CS 241: Systems Programming
Lecture 3. More Shell

Fall 2019

Prof. Stephen Checkoway

Yesterday's in-class exercise

https://checkoway.net/teaching/cs241/2019-fall/exercises/Lecture-02.html

Grab a laptop and a partner and try to get as much of that done as you can in 20 minutes

Unix philosophy

As summarized by Peter H. Salus

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

Leads to many small utilities that we string together with the shell

Typical Unix tool behavior

```
$ program
```

- reads from stdin, writes to stdout

```
$ program file1 file2 file3
```

- runs ‘program’ on the 3 files, write to stdout

```
$ program —
```

- For programs that require filenames, might read from stdin

Standard input/output/error

Every running program has (by default) 3 open "files" referred to by their file descriptor number

Input comes from stdin (file descriptor 0)

- `input()` # Python: Read a line
- `System.in.read(var)` // Java: Read bytes and store in `var` array
- `$ IFS= read -r var` # Read a line and store in `var` variable

Standard input/output/error

Normal output goes to stdout (file descriptor 1)

- `print(var)` # Python
- `System.out.println(var)` // Java
- `$ echo "${var}"` # Bash

Error messages traditionally go to stderr (file descriptor 2)

- `print(var, file=sys.stderr)` # Python
- `System.err.println(var)` // Java
- `$ echo "${var}" >&2` # Bash

Redirection

`>file` — redirect standard output (stdout) to `file` with truncation

`>>file` — redirect stdout to `file`, but append

`<file` — redirect input (stdin) to come from `file`

`|` — connect stdout from left to stdin on right

- `$ ls | wc`

`2>file` — redirect standard error (stderr) to `file` with truncation

`2>&1` — redirect stderr to stdout

Redirection examples

```
$ echo 'Hi!' >output.txt
$ cat <input.txt
$ sort <input.txt >output.txt
$ ps -ax | grep bash
$ grep hello file | sort | uniq -c
$ echo Hello | cut -c 1-4 >>result.txt
$ ./process <input | tail -n 4 >output
```

(Almost) everything is a file

Files on the file system

Network sockets (for communicating with remote computers, e.g., web browsers, ssh, mail clients etc.)

Terminal I/O

A bunch of special files

- `/dev/null` — Writes are ignored, reads return end-of-file (EOF)
- `/dev/zero` — Writes are ignored, reads return arbitrarily many 0 bytes
- `/dev/urandom` — Reads return arbitrarily many (pseudo) random bytes

Given that `/dev/null` ignores all data written to it, how can we run the program `./foo` and redirect stderr so no error messages appear in our terminal?

A. `$ ./foo >/dev/null`

B. `$ ./foo 1>/dev/null`

C. `$ ./foo 2>/dev/null`

D. `$ ./foo | /dev/null`

E. `$ ./foo &2>/dev/null`

Some programs read all of their input before terminating. How can we run a program ./foo such that it has no input at all?

A. `$ ./foo </dev/null`

B. `$ ./foo </dev/zero`

C. `$ ./foo </dev/urandom`

D. `$ ./foo </dev/eof`

E. `$ echo | ./foo`