

Introduction to Bash

Ian Miell
@ianmiell



Introduction

- **About me**
- **Contact:**
 - Twitter: @ianmiell
 - Email: ian.miell@gmail.com

Bash and Me

- Used throughout career
- Never learned formally
- Stumbled around, lots of mistakes
- Slowly learned concepts and key points
- Wrote a book

This Course

- Live Walkthroughs
 - Encourage you to follow - ‘Hard Way’ Method
- Exercises
- Polls / yes/no ‘temperature checks’
- Group chat
- Materials:
 - <https://github.com/ianmiell/introduction-to-bash>

Pre-Requisites

- Familiar with command line
- Bash version 4+
 - `$ echo $SHELL`
 - `$ bash --version`
 - <4 is still ok
- Basic shell utilities (eg grep, cat, ls)
- Any editor (I use vim)

Target Audiences

- No knowledge assumed
 - Advanced questions outside the course please
- ‘Hardly/never used bash’
 - Coverage of 90% of bash features
- ‘Used bash casually for a while’
 - Refresher on some topics, learn some new things
- ‘Used bash for years, but never studied’
 - A-ha moments

Why This Course?

- Bash is everywhere
- Shells are everywhere
- Work with it every day
- Taken for granted that it's known
- Studying it pays massive dividends
 - Gateway to deeper OS concepts

Bash is under-served

- Man page is hard to follow if you don't know the jargon
- One-liners are easy to find but concepts give you real power
- Guides that assume knowledge you may not have

Ever been confused by...?

- Difference between '[' and '['
- Globbs vs regexes
- Single vs double quotes
- Difference between `` and \$()
- What a subshell is

Recently I've used bash to...

- Fix a Terraform script
- Robustly apply changes in a cloud-init VM script
- Automate the renaming of files with spaces in my backup folders
- Setup environments at work

Poll - Experience

- Never used bash
- Used bash for <2 years
- Used bash for >2 years
- Used bash for >5 years
- Studied bash seriously

Structure of Course

- Part I – Bash Basics
- Part II – Further Bash Basics
- Part III - Scripting
- Part IV - Advanced

Discussion

- What do you want to achieve in bash?
 - Any specific goals?
- What have you been frustrated by with bash?

Part I – Bash Basics

- 1.1 Bash background
- 1.2 Variables
- 1.3 Globbs
- 1.4 Pipes and Redirects

1.1 What is Bash?

- What is a shell?
- A program takes input from a terminal
- Translates input into:
 - System calls
 - Calls to other programs
 - Computation within the bash program
- Bash excels at ‘gluing’ other commands together

