Building Software Using Pipes and Filters

Frederick M. Avolio

Digital Equipment Corporation

6406 Ivy Lane, COP/8–6

Greenbelt, MD 20770

avolio@dco.dec.com

- Some material from Greg Tarsa, Tarsa Software Consulting, NH
- Some examples from The UNIX Programming Environment by Kernighan and Pike
- UNIX is a registered trademark of USL

Overview

- Building one-time-use small tools. (E.g., manipulating data in a file.)
- Building many-time-use small tools. (E.g., system adminstration aides.)
- Building prototype software.

Pipe

gets diagram

Pipe

```
% who | mail joe

% spell file | more
```

Filter

gets diagram

Filter

```
% grep #include *.c

% sort file | awk '{ print $2, $1 }'
```

Examples

```
% from

From research!bc Fri Aug 26 12:36:02 1988
From hadron!klr  Fri Aug 26 13:43:21 1988

% wc -l /etc/passwd

117

% from | wc -l

2

%
```

Examples

```
$ cat mailc

echo You have `from | wc -l` mail messages
exit 0
$
$ mailc

You have 2 mail messages

$
```

Examples

```
csh> rwho -a | grep mikey
mikey    umd5:ttyp0   Mar  2 19:59
mikey    umd5:ttyp2   Mar  2 16:43  3:38

csh> readnews -n comp.unix.wizards -l\
     | grep -i ultrix
comp.unix.wizards/1126 Ultrix and login:
comp.unix.wizards/1175 Ultrix & uucp
comp.unix.wizards/1196 Re: Ultrix & uucp

csh> ls -l | grep -v temp
          .
          .
          .
```

Examples

```
csh> grep ^BADSU /usr/adm/sulog \
      |mail root

csh> grep ^SU /usr/adm/sulog \
      |mail root
```

Examples

Login names from the password file (ignoring UUCP logins):

```
avolio:NN7d5FJw2LfSY:100:300:Frederick M. Avolio:\
     /home/avolio:/usr/bin/csh

csh> grep -v ^U /etc/passwd | awk -F: '{ print $1 }'
```

Word Frequency

Programming Pearls, Jon Bentley, June 1986 CACM

```
csh> cat textfile
In the beginning was the Word,
and the Word was with God,
and the Word was God.
He was with God, in the
beginning.  John 1:1-2
```

Word Frequency

```
csh> wordfreq < textfile
   5 the
   4 was
   3 word
   3 god
   2 with
   2 in
   2 beginning
   2 and
   1 john
   1 he
csh>
```

1. Make all words lower case

```
csh> tr A-Z a-z < textfile
in the beginning was the word,
and the word was with god,
^C
csh>
```

2. Get one word per line

```
csh> !! | tr -cs a-z '\012'
tr A-Z a-z < textfile | tr -cs a-z '\012'
in
the
beginning
^C
csh>
```

3. Bring identical words together

```
csh> !! | sort
tr A-Z a-z < textfile | tr -cs a-z '\012'|\
                  sort
and
and
beginning
beginning
god
^C
```

4. Count repeated words

```
csh> !! | uniq -c
tr A-Z a-z < textfile | tr -cs a-z '\12' | sort | uniq -c
   2 and
   2 beginning
   3 god
   1 he
   2 in
   1 john
   5 the
   4 was
   2 with
   3 word
csh>
```

5. Sort to decreasing order

```
csh> !! | sort -rn
tr A-Z a-z < textfile | tr -cs a-z '\12' |\
  sort | uniq -c | sort -rn
   5 the
   4 was
   3 word
   3 god
   2 with
   2 in
   2 beginning
   2 and
   1 john
   1 he
```

Word Frequency

```
$ cat wordfreq
#!/bin/sh
tr A-Z a-z |
tr -cs a-z '\012' |
sort |
uniq -c |
sort -rn
```

Spelling Checker

Programming Pearls, Jon Bentley, May 1985, CACM

1. Get words from file.

2. Sort them and “unique” them.

3. Check against dictionary.

Spelling Checker

```
$ cat spellck
#!/bin/sh
   tr A-Z a-z |
      tr -cs a-z '\012' |
         sort |
            uniq  |
               common -2 /usr/dict/words
```

Grep

- All lines with *string*
- All lines without *string*

Grep

- Search for a pattern of characters (*regular expression*) in a file(s)
- g/*regular expression*/p
- grep – simple regular expressions
- fgrep – fixed string(s)
- egrep – multiple regular expressions

Grep

grep *pattern* [**file** ...]

