Pipes, Chains, and Redirection

Texas Linux Fest 2014
14 June 2014

Nathan Toups
rojoroboto

This session isn't about...

about this session

- an introduction to pipelining and Unix Philosophy
- exploration the history of pipelining in unix-like operating systems

about this session

- practical examples in the command line
- pipes, chains, and redirection in the command line like this:

```
find . -name '*.sh' | xargs wc -l > line_count.txt
```

- all examples built for Bourne Shell (bash)
- all examples are self contained
- all examples are focused on flow more than command features

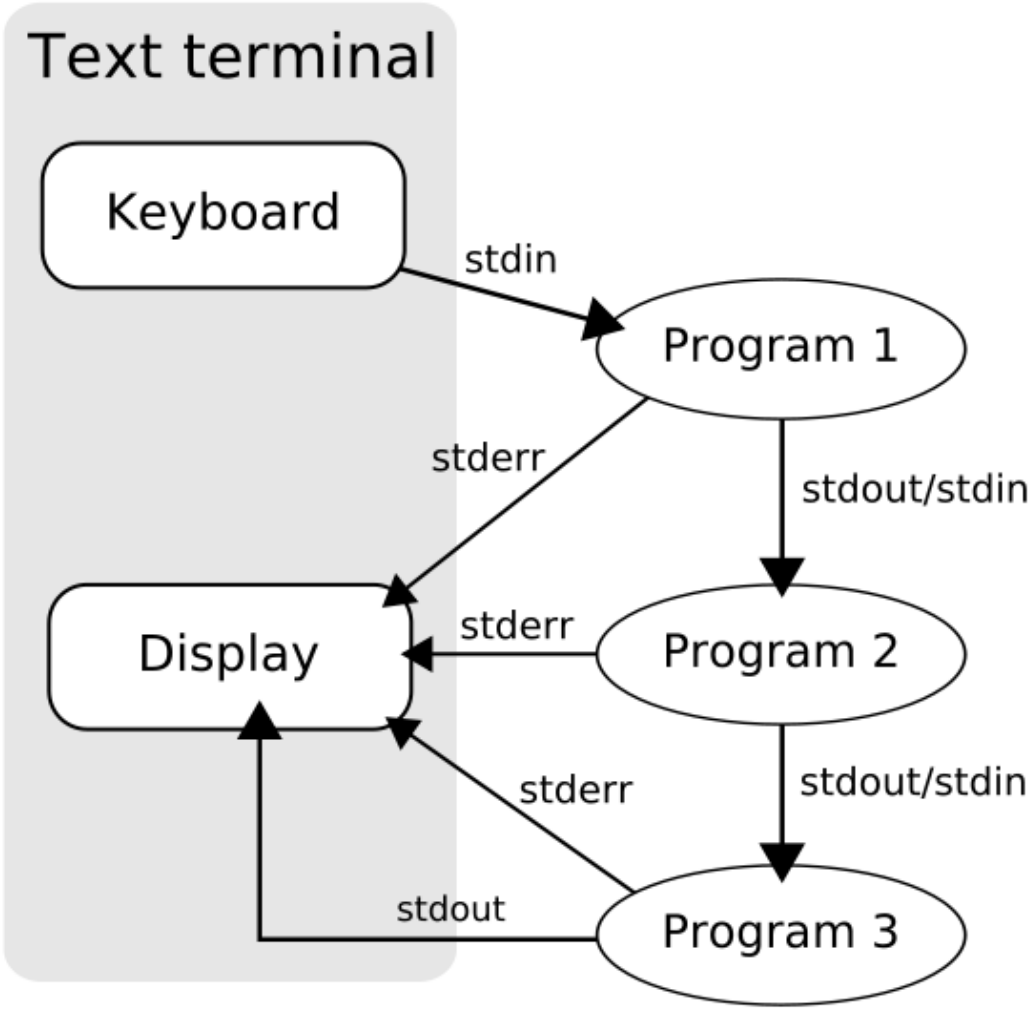
about this session

- this is an introduction to this topic, not an obfuscated code competition

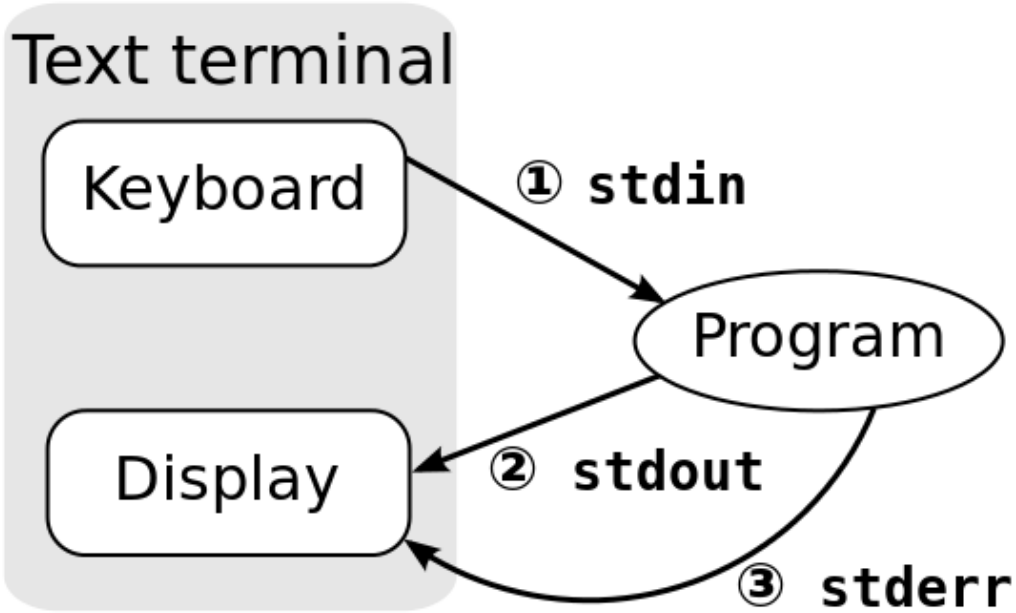
```
main( _, t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+,/+#n+,/#;\
#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l q#'+d'K#!/+k#;\
q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# ){nl]!/n{n#'; \
r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#\
\
n'wk nw' iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c ;;\
{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;\
#'rdq#w! nr'/ ') }+}{rl#'{n' ')# }'+}##(!!/")
```

- nor is it a session about advanced uses of awk, sed, find, xargs, etc

unix pipeline



unix redirection



the unix pipeline

- feed one program into another with pipes
- example from wikipedia:

```
% program1 | program2 | program3
```

- and with actual shell commands:

```
% ls -l    | grep key | less
```

A little history lesson in pipes

- concept of piping and vertical bar notation invented by Douglas McIlroy
- added to UNIX in 1973 by Ken Thompson
- wide range of influence, ported to DOS, OS/2, Windows, etc.

A little history lesson about Douglas McIlroy

- PhD in Applied Mathematics from MIT in 1959
- Bell Labs form 1958-1997
- head of Computing Techniques Research Department, the birthplace of Unix
- also wrote the unix tools diff, sort, and join, among others.
- Quote on Unix Philosophy:

```
This is the Unix philosophy:
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.
```

Unix Philosophy

Unix Philosophy

