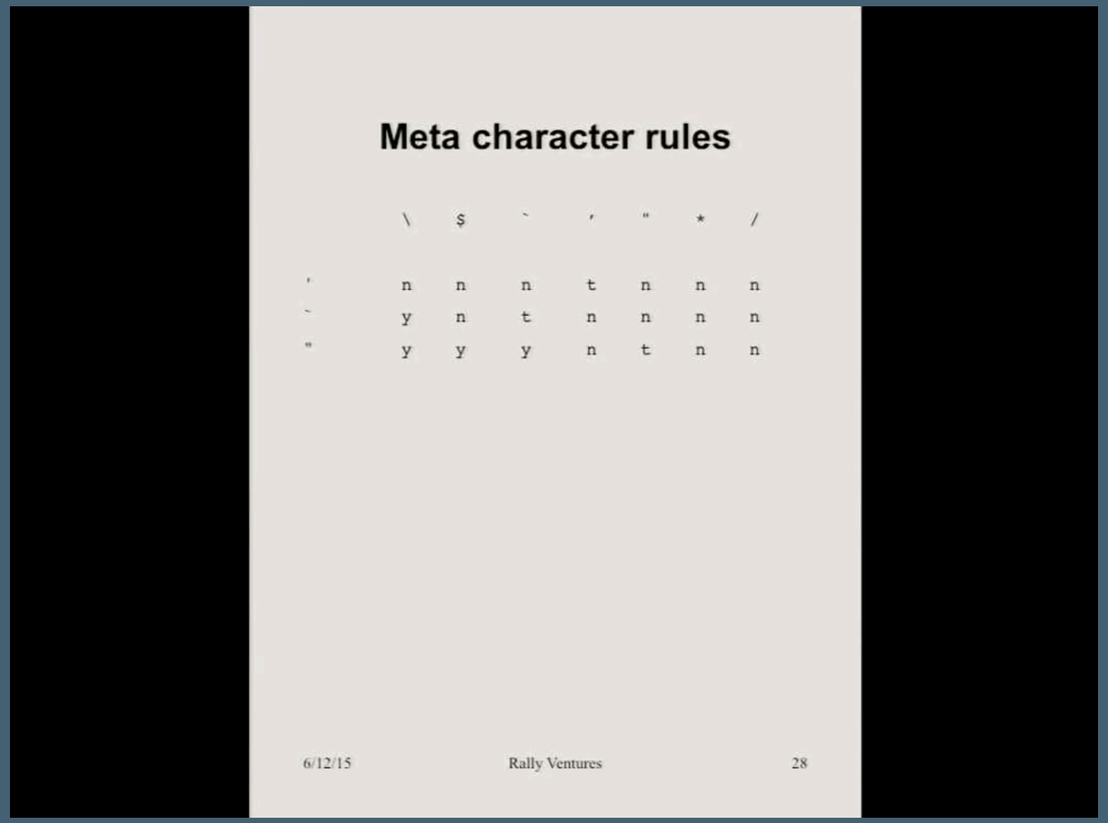
Bukowinski, Peter. "Shell Building Blocks" Macadmins Conference at Penn State. Pennsylvania State University, State College, PA. 10 June 2014. Presentation.

THAT'S NOT ENOUGH
WE HAVE TO GO DEEPER
quickmeme.co



Pannacciulli, James. "GNU Bash: An Introduction to Advanced Usage" OSCON 2014. Oregon Convention Center, Portland, OR. 23 August 2014. Presentation.

3. Some guy said $() is cleaner.



Bourne, Stephen R. "Early days of Unix and design of sh." BSDCan 2015. University of Ottawa, Ottawa, ON. 12 June 2015. Keynote Address.

PROCESS SUBSTITUTION

PROCESS SUBSTITUTION

#!/bin/bash

```
i=0

find $HOME/anaconda -name "*.plist" | while read line; do
    i=$(($i + 1))
done

echo "$i plists found"
```

```
i=0

while read -d $'\0' line; do
    i=$(($i + 1))
done < <(find $HOME/anaconda -name "*.plist" -print0)
```

Word Splitting

What is IFS?

