Redirection and Pipes*

Boris Veytsman

July 26, 2001

*This document contains lecture notes for informal Unix seminar for ITT AES employees (Reston, VA). No information in this document is either endorsed by or attributable to ITT. This document contains no ITT Privileged/Proprietary Information.

Unix Filters

A plumbing system is very much like your electrical system, except that instead of electricity, it has water, and instead of wires, it has pipes, and instead of radios and waffle irons, it has faucets and toilets. So the truth is that your plumbing systems is nothing at all like your electrical system, which is good, because electricity can kill you. *Dave Barry, "The Taming of the Screw"*

A Unix system activity is mostly processing large streams of texts.

Examples:

- Send a stream of HTML-encoded text to a client (Web server)
- Take a log of HTTP server and find all clients from `.gov` computers (Web analysis)
- Substitute all instances of *Mr. Bean* to *Dr. Bean* in the text (text processing)

You have input and change it into output.

1. A text stream

2. Programs take it from *stdin*, do something and put it into *stdout (filters).*

| connection

< input valve

> **and** >> output valve

Suppose I want to know what does the user Steve Helfand do on my machine.

Step 1: Obtain the list of all processes:

```
boris@reston-0491:~$ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       256  0.0  0.0   1004     0 tty2     SW   Jul04   0:00 [getty]
root       257  0.0  0.0   1004     0 tty3     SW   Jul04   0:00 [getty]
root       258  0.0  0.0   1004     0 tty4     SW   Jul04   0:00 [getty]
root       259  0.0  0.0   1004     0 tty5     SW   Jul04   0:00 [getty]
root       260  0.0  0.0   1004     0 tty6     SW   Jul04   0:00 [getty]
root     30255  0.0  0.0   1004     0 tty1     SW   Jul05   0:00 [getty]
shelfand 27111  0.0  0.2   2020   736 pts/2    S    Jul13   0:01 -bash
shelfand 27113  0.0  0.0   2224     0 pts/2    TW   Jul13   0:00 [equal.pl]
...
```

Step 2: Take only the ones belonging to shelfand: *grep '^shelfand'*:

```
boris@reston-0491:~$ ps au | grep '^shelfand'
shelfand 27111  0.0  0.2  2020  736 pts/2    S    Jul13   0:01 -bash
shelfand 27113  0.0  0.0  2224    0 pts/2    TW   Jul13   0:00 [equal.pl]
shelfand 27114  0.0  0.0  3184    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 27324  0.0  0.0  3180    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 28009  0.0  0.0  2224    0 pts/2    TW   Jul13   0:00 [equal.pl]
shelfand 28404  0.0  0.0  3120    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 28408  0.0  0.0  3116    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 28410  0.0  0.0  2224    0 pts/2    TW   Jul13   0:00 [equal.pl]
shelfand 28699  0.0  0.0  3128    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 28720  0.0  0.0  3124    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 28723  0.0  0.0  2224    0 pts/2    TW   Jul13   0:00 [equal.pl]
shelfand 29832  0.0  0.0  3148    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 29920  0.0  0.0  3144    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 29922  0.0  0.0  2220    0 pts/2    TW   Jul13   0:00 [equal.pl]
...
```

Step 3: Print only the name of the process (field 11): *awk '{print $11}'*

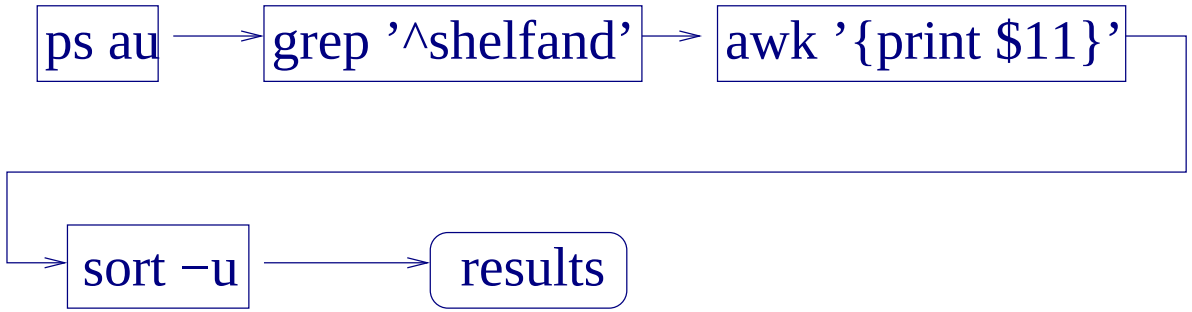
```
boris@reston-0491:~$ ps au | grep '^shelfand' | awk '{print $11}'
-bash
[equal.pl]
[trial.pl]
[trial.pl]
[equal.pl]
[trial.pl]
[trial.pl]
[equal.pl]
[trial.pl]
[trial.pl]
[equal.pl]
[trial.pl]
[trial.pl]
[equal.pl]
[trial.pl]
[trial.pl]
[trial.pl]
...
```

