Bash Yellow-Belt Class

Improve Your Command over Your Computer

About You

New to Linux

OSX user, wondering what's so great about Terminal.app

Frustrated Windows cmd.exe user

Make a better impression:

- Pair-programming
- Screen-sharing

About Me

Eric Promislow @ericpromislow

Got my first Unix account in the 1980s

Found a AT&T “Programmers’ Workbench Guide” for free at a flea market

You could read the whole thing in a few hours. `man bash` returns almost as much text now

Overqualified

Underqualified

In 45 minutes, hopefully

The Basics

ls

more & less (export LESS=-EiXm)

cd

cat

man

du

Notation

Assume the prompt is ‘$’. Commands you type look like this, and are in red:

$ echo hello computer # Which star trek movie?

Computer responses are in blue:

hello computer

Pipes

```
$ man bash | wc -l
4890
$ i=$(!!)
$ echo  $((i / 54))
106
```

Directories: where files live

- and everything is a file
- including other directories

~/ - your home directory

~jane - Jane's home directory

$ cd # cd ~

$ pwd # echo current directory

$ echo $PWD # Same as pwd

Directories

$ pushd DIR # Like cd DIR, but…

$ popd # Goes back to top of stack

$ cd - # Go to previous dir

$ cd $OLDDIR # Same. What if no OLDDIR?

Matching names

```
* # Match all files
.* # Match files that start with a '.'
.??* # Match dot files, not . and .. (nor .x ...)
*.[td][xo][tc] # Match *.txt and *.doc (and *.toc)
```

Name expansion

These force expansions, even if no match

$ echo abc.{doc,txt} # abc.doc abc.txt

$ mv abc.{doc,txt} # mv abc.doc abc.txt

$ Sequence instructions: ${1..10}

1 2 3 4 5 6 7 8 9 10

History: More than pressing up-arrow

History commands: !

```
$ !! # Run previous command
$ echo abc
abc
$ !!
echo abc
abc
```

History commands: !

```
$ !! # Run previous command
$ !-2 # Run command before previous
$ echo abc
$ echo def
$ !-2
echo abc
abc
```

History commands: !

$!! # Run previous command

$!-2 # Run command before previous

$!95 # Run command #95

Where does that ‘95’ come from?

$ history

History commands: !

$!! # Run previous command

$!-2 # Run command before previous

$!95 # Run command #95

$ history | tail -10 # list last 10 commands

$!pu # Run last command starting with 'pu'

$!?abc # Run last command containing 'pu'

History: :p -- Are you sure?

```
$ echo abc
abc
$ !!:p
echo abc
$
```

History Arguments

!^ (“Bang caret”) # First word

!$ (“Bang dollar”) # Last argument

!* # All arguments

!!:3-5 # Select args 3 - 5 (start with 1)

!!:0 # The previous command only

History: picking apart paths

Say your last argument is a full path, like /home/eric/pictures/vacation.png

!$:h - head: /home/eric/pictures

!$:t - tail: vacation.png

!$:r - root: /home/eric/pictures/vacation

!$:e - extension: png

History: editing

```
$ ehco hello
ehco: command not found
$ ^hc^ch
echo abc
abc
$ ^bc
echo a
a
```

History: up-arrow

It’s there. And it’s actually using emacs bindings

$ set -o vi # Use vi-keys (‘ESC+k’ to move up)

$ set -o emacs # Back to default

Looping

```
$ for x in *.txt
> do
> commands...
> done
```

Looping example: fixing file names

```
for i in *.WAV ; do
  x=`echo $i | tr A-Z a-z`
  mv "$i" "$x"
done

# Try
  echo mv "$i" "$x"
```

Shell-scripting sidenote: if

```
if TEST ; then
   COMMANDS
elif TEST ; then
   COMMANDS
else
   COMMAND
fi
```

Shell-scripting sidenote: case

```
case $# in
  0) echo "no args" ;;
  1) echo "one arg: $1" ;;
  *) echo "Lots of args: $*" ;;
esac
```

Rolling your own: aliases

```
$ alias hi10="history | tail -10"
$ hi10
 1 echo abc
 2 echo abc def
...
```

Best when there are no arguments

Rolling your own: scripts

```
#!/bin/sh
echo -n 'What is your name? '
read x
echo "Hello, $x. Wouldn't you rather be" \
        "playing outside or reading a" \
        "good book than telling a computer" \
        "your name?"
```

Scripts: Variables

NAME=value

For some reason names are often all-caps

Make vars available to invoked programs:

export NAME=value

quotes matter

"... $NAME ..." => ... value ...

'... $NAME ...' => ... $NAME ...

Scripts: arguments

$1 ... first arg, etc.

$* : All arguments

"$@" : All args, doesn't get hung up on spaces

Always use "$@" unless you're sure no file has spaces.

$ shift # $1=$2, $2=$1, drop $<last>

Source and the environment

$ bash SCRIPT # doesn't change environment

$ ./SCRIPT # still doesn't

$. SCRIPT # does change it

VARIABLE=VALUE

alias ...

Shell Functions

Often better than aliases and shell-scripts

Different in /bin/bash and /bin/sh

```
function hello() {
  echo "Hello, $1"
}
$ hello Hal
Hello, Hal
```

Shell functions: learn more by example

https://github.com/ericpromislow/ddirs

```
$ pwd
/home/ericp/lab/rails/cronkite
$ d1
$ cd ~/svn/apps/kd
$ d2
$ cd1
/home/ericp/lab/rails/cronkite
```

More on shell scripts

#!/bin/sh

- first line
- chmod a+x
- make sure file is in PATH
- In current dir:

$ ./test.sh # Not 'test.sh' with no leading "./"

As promised

Where to go from here

$ man bash

Read existing shell scripts more comfortably

Kick ass during screenshares

Remember !!:p

Don't blame me if you delete all your files