» Input/Internal Field Seperator

» by default it is set to <space><tab><newline>

```
#!/bin/sh
printf "%d args:" $#
printf " <%s>" "$@"
echo
```

```
griffon:~$ var="This is a variable"
griffon:~$ args $var
4 args: <This> <is> <a> <variable>

griffon:~$ var="This is a variable"; args "$var"
1 args: <This is a variable>

griffon:~$ array=(testing, testing, "1 2 3"); args "${array[@]}"
3 args: <testing,> <testing,> <1 2 3>
```

QUOTING QUOTES

The shell scans the results of parameter expansion, command substitution, and arithmetic expansion that did not occur within double quotes for word splitting

```
while IFS=/ read foo;
    …
done
```

Pathname Expansion

a.k.a globbing

```
$ d_array=(D*)

$ declare -p d_array
declare -a d_array='([0]="Desktop" [1]="Documents" [2]="Downloads" [3]="Dropbox")'
```

```
shopt -s extglob
```

!(pattern-list)

Matches anything **except** one of the given patterns

```
$ pwd; ls
/Users
Guest  Shared ladmin ryan

$ shopt -s extglob

$ echo !(Shared|ladmin)
Guest ryan

$ shopt -u extglob
```

shopt -s failglob

```
set -euf -o pipefail
```

```
set -eu -o pipefail
shopt -s failglob
```

```
$ shopt -s failglob

$ py_files=(*.py)
-bash: no match: *.py

$ shopt -u failglob
```

shopt -s nullglob

```
$ py_files=(*.py)

$ declare -p py_files
declare -a py_files='([0]="*.py")'
```

```
$ py_files=()

$ shopt -s nullglob

$ py_files=(*.py)

$ declare -p py_files
declare -a py_files='()'
```

```
$ ls *.py
brace                glob                 spam                 test1
docker               null.bash            stream_monitor       venv
file name-1.txt proc_sub.bash    test.bash

$ shopt -u nullglob
```

We're done with expansions!

A *touch* of STYLE

Be Consistent

```
if true; then
    echo "yep"
fi

if true
then
    echo "also yep"
fi
```

```
if [[ -d /Library ]]; then
    echo "exists"
fi

if [ -e /System ]; then
    echo "also here"
fi
```

Asking software the right Questions

Biggest offender

grep | awk

```
grep foo | awk '{ print $1 }'
```

```
awk '/foo/{ print $1 }'
```

Awk One-Liners Explained

Sed One-Liners Explained

Next Biggest Offender

ps | grep | grep -v grep

```
ps axuw | grep -i chrome | grep -v grep
```

```
pgrep -i chrome
```

```
$ dscl . -read /Users/ryan | grep NFSHomeDirectory
NFSHomeDirectory: /Users/ryan

$ dscl . -read /Users/ryan NFSHomeDirectory
NFSHomeDirectory: /Users/ryan
```

```
$ sw_vers | awk '/ProductVersion/{ print $2 }'
10.11.5

$ sw_vers -productVersion
10.11.5
```

```
$ help while
while: while COMMANDS; do COMMANDS; done
    Execute commands as long as a test succeeds.

    Expand and execute COMMANDS as long as the final command in the
    `while' COMMANDS has an exit status of zero.

    Exit Status:
    Returns the status of the last command executed.
```

if and [[are not married

```
$ help -s if
if: if COMMANDS; then COMMANDS; [ elif COMMANDS; then COMMANDS; ]... [ else COMMANDS; ] fi
```

```
if grep -q 10.11.4 files-01.txt; then
    echo "yep"
else
    echo "nope"
fi
```

```
$ if grep -q 10.11.4 files-02.txt; then echo "yep"; else echo
"nope"; fi
yep
```

Questions?

Feedback

https://bit.ly/psumac2016-60