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.sysinit
rc
mingetty
login

Supporting roles
shell
cast of thousands
profile
bash_profile

What does the shell do?

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

Shell runs late in the game if at all!
**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

arithmetic & variable expansion

Filename expansion metachars

- * any character(s) even none! echo **
- ? any single character cat lab?
- [...] a single one of ... ls lab[a-m]
Examples: filename expansion

```
c ode  h? 
abc bcc cbc h1 report report2 reportk report.txt
ar eport breport cbc ha ho report1 report2 reportk2
 [root$DMACMD$ test]#
 [root$DMACMD$ test]#
 [root$DMACMD$ test]# [root$DMACMD$ test]# echo report'[7]
report2 reportk2 reportk2
 [root$DMACMD$ test]#
 [root$DMACMD$ test]#
 [root$DMACMD$ test]# [root$DMACMD$ test]#   'ha ha ho'
```
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`

I/O descriptors in a unix process

```
/home/david
0 = ls  1 = -l  2 = foo  3 = bar
UID=500  LINES=25  PATH=...
```

```
1 = default
2 = handler pointer
3 = ignore
```

```
0 = stdin
1 = stdout
2 = stderr
3 = /somefile
```

```
process ID #1234
  current dir
  arguments
  environment (variables)
  signal table
  file descriptors
  user
  code
  data
```

© David Morgan 2003-15
echo “hello”

current dir | code
arguments
environment (variables)
signal table

0: stdin
1: stdout
2: stderr
user

echo

"hello"

hello

code

echo

"hello"

hello

current dir | code
arguments
environment (variables)
signal table

0: stdin
1: stdout
2: stderr
user

> file

echo “hello” > file

current dir | code
arguments
environment (variables)
signal table

0: stdin
1: stdout
2: stderr
user

> file

© David Morgan 2003-15

© David Morgan 2003-15
Piping

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

```
$ echo -e "jan\nfeb\nmarch"
```

```
stdin
```

```
stdout
```

```
stdin
```

```
stdout
```

```
echo process
```

```
Jan
Feb
Mar
```

```
stdin
```

```
stdout
```

```
stdin
```

```
stdout
```

```
echo process
```

```
Jan
Feb
Mar
```

```
stdin
```

```
stdout
```

```
stdin
```

```
stdout
```
echo -e "jan\nfeb\nmar" | sort

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
Background processing

- `ls -l | lpr &`
- `sort bigfile &`

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