SEMI-ADVANCED BASH

Richard Purves
(aka franton)
London Apple Admins - April 2019

START WITH A #!

Abbreviation	Name	Effect
`-B`	brace expansion	*Enable* brace expansion (default setting = *on*)
`+B`	brace expansion	*Disable* brace expansion
`-C`	noclobber	Prevent overwriting of files by redirection (may be overridden by >|)
`-D`	(none)	List double-quoted strings prefixed by $, but do not execute commands in script
`-a`	allexport	Export all defined variables
`-b`	notify	Notify when jobs running in background terminate (not of much use in a script)
`-c ...`	(none)	Read commands from **...**
`checkjobs`		Informs user of any open jobs upon shell exit. Introduced in version 4 of Bash, and still "experimental." *Usage:* shopt -s checkjobs (*Caution:* may hang!)
`-e`	errexit	Abort script at first error, when a command exits with non-zero status (except in until or while loops, if-tests, list constructs)
`-f`	noglob	Filename expansion (globbing) disabled
`globstar`	*globbing* star-match	Enables the ** globbing operator (version 4+ of Bash). *Usage:* shopt -s globstar
`-i`	interactive	Script runs in *interactive* mode
`-n`	noexec	Read commands in script, but do not execute them (syntax check)
`-o Option-Name`	(none)	Invoke the *Option-Name* option
`-o posix`	POSIX	Change the behavior of Bash, or invoked script, to conform to POSIX standard.
`-o pipefail`	pipe failure	Causes a pipeline to return the exit status of the last command in the pipe that returned a non-zero return value.
`-p`	privileged	Script runs as "suid" (caution!)
`-r`	restricted	Script runs in *restricted* mode (see Chapter 22).
`-s`	stdin	Read commands from `stdin`
`-t`	(none)	Exit after first command
`-u`	nounset	Attempt to use undefined variable outputs error message, and forces an exit
`-v`	verbose	Print each command to `stdout` before executing it
`-x`	xtrace	Similar to `-v`, but expands commands
`-`	(none)	End of options flag. All other arguments are positional parameters.
`--`	(none)	Unset positional parameters. If arguments given (`-- arg1 arg2`), positional parameters set to arguments.

START WITH A #!

- #!/bin/bash -e
- Causes the script to stop at first error
- #!/bin/bash -n
- Syntax Check. Processes but doesn't execute script
- #!/bin/bash -x
- Prints each command and result to STDOUT.

BUILT-IN VARIABLES

```
bash-3.2#
bash-3.2# ./secureboottest.sh &
[1] 23769
bash-3.2#
bash-3.2# pid=$!
bash-3.2#
bash-3.2# echo $pid
23769
bash-3.2#
```

- $!
- is the process ID of the last job run in the background
- Useful if you want to run something and kill it later
- For when you need something more surgical than “killall python”