Step 4: Find the unique names: *sort -u*

```
boris@reston-0491:~$ ps au | grep '^shelfand' | awk '{print $11}' |sort -u
[a.out]
-bash
[equal.pl]
[trial.pl]
```

Step 5: Save everything in a file

```
boris@reston-0491:~$ ps au | grep '^shelfand' | awk '{print $11}' \
  |sort -u > results
```


ps au
grep ’^shelfand’
awk ’{print $11}’
sort −u
results

Argument List and Backtics

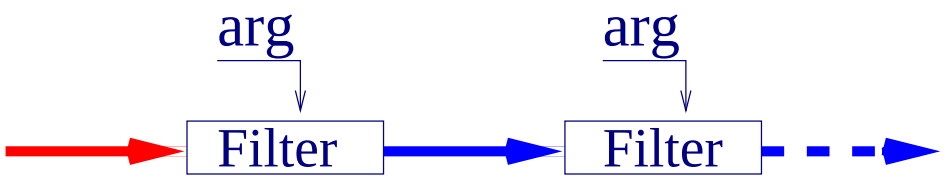
He draweth out the thread of his verbosity finer than the staple of his argument. *William Shakespeare, "Love's Labour's Lost"*

Two ways to supply information to the program:

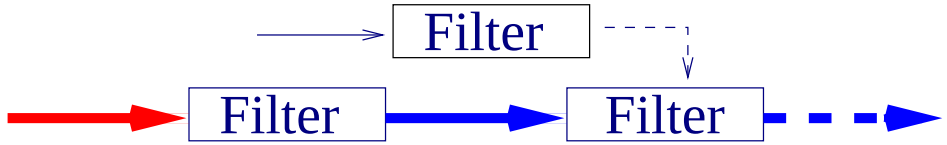
1. Through standard input: *ls *.c | more* shows the list of files screen by screen

2. Through argument list: *more *.c* shows each file contents screen by screen

Additional valve in the filters!



A creative use of argument list:



We can process the argument list through its own filters. The tool: backtics.

Suppose I want to kill all Steve Helfand's processes on my machine. I can use *kill number number. . .* , but how can I obtain the numbers?

Step 1: Get the list of processes:

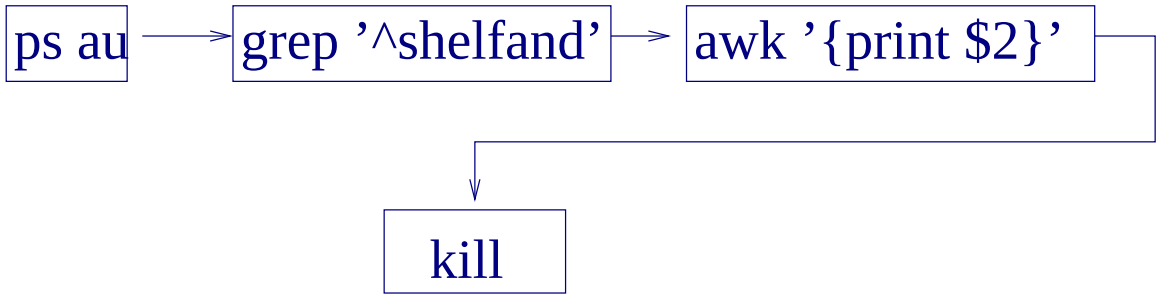
```
boris@reston-0491:~$ ps au | grep '^shelfand'
shelfand 27111  0.0  0.2  2020  736 pts/2    S    Jul13   0:01 -bash
shelfand 27113  0.0  0.0  2224    0 pts/2    TW   Jul13   0:00 [equal.pl]
shelfand 27114  0.0  0.0  3184    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 27324  0.0  0.0  3180    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 28009  0.0  0.0  2224    0 pts/2    TW   Jul13   0:00 [equal.pl]
shelfand 28404  0.0  0.0  3120    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 28408  0.0  0.0  3116    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 28410  0.0  0.0  2224    0 pts/2    TW   Jul13   0:00 [equal.pl]
shelfand 28699  0.0  0.0  3128    0 pts/2    TW   Jul13   0:00 [trial.pl]
shelfand 28720  0.0  0.0  3124    0 pts/2    TW   Jul13   0:00 [trial.pl]
...
```