- pattern
  - regular expression, string
  - as in **ed(1)**
- file
  - file(s) to be searched

Grep

```
csh> grep #ifdef *.c *.h
deliver.c:#ifdef RECORD
main.c:#ifdef HOSTNAME
main.c:#ifdef GETHOSTNAME
defs.h:#ifdef SENDMAIL
defs.h:#ifdef BSD

csh> grep -n #ifdef *.c
deliver.c:48:#ifdef RECORD
main.c:189:#ifdef HOSTNAME
main.c:193:#ifdef GETHOSTNAME
defs.h:88:#ifdef SENDMAIL
defs.h:100:#ifdef BSD
```

Grep

```
csh> cat wordlist
wobegon
archenland
recieve
gildor


csh> fgrep -i -f wordlist file?
file1:before him, Lord Gildor reached
file1:can you not recieve what was
file2:the south of Archenland they rode
^C
```

Grep

```
csh> grep the file
In the past, electronic mail has used
details, you should read some of the
documents referenced at the end.
^C
csh> grep ^the file
the Reynoldsburg C.O.), 1234
the same address no matter where
the location of the sender.
^C
csh> grep the\$ file
you should read some of the
 divide into streets, the
name ‘‘AT&T Bell Laboratories’’ and the
^C
```

Grep

```
csh> rwho -a | grep mikey
mikey    umd5:ttyp0   Mar  2 19:59
mikey    umd5:ttyp2   Mar  2 16:43  3:38

csh> rwho -a |egrep "(huey) | (louie) | (duey)"
```

Grep

```
csh> fgrep -l sys/file.h *.c *.h
date.c
ld.c
login.c
passwd.c
strip.c
tar.c
test.c
```

Sed

sed *edit commands* [**file** ...]

- Edit files (**ed(1)**)
- Edit as passes through
- Can edit large or many files
- Can edit **stdin**
- Does not change the file! (Edited files go to **stdout.**)

cmd1 | sed *args* | *cmd2*

Sed

Selective printing

The first 20 lines of file:

```
csh> sed 20q file
```

All lines after first 300 lines

```
csh> sed 1,300d file.c
```

Other examples

```
csh> sed -n '301,$p' file.c
csh> sed -n 20,25p file.c
```

Sed

```
csh> mv file.c OLDfile.c
csh> sed 's/<term.h>/<sys\/term.h>/' \
     OLDfile.c > file.c


csh> foreach i ([a-z]*.c)
?    mv $i OLD$i
?    sed 's/<term.h>/<sys\/term.h>/'\
     OLD$i > $i
?    end

csh>
```

Sed

Mispelled Words

```
csh> cat /tmp/sedfile
s/recieve/receive/
s/[Uu]nix/UNIX/
s/DEC/Digital/
/Avolio/s/Phred/Fred/

csh> sed -f /tmp/sedfile doc > NEWdoc
```

Sed

Multiple changes based on pattern

Replace all include lines "#include <name>" with an explicit path.

(EG, *#include <sys/term.h>* becomes *#include "/u/other/sys/term.h"*)

```
csh> cat sedjunk
/#include/{
        s;<;"/u/other/;
        s/>/"/
        }

csh> sed -f sedjunk file.c > NEWfile.c
```

Sed

```
csh> egrep -i fred people.dec \
     people.non-dec non-people.dec
people.dec:Frederick Avolio
people.dec:Alfred Smith
people.dec:Fred Jones
people.non-dec:Fred Allen
people.non-dec:Fred Sanford
^C

csh> egrep -i fred people.dec \
           people.non-dec non-people.dec \
     | sed 's/.*://' | sort +2 > freds
```

Sed

Print paragraphs containing a given string

```
#!/bin/sh
# extracted from usenet.
#  written by Maarten Litmaath in Amsterdam.
# quote slashes in the pattern
pattern=‘echo "$1" | sed ’s-/-\\\\/-g’‘
shift

tab="	" #this is a hard tab
# the pattern of an empty line
empty="^[ $tab]*$"
SED="
: gap
/$empty/!b para
: gap1
n
b gap
: para
/$pattern/b found
H
n
/$empty/!b para
: cleanup
s/.*//
x
b gap1
```