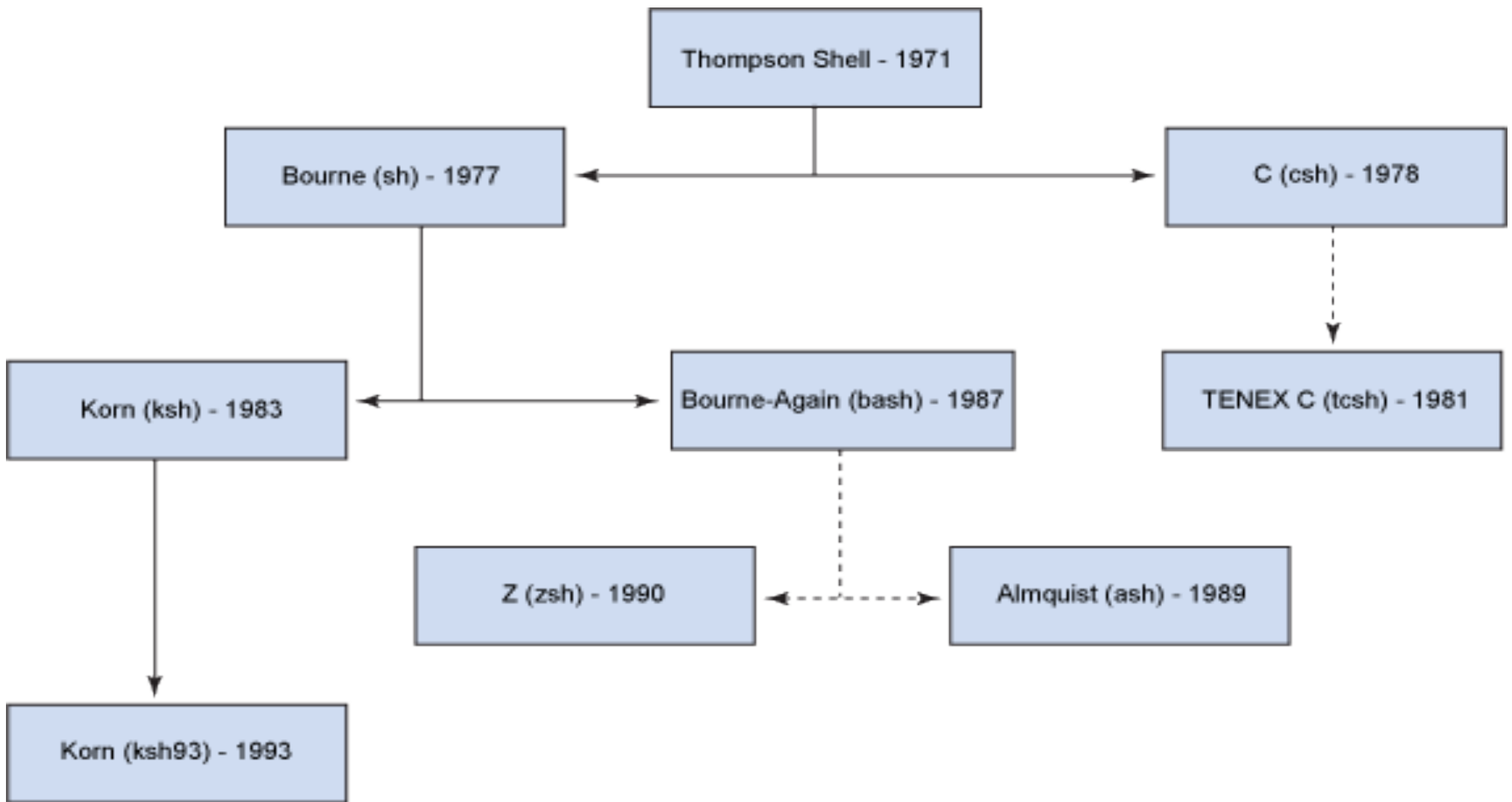
Other shells

- sh
- ash
- ksh
- tcsh
- tclsh

Walkthrough

- Run tcsh from bash

History of Shells



Bash in the Market

- Most popular shell
- Lots of competition:
 - zsh will be default on mac
 - fish is also popular
- Very rarely, you find servers that don't have bash on still

1.2 Variables

- Basic variables
- Quoting variables
- 'env' and 'export'
- Simple arrays

Walkthrough - Basic Variables

- Assignment
- De-referencing

Walkthrough – Variables and Quotes

- Why Quote?
- Space separation
- Double quotes and variables
- Single quotes and variables

Walkthrough – Shell Variables

- What are Shell Variables?
- Readonly variables
- Exporting variables
- Outputting exported and shell variables

Walkthrough - Arrays

- Zero-indexed
- Curly braces required
- All variables are arrays!
- declare -a

Recap - Variables

- `$` dereferences
- Variables in double quotes are interpreted, single quotes not
- Exported variables are passed to programs run within the shell
- `Env` shows exported variables, `compgen -v` shows all variables

1.3 Globbing

- What does '*' mean?
- Differences to regular expressions
 - Not familiar with regexes?
- Dotfiles

Activity – Basic globbing

- `ls *`
- Shell interpretation
- Globbing and quoting

Activity – Other Glob Characters

- Character classes
- Dots and dotfiles
- `.*` vs `*`
- `?`
- Special directories

Activity – Difference to Regexp

- Renaming a set of files
- ‘.’ is not the same as ‘*’!
- Extended globbing available in bash (not covered)

