

# CS 241: Systems Programming

## Lecture 3. More Shell

Fall 2019

Prof. Stephen Checkoway

# 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`

# Bash simple command revisited

Recall we said a simple command has the form:

⟨command⟩ ⟨options⟩ ⟨arguments⟩

The truth is more complicated

- ▶ ⟨variable assignments⟩ ⟨words and redirections⟩ ⟨control operator⟩
- ▶ Variables and their assigned values are available to the command
- ▶ The first word is the command, the rest are arguments\*
- ▶ FOO=blah BAR=okay cmd aaa >out bbb 2>err ccc <in ;
- ▶ FOO=blah BAR=okay cmd aaa bbb ccc <in >out 2>err
- ▶ Real example: \$ IFS= read -r var

\* Bash doesn't distinguish between options and arguments, that's up to each command

# Permissions

Every user has an id (uid), a group id (gid) and belongs to a set of groups

Every file has an **owner**, a **group**, and a set of **permissions**

```
steve@clyde:~$ id
uid=1425750506(steve) gid=1425750506(steve) groups=1425750506(steve),1425700508(faculty)
steve@clyde:~$ ls -ld /home
drwxr-xr-x 4 root root 4096 Aug 13 2013 /home
steve@clyde:~$ ls -ld ~
drwxr-x--x 30 steve faculty 50 Sep 2 11:31 /usr/users/noquota/faculty/steve
steve@clyde:~$ ls -l hello.py
-rwx----- 1 steve steve 100 Aug 31 14:31 hello.py
```

First letter of permissions says what type of file it is: – is file, d is directory

# Permissions

The next 9 letters `rwxrwxrwx` control who has what type of access

- ▶ owner
- ▶ group
- ▶ other (everyone else)

Each group of 3 determines what access the corresponding people have

- ▶ Files
  - ▶ r — the owner/group/other can read the file
  - ▶ w — the owner/group/other can write the file
  - ▶ x — the owner/group/other can execute the file (run it as a program)
- ▶ Directories
  - ▶ r — the owner/group/other can see which files are in the directory
  - ▶ w — the owner/group/other can add/delete files in the directory
  - ▶ x — the owner/group/other can access files in the directory

# Permissions example

```
-rw-r--r-- 1 steve steve 0 Sep  3 14:25 foo
```

The owner (steve) can read and write foo, everyone else can read it

```
-rwx----- 1 steve steve 100 Aug 31 14:31 hello.py
```

The owner can read, write, or execute, everyone else can do nothing

```
drwxr-x--x 33 steve faculty 54 Sep  3 14:25 .
```

```
drwxrwxr-x 2 steve faculty 4 Sep  2 11:45 books/
```

steve and all faculty have full access to ./books, everyone else can see the directory contents

# Changing owner/group/perms

## Handy shell commands

- `chown` — Change owner (and group) of files/directories
- `chgrp` — Change group of files/directories
- `chmod` — Change permissions for files/directories

## Permissions are often specified in octal (base 8)

- 0 = ---      4 = r--
- 1 = --x      5 = r-x
- 2 = -w-      6 = rw-
- 3 = -wx      7 = rwx

Common values 777 (rwxrwxrwx), 755 (rwxr-xr-x) and 644 (rw-r--r--)



We can set a file's permissions by giving the numeric value of the permission (recall  $r = 4$ ,  $w = 2$ ,  $x = 1$ ) as an argument to `chmod`. Which command should we use to make a file, `foo`, readable and writable by the owner, readable by anyone in the file's group, and no permissions otherwise?

A. `$ chmod 644 foo`

B. `$ chmod 641 foo`

C. `$ chmod 640 foo`

D. `$ chmod 421 foo`

E. `$ chmod 046 foo`



# In-class exercise

<https://checkoway.net/teaching/cs241/2019-fall/exercises/Lecture-03.html>

Grab a laptop and a partner and try to get as much of that done as you can!