The Shell

What is “the shell?”

- A program
- /bin/sh
- or maybe /bin/bash or /bin/csh or /bin/zsh
  - or some other executable
  - determined for each user in /etc/passwd
What is “the shell” not?

- *not* the operating system
  - merely one program that can run under the OS
- *not* necessary
  - machine doesn’t necessarily even run it

*Not the operating system*
Not necessary

Shell in context – bootup diagram

- Starring roles
  - kernel
  - init
  - rc
  - mingetty
  - login
  - shell

- Supporting roles
  - rc.sysinit
  - init
  - login
  - shell

operating system

shell runs late in the game if at all!

What does the shell do?

- Command processing
  - parse
  - expand
  - execute
- I/O redirection
- Piping
- Environment control
- Background processing
- Shell scripts
Command-line format

command [ - options ] [ arguments ]

ls -l m*
tokens

Command processing

- Reads what you type at the prompt
- Parses it (command/options/args token analysis)
- Optionally expands (transforms) it
- Responds by
  - locating/executing program file, or
  - emitting error message
Command-line expansion

- transformation of command line’s args
- performed by shell on each argument
- per special characters in arg (if any)
- altered result gets executed
- several transformation types

Types of transformation

- brace expansion
- tilde expansion
- parameter expansion
- variable expansion
- command substitution
- arithmetic expansion
- word splitting
- filename (pathname) expansion

– successively, in above order
Transformation triggers

- caused by special characters (metacharacters)
- `& ; | * ? ` " " [ ] ( ) $ < > { } ^ # / \ % ~`
- shell has extraordinary responses to them
- ordinary characters are nouns, embedded special characters are adverbs—“how to” treat the surrounding ordinary characters

Examples: various transformations

- brace expansion `{ }`
- tilde expansion `~`
- variable expansion `$` or `${ }`
- arithmetic expansion `$[ ]$
- filename expansion `* ? [ ]`
Expansion order matters

brace & variable expansion

brace expansion is already over

does not "work" because brace expansion is already over when variable expansion occurs

arithmetic & variable expansion

"works" because arithmetic expansion is yet to be performed when variable expansion occurs

Filename expansion metachars

- `*` any character(s) even none! \(\) echo `*.*`

- `?` any single character \(\) cat `lab?`

- `[…]` a single one of … \(\) `ls lab[a-m]`

see man `7 glob`
Examples: filename expansion

```
$ ls
  abc bcc cbc ebc hi report report2 reportk report.txt
  areport breport cbc ha ho report1 report2 reportkz
$ [root@BENCHMARK test]
$ [root@BENCHMARK test]
$ [root@BENCHMARK test]
$ [root@BENCHMARK test]
$ [root@BENCHMARK test]
$ [root@BENCHMARK test]
$ [root@BENCHMARK test]
$ $[root@BENCHMARK test]
$ [root@BENCHMARK test]
$ [root@BENCHMARK test]
$ [root@BENCHMARK test]
$ [root@BENCHMARK test]
$ [root@BENCHMARK test]
$ [root@BENCHMARK test]
$ [root@BENCHMARK test]
```

shell vs launched (child) process: command-line before vs after

bash, as parent process

- arguments
- environment
- signal table
- file descriptors

```
$ echo h?
```

```
$ echo h?
```

After

Child is expansion-oblivious, never sees the “?”

shell passes “ha ha ho” to echo as arg list (for example)

```
$ echo “ha ha ho”
```

```
$ echo “ha ha ho”
```

Child is expansion-oblivious, never sees the “?”
Suppressing expansion

- prevents metacharacters’ special interpretation
- single quotes
  - defeat all enclosed metacharacters
- backslash
  - defeats the metacharacter that follows it
- double quotes
  - defeat enclosed metacharacters except for variable ( $( ) ) and command ( $( ) ) expansion

Examples: expansion suppression

double quotes suppress filename expansion
but not variable or command expansion
single quotes do that though
backslash does it on single characters

Including backslashes
I/O redirection

- cat lab? >> assignments
- mail dmorgan1 < prepared-message
- ls –z 2> errorlog.txt
**echo “hello”**

- **current dir**
- **arguments**
- **environment (variables)**
- **signal table**
- **user**

- **code**
- **data**

**echo “hello” > file**

- **current dir**
- **arguments**
- **environment (variables)**
- **signal table**
- **user**

- **code**
- **data**

© David Morgan 2003-18
myecho.c  (an echo command do-alike)

```c
main (int argc, char *argv[]) {
    write(1, argv[1], strlen(argv[1]));
    write(1, "\n", 1);
}
```

- 

Piping

- `who | sort +4`
- `ls -l | grep ^d`

© David Morgan 2003-18
Environment control

- variables
  - environment vars (get passed to child processes)
  - local vars (don’t get passed)

- setting variables
  - declare DAY=Monday, or
  - DAY=Monday

- displaying variables
  - set or declare will display all variables
  - env or printenv will display environment vars only
  - echo $DAY to display value of single var

Shell scripts

- Files containing a list of commands
- Permissions set to executable
- May be run as programs
Information

- `$man bash` (man page for bash)
- *Unix Shells by Example*, Ellie Quigley