- Originated by Ken Thompson (Bell Labs, B, Go)
- Outlines a set of core principles that guide building tools in Unix
- Mike Gancarz (X Windows) outlines philosophy in 9 precepts:

```
1. Small is beautiful.
2. Make each program do one thing well.
3. Build a prototype as soon as possible.
4. Choose portability over efficiency.
5. Store data in flat text files.
6. Use software leverage to your advantage.
7. Use shell scripts to increase leverage and portability.
8. Avoid captive user interfaces.
9. Make every program a filter.
```

Pipes, Chains, and Redirection

Pipes, Chains, and Redirection

pipes

Pipes are used to feed the output of one program into another using the vertical bar notation (|).

chains

Chains are are used to string several commands together, a pipe is actually one of many chaining mechanisms. Other common mechanisms are the ampersand (&), semi-colon (;), AND (&&), OR (||), NOT (!).

redirection

Redirection, as it applies to the Bourne shell (bash), deals with manipulating standard I/O streams such as Standard Input (stdin), Standard Output (stdout), and Standard Error (stderr).

pipes

pipes

```
#!/bin/bash

## This command shows me a list of 3 names




cat list_of_names.txt
```

Run

pipes

```
#!/bin/bash

## This command shows me a list of 3 names
## piped into grep, which filters out only names with the letter "G"



cat list_of_names.txt | grep G
```

Run

pipes

```
#!/bin/bash

## This command shows me a list of 3 names
## piped into sort, to sort alphabetically



cat list_of_names.txt | sort
```

Run

pipes

```
#!/bin/bash

## This command shows me a list of 3 names
## piped into grep, which filters out only names with the letter "n"
## piped into sort, to sort alphabetically


cat list_of_names.txt | grep n | sort
```

Run

chains

chains

```
#!/bin/bash

#the simicolon allows you to run one command after the other




ls ; echo "above is a list of files in the directory"
```

Run

chains

```
#!/bin/bash

#the simicolon allows you to run one command after the other
#but it doesn't check that commands run successfully




ls oops ; echo "above is a list of files in the directory"
```