Recap - Globs

- What a glob is
- What a dotfile is
- Special directory files
- Globs, regexps and dots

Exercise I



1.4 Pipes and Redirects

- Basic redirects
- Basic pipes
- File descriptors
- Special files
- Standard out vs standard error

Walkthrough – Basic Pipes and Redirects

- The ‘>’ operator
- Simple pipe

Walkthrough – Standard out / error

- Errors and pipes
- ‘Channels’
- File descriptors
 - 0 – standard input
 - 1 – standard output
 - 2 – standard error

Walkthrough – Standard out / error

- ‘n>’ notation
- 2>&1
- Ordering is important
- Bash parses left to right

Walkthrough – Pipes vs Redirects

- Pipes ‘eat’ standard output from a command
- Pipes ‘output’ standard input to another command
- Redirects send a channel of output to a file
- The ‘<’ operator
- The ‘>>’ operator

Recap – Pipes vs Redirects

- The main 3 file descriptors
- ‘>’ vs ‘>>’
- *n*> and standard error
- 2>&1 and ordering

Part I Recap

- Globs
 - vs regexps
- Variables, arrays
- Pipes and redirects
- File descriptors

Exercise II



Break



Part II – Further Bash Basics

- 2.1 Command Substitution
- 2.2 Functions
- 2.3 Tests
- 2.4 Loops
- 2.5 Exit Codes

Discussion

- Is bash a programming language?
- What is a programming language?
- Why has bash lasted so long?

2.1 Command Substitution

- The '\$()' operator

Walkthrough – Command Substitution

- The '\$()' operator
- '\$()' vs ``
- Nesting

2.2 Functions in Bash

- Four types of command:
 - Function
 - Alias
 - Program
 - Builtin

Walkthrough - Basic Functions

- Declaring a function
- Function arguments
- Variable scope
- Local variables

Walkthrough - Builtins

- `cd` is a builtin
- Builtins can also be programs
- `'builtin'` is a builtin!
- Functions and builtins
- `unset -f`
- `declare -f / -F`

Walkthrough - Programs

- ‘which’

Walkthrough - Aliases

- 'alias'
- 'unalias'
- The 'type' builtin

2.3 Tests

- Bash tests
- Different ways of writing tests
- Logical operators
- Binary and unary operators
- 'if' statements

Walkthrough – Basic Tests

- Tests and exit codes
- Comparing values
- What is '['?

Walkthrough – Logical Operators

- **'!' means 'NOT'**
- **'||' means 'OR'**
- **'&&' means 'AND'**
- **'(...)' evaluates first**
- **'-a' vs '&&'**

Walkthrough – The ‘[[’ Operator

- Why do both exist?
- Confused?

Walkthrough – Unary and Binary Operators

- **'-z'**
- **'-a'**
- **'-d'**
- **Types in bash**

Walkthrough – if Statements

- Basic 'if' statements
- Bare 'if' statements

Recap - Tests

- Tests in bash
- '[' vs '['[
- Unary vs Binary operators
- Types in bash (limited)

2.4 Loops

- 'C'-style for loops
- 'for' loops over items 'in' lists
- 'while' and 'until' loops
- 'case' statements

Walkthrough – Loops

- C-style - ‘(‘
 - Different variable referencing
- ‘for f in \$()’
 - Beware!
- ‘while’ and ‘until’
- Infinite loop form with ‘break’

Walkthrough – Loops

- ‘case’ statements
- ‘esac’, ;; and *)
- Command line options

Recap – Loops

- Two types of for loop
- while loops
- case statements
- command line options

2.5 Exit Codes

- What an Exit Code is
- How to set one
- Conventions
- Other 'special' parameters

Walkthrough – Exit Codes

- The '\$?' variable
- '0' or 'not 0'
- Exit codes and tests

Standard Exit Codes

- 0 – OK
- 1 – General Error
- 2 – Misuse of shell builtin
- 126 – Cannot execute
- 127 – No file found matching command
- 128 – Invalid exit value
- (128 + n) – Process killed with signal 'n'
- (Signals covered in Part IV)

