Chapter 5 – “The” shell

September 27, 2016

Understanding the parts of the command line

- Executing a command
- Standard input and output
- Redirection
- Pipes
- Background versus foreground
- Globs
- Built-ins

Simple execution

```
CMD [ARG] [ARG] ... [RETURN]
```

- Where do we find "commands"? Either built-in, absolute path, relative path, or via "$PATH"

Arguments and options

- We have a sequence with shell interpretation. The shell first parses the command, expanding “metacharacters” and variables
- Then it tries to execute the simple command (we will get to compound commands expressing iteration and alternation (like “while” and “if”) later)
- Then, if this is a simple binary, bash does a fork(2), then the child does an execve(2) of the new binary; the execve(2) sets up the child process’s stack with these arguments.
- It’s then up to the process to work through these arguments.

The common case

- The most common case for execution is
  - First, do a “getopt(3)” to parse out the options
  - Then work through any remaining tokens, treating them as arguments

The common case

- Usually a process receives three file descriptors: 0, 1, and 2, which are conventionally interpreted as standard input, standard output, and standard error.
- It is very common (though certainly not requisite) to use libc buffering over these file descriptors.

Editing the command line

- Very common these days; in the Unix world, generally the "readline" library is used for this (and its default editing bindings are those of emacs.)

Pipes

- You can use "|" to pipe stdout to stdin between processes.

Lists

- In shell-speak, a “list” is a sequence of commands and pipes separated by by these metacharacters:

```
; && ||
```

- Some shells (notably Bash) list the “background” operator & among these.

Globs

- As mentioned earlier, globs are created by “metacharacters”, somewhat resembling traditional regular expression syntax. The canonical implementation is in libc (see glob(3).)

```
* ? [ {
```

- There’s a whole concepts manpage glob(7) describing all of the details of glob syntax.

Useful globbing

```
$ touch file{1,2,3}
$ ls file*
file1  file2  file3
$ touch file1.backup file2.backup file3.backup
$ ls file*
$ ls !(*.backup)
```

Useful utilities and built-ins introduced in Chapter 5

- tr
- tee
- bg
- fg
- jobs

tr example

```
tr ' \t' '\n' < /etc/hosts
```

tee example

```
egrep local /etc/* 2>/dev/null | tee testfile
```