Step 2: Get the numbers of processes: *awk '{print $2}'*:

```
boris@reston-0491:~$ ps au | grep '^shelfand' | awk '{print $2}'
27111
27113
27114
27324
28009
28404
28408
28410
28699
28720
28723
29832
29920
29922
29923
29964
...
```

Step 3: Send this to *kill*:

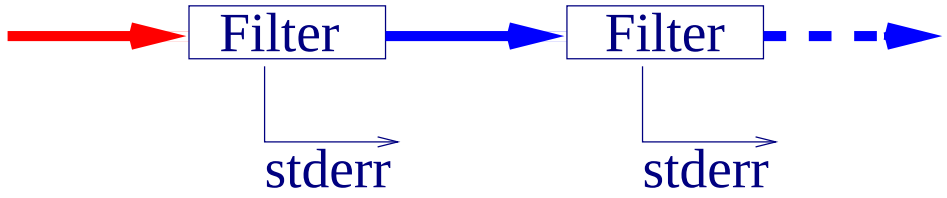
```
boris@reston-0491:~$ kill `ps au | grep '^shelfand' | awk '{print $2}'`
bash: kill: (27111) - Not owner
bash: kill: (27113) - Not owner
bash: kill: (27114) - Not owner
bash: kill: (27324) - Not owner
bash: kill: (28009) - Not owner
bash: kill: (28404) - Not owner
bash: kill: (28408) - Not owner
bash: kill: (28410) - Not owner
bash: kill: (28699) - Not owner
bash: kill: (28720) - Not owner
bash: kill: (28723) - Not owner
bash: kill: (29832) - Not owner
bash: kill: (29920) - Not owner
bash: kill: (29922) - Not owner
bash: kill: (29923) - Not owner
bash: kill: (29964) - Not owner
...
```


ps au
grep ’^shelfand’
awk ’{print $2}’
kill

Standard Error

A man of genius makes no mistakes. His errors are volitional and are the portals of discovery. *James Joyce, "Ulysses"*

Besides standard output, a Unix program usually has a standard error. Another output valve:



Example:

```
kill ‘ps au | grep ’^shelfand’ | awk ’{print $2}’‘ > results
```

The file *results* is *empty*

You can use >&:

```
boris@reston-0491:~$ kill ‘ps au | grep ’^shelfand’ | \
  awk ’{print $2}’‘ >& results
boris@reston-0491:~$ cat results
bash: kill: (27111) - Not owner
bash: kill: (27113) - Not owner
...
```

Advanced Redirection in */bin/sh*

> Any sufficiently advanced technology is indistinguishable from magic. *Arthur C. Clarke*

Unfortunately */bin/csh* has nothing beyond *>&* (see `http://www.perl.com/pub/language/versus/csh.html`). Use */bin/sh* and derivatives.

File descriptors: streams associated with I/O (like in C, Fortran, Perl. . .)

Standard descriptors:

- **0:** standard input
- **1:** standard output
- **2:** standard error

User-made descriptors: 3, 4,. . .

Redirection of descriptors: *m*>&*n* means "send *m* to the place *n* is going"

Sending error to file and processing input.

```
calculate 2>error.log | analyze
```

Sending output to file and processing error. Here we must be careful. The obvious solution is **wrong:**

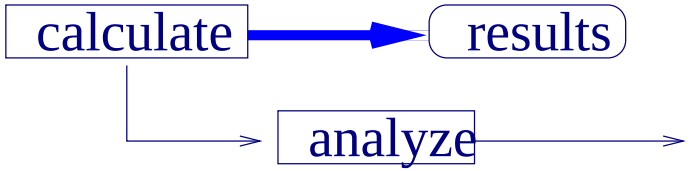
```
calculate 1>results 2>&1 | analyze
```

1. Standard output goes to the file *results*
2. Standard error goes to the *same* place, i.e. to the file *results*

Right solution:

```
calculate 2>&1 1>results  | analyze
```

1. Standard error goes to where standard output is going
2. Standard output goes to the file *results*, leaving standard error untouched



Switching error and output: Again the obvious solution is **wrong:**

```
calculate 2>&1 1>&2
```

What does it do?