Run

chains

```
#!/bin/bash

# therefore, it is better to use the AND operator to chain commands
# that depend on one another.
#
# this command only runs echo IF ls runs successfully



ls oops && echo "above is a list of files in the directory"
```

Run

chains

```
#!/bin/bash

#ping local host, if successful, display "verified"




ping -c3 127.0.0.1 && echo "Verified"
```

Run

chains

```
#!/bin/bash

#ping local host, if successful, display "verified"
#if ping is not successful, echo "host down"




ping -c3 127.0.0.1 && echo "Verified" || echo "Host Down"
```

Run

chains

```
#!/bin/bash

#ping local host, if successful, display "verified"
#if ping is not successful, echo "host down"




ping -c3 kewldudez.biz && echo "Verified" || echo "Host Down"
```

Run

redirection

redirection

```
#!/bin/bash

# redirection allows you to change the default behavior of standard IO
# in this case, stdout is redirected to the file "sorted_list_of_names.txt"




cat list_of_names.txt | sort > sorted_list_of_names.txt
```

Run

redirection

```
#!/bin/bash

# redirection allows you to change the default behavior of standard IO
# in this case, stdin is redirected to sort and the data goes to stdout




sort < list_of_names.txt
```

Run

redirection

```
#!/bin/bash

# redirection allows you to change the default behavior of standard IO
# in this case, stdin is redirected to sort
# sort's stdout is redirected to the file sorted_list_of_names.txt
# this command is equivalent to:
# cat list_of_names.txt | sort > sorted_list_of_names.txt


sort < list_of_names.txt > sorted_list_of_names.txt
```

Run

redirection

```
#!/bin/bash

# redirection allows you to change the default behavior of standard IO
# in this case, ping is redirected to a file with the "append" redirection
# this adds the output of ping to the end of the file, instead of overwriting it




ping -c3 127.0.0.1 >> ping-success-tracker.txt
```

Run

redirection

```
#!/bin/bash

# redirection allows you to change the default behavior of standard IO
# in this case, ping is redirected to a file with the "append" redirection
# this adds the output of ping to the end of the file, instead of overwriting it
# notice that kewldudez is not logged to the ping-success-tracker
# and that output is displayed on the screen. This is stderr.


ping -c3 kewldudez.biz >> ping-success-tracker.txt
```

Run

redirection

```
#!/bin/bash

# redirection allows you to change the default behavior of standard IO
# in this case, ping is redirected to a file with the "append" redirection
# this adds the output of ping to the end of the file, instead of overwriting it
# notice that kewldudez is not logged to the ping-success-tracker
# and that output is displayed on the screen. This is stderr.


ping -c3 kewldudez.biz >> ping-success-tracker.txt 2>ping-failure-tracker.txt
```

Run

redirection

```
#!/bin/bash

# redirection allows you to change the default behavior of standard IO
# in this case, ping is redirected to a file with the "append" redirection
# and the 2>&1 folds the file descriptor for stderr into the stdout stream




ping -c3 kewldudez.biz >> ping-tracker.txt 2>&1
```

Run

redirection

```
#!/bin/bash

# uses find to search current directory and subdirectories for *.sh
# pipes the output into xargs which performs a line count in each file




find . -name "*.sh" | xargs wc -l
```

Run

redirection

```
#!/bin/bash

# uses find to search current directory and subdirectories for *.sh
# pipes the output into xargs which performs a line count in each file




find . -name "*.sh" | xargs wc -l | grep total | awk '{print $1}'
```

Run

redirection

```
#!/bin/bash

# echo spits out description and feeds to tr to remove the line break "\n"
# AND if successful it uses find to search current directory and subdirectories for *.sh
# pipes the output into xargs which performs a line count in each file
# which is filtered by grep for total and then piped into awk to pull the total number



echo "total lines of code:" | tr -d "\n" && find . -name "*.sh" | xargs wc -l | grep total | aw
k '{print $1}'
```

Run

Sources:

- Pipeline (Unix) (http://en.wikipedia.org/wiki/Pipeline_%28Unix%29)
- Unix Philosophy (http://en.wikipedia.org/wiki/Unix_philosophy)
- Douglas McIlroy (http://en.wikipedia.org/wiki/Douglas_McIlroy)
- Ken Thompson (http://en.wikipedia.org/wiki/Ken_Thompson)
- Chaining Operator Examples (http://www.tecmint.com/chaining-operators-in-linux-with-practical-examples/)

Thank you

Nathan Toups
rojoroboto
n@rjrbt.io (mailto:n@rjrbt.io)
@rojoroboto (http://twitter.com/rojoroboto)