Walkthrough – Exit Codes

- Exit codes used differently by different apps
- eg grep
- The ‘exit’ builtin
- The ‘return’ builtin

Walkthrough – Special Parameters

- Special parameters == special variables
- \$?
- \$\$
- man bash

Recap – Exit Codes

- Standard exit codes
- Exit code usage (eg grep)
- Setting exit codes
- ‘return’ing from functions
- Special parameters

Exercise III



Discussion / Recap – Part II

- Bash more as programming language:
 - Functions
 - Tests / ifs
 - Loops
 - Return/Exit codes
 - Process and command substitution
- `$()` vs `` ``

Break



Part III - Scripting

- **Scripts and Startup**
- **The 'set' Command**
- **Debugging in bash**
- **Subshells**
- **IFS**

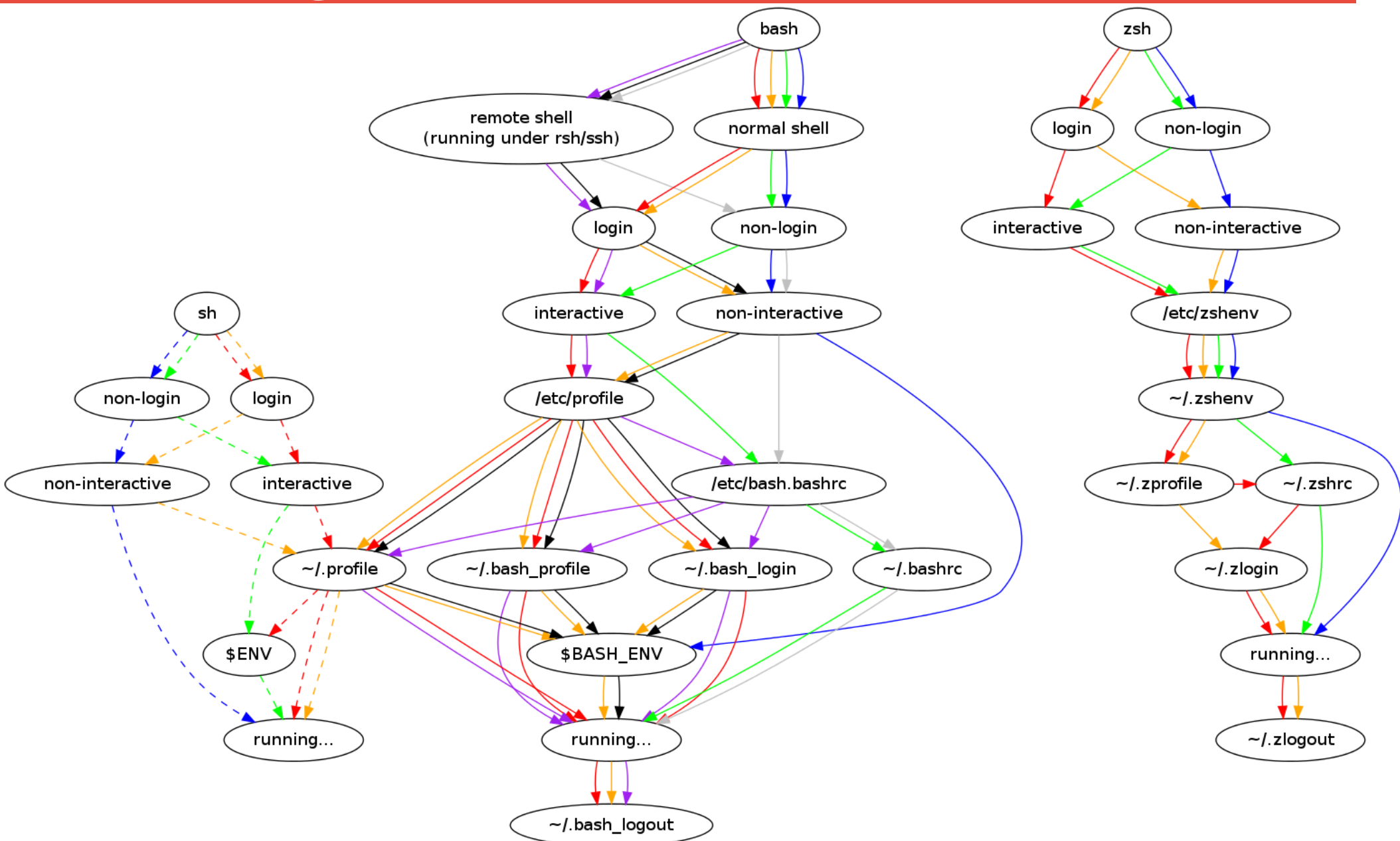
3.1 Scripts and Startup

- What shell scripts are
- What happens on bash startup
- This has cost me many hours!