To switch two variable you need the *third* one!

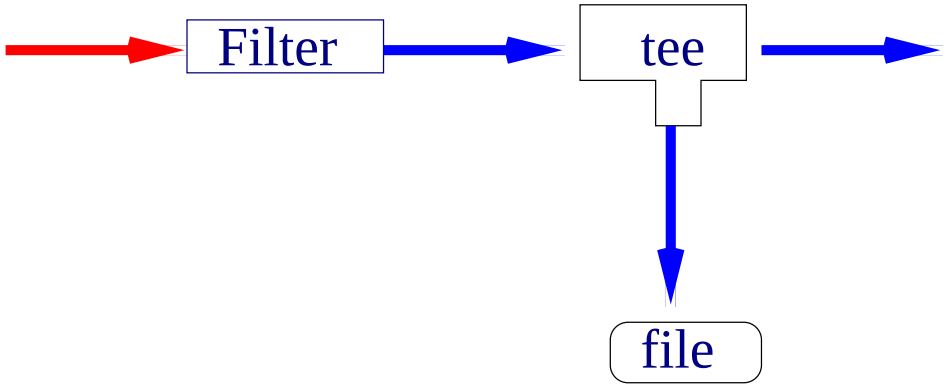
```
calculate 3>&2 2>&1 1>&3
```

The pipe | connects only standard output to standard input, but due to redirection we can make whatever we want!

Duplicating Streams: *tee*

clone, n: 1. An exact duplicate, as in "our product is a clone of their product." 2. A shoddy, spurious copy, as in "their product is a clone of our product."

Suppose we want to *both* save the output and analyze it? In plumbing we have T-connectors. In Unix we have *tee*.



Example:

```
calculate |tee results | analyze > processed_results
```

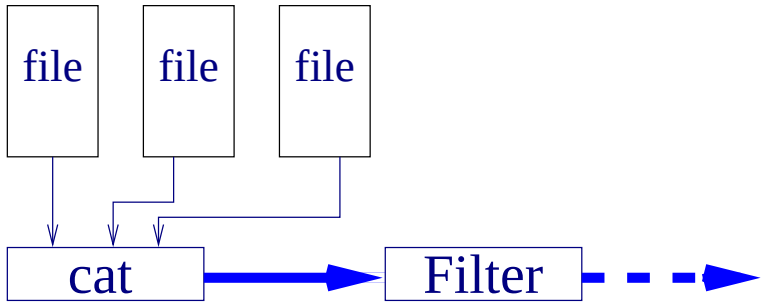
GNU *tee* can write to several files, append (*-a*), etc.

Concatenating Files: *cat*

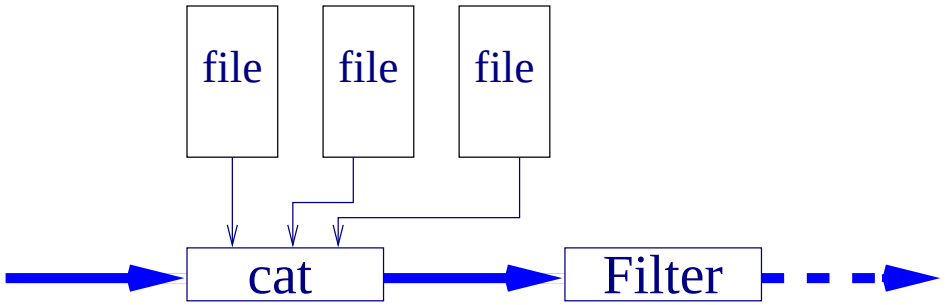
A commune is where people join together to share their lack of wealth. *R. Stallman*

Besides forking we need joining. . .

cat (from *concatenate*) takes several files and joins them together:



cat can even use standard input:



Example:

```
cat file1 file2 - file3
```

cat will dump *file1*, then *file2*, then its input, then *file3*.