Input/Output Redirection and Pipes

OPS102 Week 4 Class 2

Tiayyba Riaz/John Sellens

September 23, 2024

Seneca Polytechnic

Outline

The Unix Philosophy

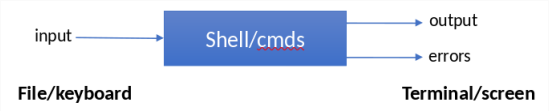
The Unix Philosophy

- The early developers of Unix established a “philosophy”, as a result of their brilliance and experience.
- In his book “A Quarter-Century of UNIX” (1994), Peter H. Salus summarized the Unix Philosophy as
  - 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.
- Many of the tools and commands we have seen so far reflect these ideas.
- Many Linux/Unix commands act as “filters” – they read input, process or modify it, and send it along as output.
- https://en.wikipedia.org/wiki/Unix_philosophy

Input/Output Redirection

Input/Output Redirection

- We give commands to the shell via the terminal and the shell executes them.
- But commands often need input and produce output.
- Usually input comes from files, and output goes to the terminal.
- Commands can also give errors if something goes wrong, these error messages are also, normally, printed on screen.



- Commands can also get their input from other commands or the terminal.
- Output and error messages can be sent (“redirected”) to files (or other commands) instead of displaying them on the terminal.

Standard Input, Output, and Error

- Normally, every program or command is invoked with three open file descriptors, which often default to being attached to the terminal:
  - 0 – the standard input
  - 1 – the standard output
  - 2 – the standard error output
- C programmers will recognize these as stdin, stdout, and stderr from `"stdio.h"`.
- The shell provides convenient ways to attach these file descriptors to files, devices, or other commands.

Input/Output Redirection and the Shell

- The shell uses special characters and syntax on command lines to implement I/O redirection.
- Use less than < before a filename to read input from that file.
- Use greater than > before a filename to write output to that file.
  - I remember > as it looks like an arrow pointing into a file.
  - If the file already exists, it is emptied before writing, otherwise it is created.
- Use two greater thans >> before a filename to append (add to the end of) output to that file.
- e.g. `"tr '[A-Z]' '[a-z]' <mixedcase.txt >lowercase.txt"`
- I/O redirection can be used with any command, though it's less useful with some commands e.g. `"date"` doesn't read input.

Redirecting Error Output

- Remember that there is also the standard error output on file descriptor 2.
- `"gcc -o myprog myprog.c 2>gccerrors.txt"`
- `"gcc -o myprog myprog.c 2>>allerrors.txt"`
- `"./myprog >myoutput.txt 2>&1"`
  - The `"2>&1"` means send file descriptor 2 output (errors) wherever file descriptor 1 output is currently going.

Input/Output and "/dev/null"

- Remember that the "/dev" directory contains special device files.
- "/dev/null" is a place you can read nothing from, or write anything to.
  - It's also called the "bitbucket", or the "black hole".
- "/dev/null" is handy if you don't want to give any input to a command, or you want to ignore any output (or errors).
- e.g. "grep Linux * 2>/dev/null" looks for "Linux" in all matching files, but throws away any error messages (say, if you don't have read permission on some files).

Connecting Commands with Pipes

Inter Process Communication: Pipes

- What if we want the put the output of a command to work as the input of another command?
- Commands can send their standard output directly to the standard input of other commands.
- Two or more simple commands can be combined to form a more powerful command sequence.
- No intermediate files need to be created.
- This is the magic of pipes!

How To Use Pipes

- We make a connection (pipe) between two commands by using the `"|"` pipe operator between commands.
- Many commands can be piped together especially Linux filter commands such as `"sort"`, `"cut"`, `"more"`, `"less"`, etc.
- The shell uses the pipe(2) system call to connect the necessary file descriptors before running commands.
- As usual, each command processes its input and generates its output.
- Commands must be chained in a specific order, depending on what you wish to accomplish (obviously).
- Example pipe use: `"ls -al | sort -nk 5"`
- Also works on Windows command prompt similarly to Linux: `"dir | sort"`
- https://en.wikipedia.org/wiki/Pipeline_(Unix)

Advantages of Using Pipes

- The most obvious advantage is convenience for the user.
  - It's a simple quick syntax, and very powerful and useful.
- It also means that intermediate output does not have to be written to and read from temporary files.
  - Disk input/output is one of the slowest parts of computing.
- With multi-processor machines, more than one command can actually be running at the same time.
  - Parallel, rather than serial, processing usually reduces the time taken to complete a task.

Summary

- The Unix Philosophy
- Standard Input, Output, and Error
- The shell lets you redirect I/O
- Pipes are fun!