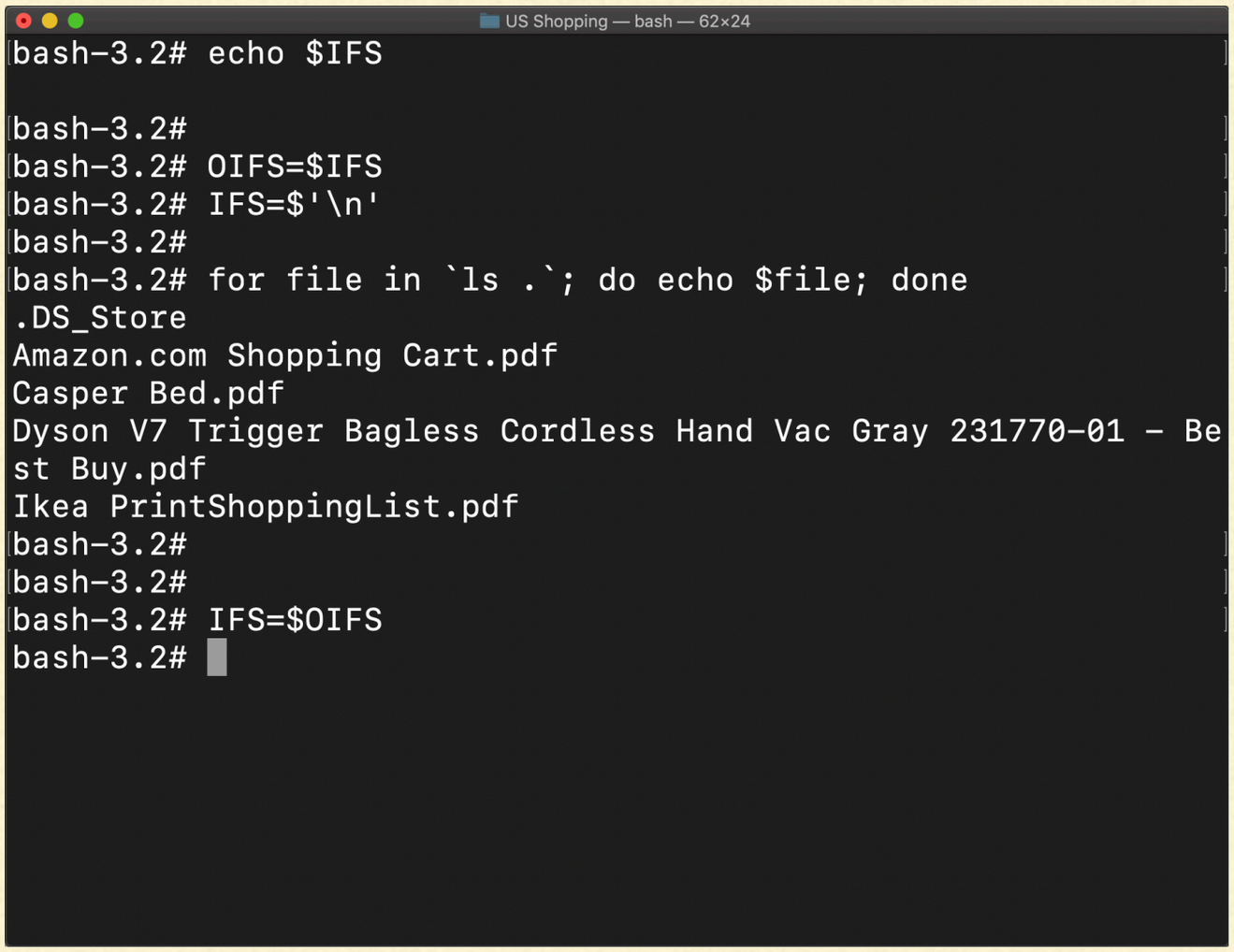
BUILT-IN VARIABLES

```
bash-3.2# ./secureboottest.sh
My PID is 24128

  PID TTY           TIME CMD
24128 ttys001    0:00.00 /bin/

<result>Full</result>
bash-3.2#
```

```
echo "My PID is $$"
echo ""
ps -ax $$
echo ""
```

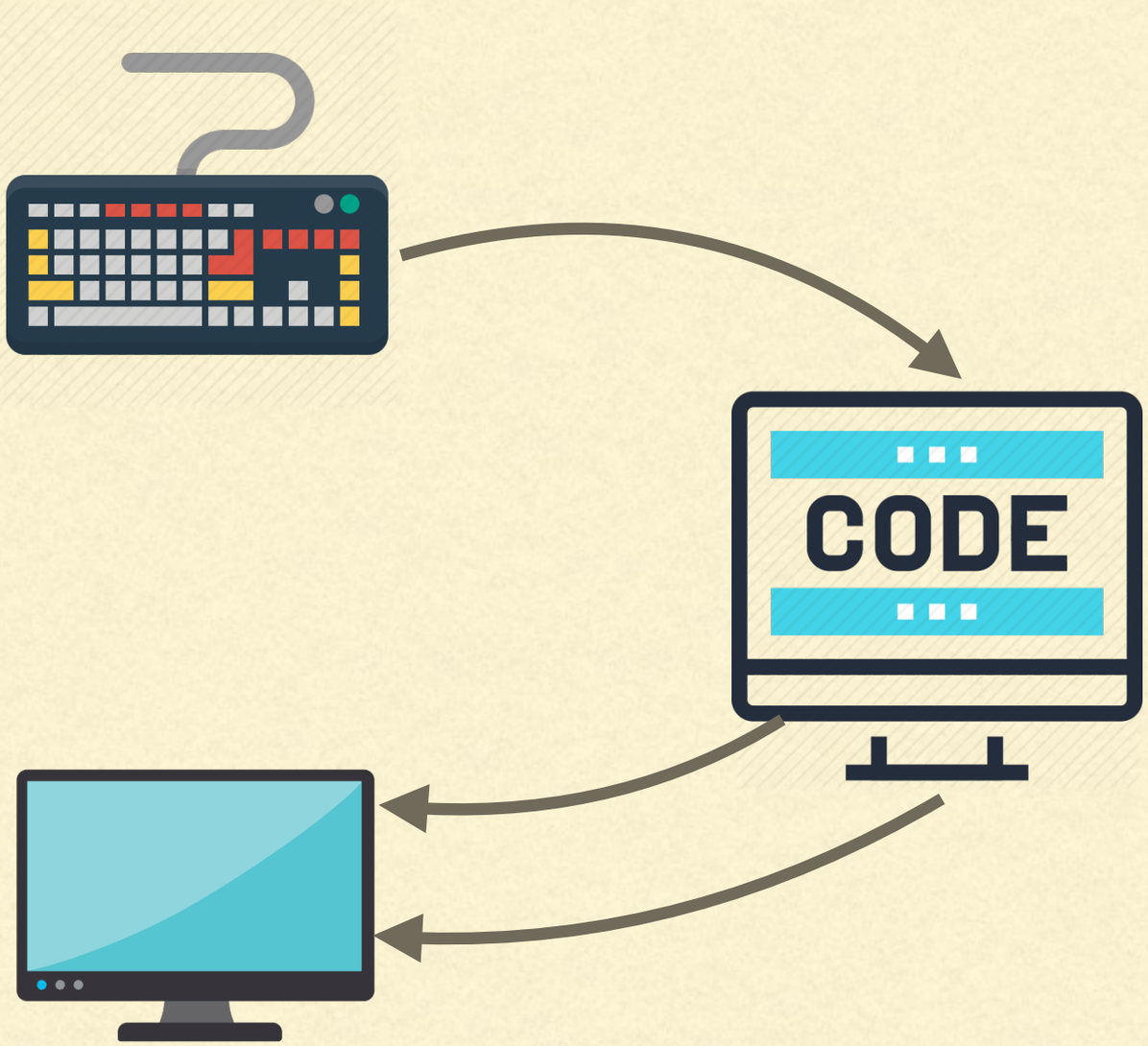
- $$
- is the process ID of the script itself
- Useful if you want to pass the PID details.
- e.g. startosinstall --pidtosignal $pidinfo
- For when you need something more surgical than “killall python”