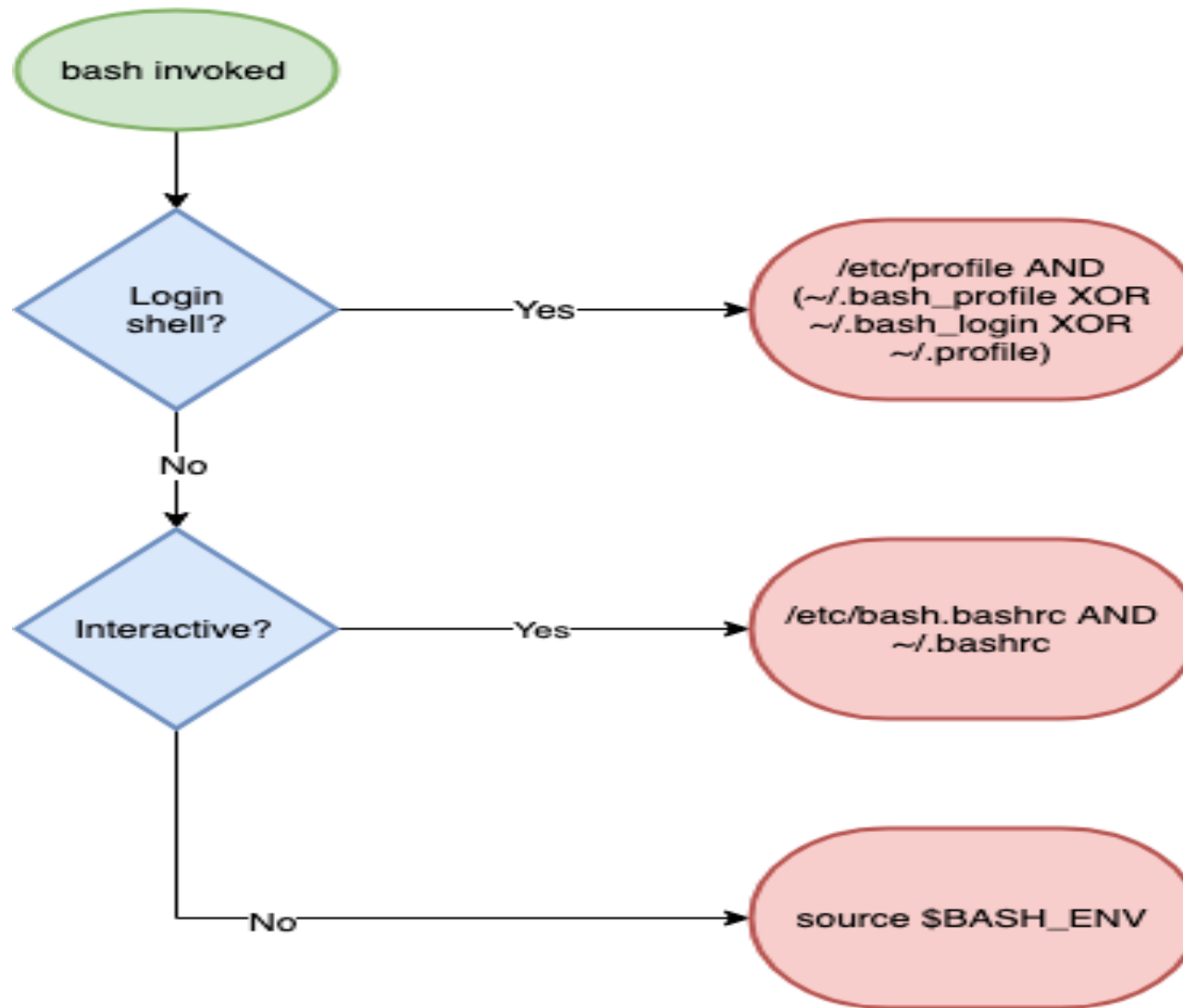
Walkthrough – Shell Startup

- The ‘shebang’ - ‘#!’
- What happens on bash startup
- Running an executable file
- Making a file executable

Walkthrough – Startup Explained



Walkthrough – Startup Explained (simpler)



Walkthrough - sourcing

- The 'source' builtin
- 'source' vs './'
- The 'sh' suffix
- Debugging with 'env -i'

Recap - Scripts and Startup

- What shell scripts are
- How complex bash startup can be
- Keep diagram handy!


3.2 The 'set' builtin

- Setting options in bash
- What POSIX is
- Most useful options:
 - nounset
 - xtrace
 - errexit
 - pipefail
- 'set' vs 'shopt'

Walkthrough – Running ‘set’

- ‘set’
- POSIX
- set POSIX off
- + is off – is on (this IS confusing)
- ‘set’ vs ‘env’

Walkthrough – ‘set’ Options

- **errexit**
- **xtrace**
- **nounset**
- **flags vs -o *name***
- **pipefail**
- **‘set’ vs ‘shopt’**
- 

Recap - 'set'

- Options: + off, - on
- POSIX
- Most common options
- shopt and set
-

Exercise IV



3.3 Subshells

- What is a subshell?
- How to create a subshell
- Why they are useful
- `()` vs `{}`

Walkthrough - subshells

- A simple subshell
- Subshells and variable scope
- Subshells and redirection
- `()` or `{}`?
- Subshells and working directory

3.4 Internal Field Separator

- aka IFS
- Why it's important
- How to use it

Walkthrough – Spaces in Filenames

- ‘for’ looping over files
- The IFS shell variable
