Bash: pipes and redirection

Glenn Bruns
CSUMB

What does this do?

```
$ ps -eF | awk '{print $1}' | sort | uniq > users.txt
```

Lecture Objectives

After this lecture, you should be able to:

- combine commands using redirection and pipes
- sequence commands
- write for loops

Output redirection >

Send **output** of a command to a file:

command > file

Redirection is part of bash – it's not a Linux command.

```
$ date > temp.txt
$
$ cat temp.txt
Tue Sep 29 12:41:35 PDT 2015
```

Appending output >>

```
$ who > temp.txt
$ cat temp.txt
brun1992 pts/0        2015-09-29 12:39 (10.11.84.204)
brun1992 pts/2        2015-09-29 10:33 (10.11.84.204)
$ date > temp.txt
$ cat temp.txt
Tue Sep 29 12:45:59 PDT 2015
$ who >> temp.txt
$ cat temp.txt
Tue Sep 29 12:45:59 PDT 2015
brun1992 pts/0        2015-09-29 12:39 (10.11.84.204)
brun1992 pts/2        2015-09-29 10:33 (10.11.84.204)
$
```

> overwrites file contents

>> appends

Input redirection <

```
$ ls -1 > temp.txt
$ cat temp.txt
complaints.csv
employees.txt
README.txt
salaries.csv
songs1.csv
temp.txt
$ sort < temp.txt
complaints.csv
employees.txt
README.txt
salaries.csv
songs1.csv
temp.txt
```

Linux file descriptors

Each running program has a table identifying open files.

A file descriptor is an index into this table.

Three standard file descriptors:

0 – standard input (stdin)

1 – standard output (stdout)

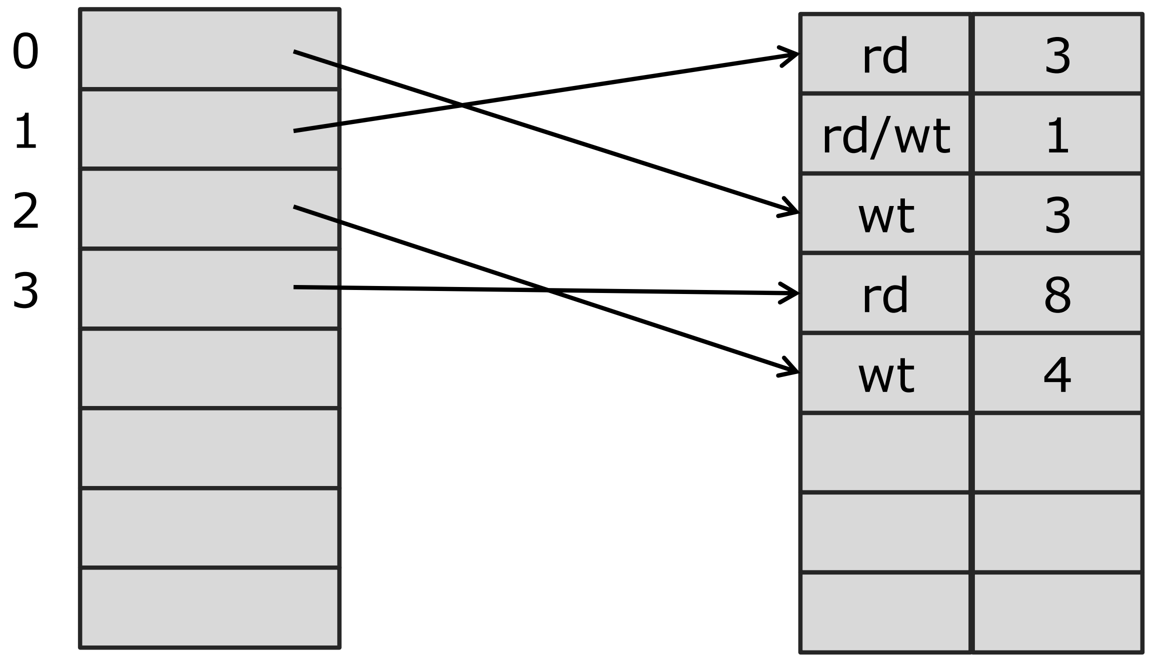
2 – standard error (stderr)

By default, stdout and stderr and sent to the terminal, stdin comes from the keyboard.

File descriptor tables

per-process
file descriptor table

system-wide
file table



file descriptor 3 is associated with a newly-opened file

Redirecting errors

```
$ ls
complaints.csv  employees.txt  README.txt  salaries.csv
songs1.csv  temp.txt
$ ls badfile > temp.txt
ls: cannot access badfile: No such file or directory
$ cat temp.txt
$
$ ls badfile 2> temp.txt
$ cat temp.txt
ls: cannot access badfile: No such file or directory
$
```

2> redirects stderr

Multiple redirection in one command

```
$ ls
complaints.csv  employees.txt  README.txt  salaries.csv
songs1.csv
$ head -2 employees.txt badfile > temp.txt 2> errs.txt
$ cat temp.txt
==> employees.txt <==
100  Thomas  Manager    Sales       $5000
200  Jason   Developer  Technology  $5500
$ cat errs.txt
head: cannot open `badfile' for reading: No such file or
directory
$
```

redirect stdout and stderr

Redirecting stdin, stderr to same file

```
$ ls
$ employees.txt  errs.txt  README.txt  salaries.csv
songs1.csv  songs.csv  temp.txt
$ head -2 employees.txt badfile > temp.txt 2>&1
$ cat temp.txt
==> employees.txt <==
100  Thomas  Manager    Sales       $5000
200  Jason   Developer  Technology  $5500
head: cannot open `badfile' for reading: No such file or
directory
```

redirect stderr to stdout

Pipe |

```
$ ls -1 | sort > temp.txt
$ cat temp.txt
complaints.csv
employees.txt
README.txt
salaries.csv
songs1.csv
temp.txt
```

What a pipe does **not** do:

```
$ ls -l > foo.txt
$ sort < foo.txt > temp.txt
```

Instead, the two processes are run at the same time.

Second command can start before first is finished.

Sequencing commands

A; B   run A and then B (regardless of success of A)

```
$ cp file1 file2; cp file1 file3; rm file1
```

A && B   run B if A succeeded

```
$ cp file1 file2 && cp file1 file3 && rm file1
```

Why would you use one or the other?

A || B   run B if A failed

For loops

```
$ for f in *.c
> do
> echo $f
> done
barrier-skeleton.c
fsbuf-skeleton.c
richer-barrier.c
rwlock1.c
```

```
for name in LIST
do
   COMMAND
done
```

Use can use ';' to put it on one line

```
$ for f in *.c; do echo $f; done
barrier-skeleton.c
fsbuf-skeleton.c
richer-barrier.c
rwlock1.c
```

For loops, example 2

```
$ for n in 1 2 3
> do
> head -$n rwlock1.c
> done


/*


/*
 * A pthreads readers/writers lock
$
```

On one line:

```
$ for n in 1 2 3; do head -$n rwlock1.c; done
```

Summary

We covered the basics of bash pipes and redirection:

`<`	redirect standard input
`>`	redirect standard output
`2>`	redirect stderr
`>>`	output redirection, append
`|`	pipe

We also looked at sequencing commands and for loops.