BUILT-IN VARIABLES



- $IFS
- Internal Field Separator - defines what causes line splitting
- Default “<tab><space><newline>”
- So let’s change it to only split on newline. We can now process things with spaces in the name.
- For “killall python 2.7” kills the wrong one

I/O REDIRECTION



- Standard Streams are pre-connected input and output between a computer program and its environment.
- There are three Standard Streams
  - STDIN (aka 0)
  - STDOUT (aka 1)
  - STDERR (aka 2)
- And we can redirect where they go

I/O REDIRECTION

- Examples:
  - Capturing STDERR with STDOUT
  - Directing STDERR to nowhere
  - Creating your own named pipe!

I/O REDIRECTION

```
# Example pipe output from script to cocoaDialog

# We want to use file descriptor "20" so close any existing reference to it.
exec 20>&-

# Set up named pipe file to cocoaDialog so we can pass it info
mkfifo /var/tmp/progressbar
sleep 1

# Download macOS using the script and options we worked out.
./installinstallmacos.py --workdir="$workdir" --ignore-cache --compress "$args" &
pyscriptpid=$!

# Invoke cocoaDialog in background but direct the pipe input to it.
./cd progressbar --title 'Downloading macOS' \
                 --text 'Downloading macOS Installer: ' \
                 --width 500 \
                 --posY top < /var/tmp/progressbar &
cdpid=$!

# Wait half a second for cocoaDialog to initialise
sleep 1

# Use exec to avoid creating a BASH subshell while redirecting everything from
# file descriptor "20" to pipe file "progressbar"
exec 20<> /var/tmp/progressbar
```

I/O REDIRECTION

```
# Loop to show download progress via cocoaDialog
percent="0"

while [ "$percent" != "100" ];
do
    # How big is the download folder currently?
    percent=$( code to work out percent download here )

    # Give the user some hope. Might be zero for a while on slow connections.
    if [ "$percent" = "0" ]; then percent="1"; fi

    # Echo out percentage report to cocoaDialog via the pipe we set up earlier
    # Format: progress bar percentage, then text output of percentage completed
    echo "${percent} ${percent}% Completed" >&20

done

# Remove previous cocoaDialog window
kill $cdpid

# Delete the pipe to clean up.
rm -f /var/tmp/progressbar
```

SECURIT VERSY

- Who here has
- https://github.
- This is **obfus**
- And this is en

SECURITY AWARENESS

- Check $PATH and $HOME before you rely on them!
- $PATH might point to unexpected places.
- (you could hard code paths but that's annoying)
- $HOME becomes interesting if set to "/etc".

SECURITY

- Don’t pass credentials as script parameters! Shows up in ps -ax
- e.g ./admintask.sh username password
- I’m going to leave this here as an exercise, if you’re interested.
- Use named pipes as IPC to pass credentials back and forth?

PERSONAL ANNOUNCEMENT

- My current role will be my last …
- … in the UK at least
- (with certain exceptions)
- Blog post should be live about now