- The \$” construct

Walkthrough – Spaces in Filenames

- Setting IFS
- The 'find' command and 'xargs'
- find, xargs and the null byte separator

Exercise V



Part III – Discussion / Recap

- Shell Startup
- Practical bash usage
 - Shell options
 - Shell debugging
 - IFS

Break



Part IV – Advanced Bash

- Traps
- String manipulation
- Autocomplete
- Walkthrough a ‘real’ script

4.1 Jobs and Traps

- Background jobs
- Traps and signals
- The 'kill' command
- The 'wait' builtin
- Trapping signals
- Process groups

4.1 Jobs Walkthrough



Standard Exit Codes - Refresher

- 0 – OK
- 1 – General Error
- 2 – Misuse of shell builtin
- 126 – Cannot execute
- 127 – No file found matching command
- 128 – Invalid exit value
- (128 + n) – Process killed with signal 'n'

4.2 Process Substitution

- The '`<()`' operator
- Substitution of file arguments

Walkthrough – Process Substitution

- The '`<()`' operator
- Substitution of file arguments

4.3 Debugging bash

- 'set' flags already covered
- Syntax checking
- Profiling bash and 'PS4'
- Shellcheck

Walkthrough – bash Arguments

- `bash -n`
- `bash -v`
- `bash -x`
- `bash -x` and `PS4`
- `shellcheck my bashrc`

Exercise VI



Wrapup

- Feedback welcome
- Thanks!
- Any questions:
 - @ianmiell
 - ian.miell@gmail.com

Wrapup

- <https://github.com/ianmiell/introduction-to-bash>
- @ianmiell
- ian.miell@gmail.com

Introduction to Bash

Ian Miell
@ianmiell