Print paragraphs containing a given string – continued

```
: found
H
\${
g
P
}
n
/$empty/!b found
g
p
b cleanup
"

sed -n "$SED" $*
```

Awk

- Aho, Weinberger, and Kernighan
- Scans patterns, "Does things" based on what patterns are matched
- Programming Language
  - contants, variables
  - arithmetic, string functions
  - print, printf
  - if-then, for, etc.
- Deals with records and fields

Awk

awk 'program' [**file** ...]

file(s) – read as a stream (stdin)

program – awk program to apply to file

```
[pattern] { action }
[pattern] { action }
[pattern] { action }
        .
        .
        .
```

Awk

Input records—lines— broken up into fields (white space)

- $1, $2, ... $NF
- $0
- NF
- NR
- FILENAME

Awk

Patterns

- Regular expressions
- Conditionals

/RE/ { action }

/RE/~$2 *{ action }*

/RE/!~$2 *{ action}*

condition { action }

Examples

$\$1 > 4$ { action }

$\$4 <= 10$ { action }

$\$9 == "string"$ { action }

$\$2 > 3 || \$5 == "Test"$ { action }

Examples

Name and time of login sorted by time:

```
csh> who | awk '{ print $5, $1 }' | sort
07:55 joe
08:20 jill
13:48 fred
```

Awk

Examples

Login names fom the password file (ignoring UUCP logins):

```
csh> awk -F: '{ print $1 } /etc/passwd \
     | sed /^U/d  | sort > users
```

```
csh> awk -F: '$1 \!~ /^U/  {print $1 } /etc/passwd \
     |sort > users
```

Awk

Using a file

```
csh> cat prog1.awk
{count = count+1}
$2 ~ /bossman/ {boss = boss+1}
END {printf "You have %d msgs, %d from your boss.\n", \
        count, boss}

csh> from | awk -f prog1.awk

You have 143 msgs, 12 from your boss.

csh>
```

Finding double words in a text file

```
# if a new file...
FILENAME != prevfile {
  NR = 1   # reset line
  prevfile = FILENAME
  }
NF > 0 {
  if ( $1 == lastword)
     printf "double %s, file %s, line %d\n", \
              $1, FILENAME, NR;
     for (i = 2; i <= NF; i++)
        if ($i == $(i-1))
           printf "double %s, file %s, line %d\n", \
               $i, FILENAME, NR;
       if ( NF > 0)
           lastword = $NF
       }

csh> awk -f double.awk file1 file2 file3 ...
```

Disk Usage by Old Files

```
csh> du -a `find . -mtime +100 -print` \
   |  awk '                              \
         $1 != ""  {total += $1;  \
                    print $0}     \
         END {printf "%d\tTotal\n",\
              total}'
```

Awk

Stats on a file

Sample of file:

```
/usr/spool/news/comp/arch/477
/usr/spool/news/comp/sources/wanted/679
/usr/spool/news/comp/os/minix/374
/usr/spool/news/comp/sys/ibm/pc/2207
```

Goal:

Get a total of how many articles per news group for a given day.

Histstat shell script

```
csh> cat histstat

#! /bin/sh
sed -e 's/\/[0-9]*$//' \
    -e  's@/usr/spool/news/@@' \
    -e  's@/@.@g' \
    | sort | uniq -c | sort -rn |
    awk '{ print $0 }
         { s += $1 }
     END { printf "\n  %g  TOTAL\n", s } '
```

Histstat: Output

```
csh> histstat < ~news/today
  20 comp.sys.ibm.pc
  18 rec.arts.movies
  15 sci.med
  14 rec.humor
          .
          .
          .

  291  TOTAL

csh>
```

Summary

10 Tenets of UNIX Philosophy

1. Smaller is better
2. Make each program do one thing well.
3. Build a prototype as soon as possible.
4. Choose portability over efficiency.
5. Use shell scripts whenever possible.
6. ASCII data files are preferable to binary files.
7. Learn how to use regular expressions.
8. Avoid captive user interfaces.
9. Every program should act as a filter.
10. Save trees.

Thanks to Mike Gancarz