Summer 2013 Handout 12

Rule 1 for formatting data: only one item per line

```
1$ man ascii
2$ awk '27 <= NR && NR <= 31' /usr/pub/ascii
```

| *is one of the characters Handout 2, p. 1.*

```
| 40  @ | 41  A | 42  B | 43  C | 44  D | 45  E | 46  F | 47  G |
| 48  H | 49  I | 4a  J | 4b  K | 4c  L | 4d  M | 4e  N | 4f  O |
| 50  P | 51  Q | 52  R | 53  S | 54  T | 55  U | 56  V | 57  W |
| 58  X | 59  Y | 5a  Z | 5b  [ | 5c  \ | 5d  ] | 5e  ˆ | 5f  _ |
| 60  ‘ | 61  a | 62  b | 63  c | 64  d | 65  e | 66  f | 67  g |
```

```
3$ awk '27 <= NR && NR <= 31' /usr/pub/ascii | wc -l        off by a factor of eight
4$ awk '27 <= NR && NR <= 31' /usr/pub/ascii | sort -n      drags around whole lines
5$ awk '27 <= NR && NR <= 31' /usr/pub/ascii | grep 'A'     outputs 7 unwanted codes
```

```
40 @
41 A
42 B
43 C
44 D
45 E                                  etc.
```

To convert the format from 1-per-line to 8-per-line is simple. The third argument is minus lowercase L sixteen. For the other arguments of **pr**, see Handout 4, p. 2.

```
6$ pr -8 -i' '1 -l16 -t | more
```

To convert from 8-per-line to 1-per-line is harder:

```
#!/bin/ksh
#Convert the 8-per-line format to 1-per-line.

awk '{print \
     $2,  $3 "\n" \
     $5,  $6 "\n" \
     $8,  $9 "\n" \
    $11, $12 "\n" \
    $14, $15 "\n" \
    $17, $18 "\n" \
    $20, $21 "\n" \
    $23, $24 "\n" \
}'
```

Rule 2 for formatting data: only one line per item

```
Galt,John
(212)999-9999
Rearden,Hank
(215)765-4321
Roark,Howard
(212)211-1111
```

For **`previous`**, see Handout 11, pp. 5−6. You can remove the **`BEGIN {previous = ""}`** (Handout 11, p. 4) and the **`{print $0}`** (Handout 4, p. 16, lines 3−6).

```
#!/bin/ksh
#Output Hank Rearden's phone number.

awk '
    BEGIN                      {previous = ""}
    previous == "Rearden,Hank" {print $0}
                               {previous = $0}
'
```

```
(215)765-4321
```

Write each person all on one line:

```
(212)999-9999:Galt,John
(215)765-4321:Rearden,Hank
(212)211-1111:Roark,Howard
```

Now the **`awk`** needs only one pattern-action pair:

```
1$ awk -F: '$2 == "Rearden,Hank" {print $1}'      Output Hank's phone number
(215)765-4321
```

Rule 3 for formatting data: use the same delimiter all across the line

But suppose we want to output only Hank's area code. We would have to remove everything except the contents of the leftmost pair of parentheses:

```
#!/bin/ksh
#Output Hank Rearden's area code.

awk -F: '$2 == "Rearden,Hank" {print $1}' |
sed '
    s/^[^(]*(//;  #remove everything up to and including leftmost (
    s/).*//;      #remove everything after and including rightmost )
'
```

```
215
```

To feed the data more easily into **`awk`** or Perl, delimit the fields with all blanks, or with all tabs, or with all of one kind of punctuation mark,

```
212:9999999:Galt:John
516:7654321:Rearden:Hank
212:2111111:Roark:Howard
```

Now the **`sed`** is no longer needed. In Perl, **`-a`** is "autosplit", **`-n`** is "no print", and **`-e`** means that the following argument is the program to be executed, not an input file.

```
1$ awk -F: '$4 == "Hank" && $3 == "Rearden" {print $1}'        Hank's area code
2$ perl -F: -ane 'print "$F[0]\n" if $F[3] eq "Hank\n" && $F[2] eq "Rearden"'

    215
```

For another file that uses different delimiters, see Handout 5, pp. 9−10. For formatting and indenting, see Handout 6, pp. 4−5.

Render binary data into ASCII

For the **netpbm** documentation,

http://netpbm.sourceforge.net/doc/

http://i5.nyu.edu/~mm64/INFO1-CE9545/src/giftotext

```
#!/bin/ksh
#Output a GIF file as decimal ASCII numbers, one pixel per line.
#pnmtoplainpnm outputs 6 pixels per line, separated by a double bank.
#The sed changes each double blank to a newline,
#so that each pixel is output on a separate line.

export PATH=$PATH:\
˜mm64/public_html/INFO1-CE9236/netpbm/bin

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:\
˜mm64/public_html/INFO1-CE9236/netpbm/lib

˜mm64/public_html/INFO1-CE9236/netpbm/bin/giftopnm |
˜mm64/public_html/INFO1-CE9236/netpbm/bin/pnmtoplainpnm |
sed 's/\([0-9][0-9]* [0-9][0-9]* [0-9][0-9]*\)[   ][  ]*/\1\
/g'

exit 0

sed 's/  /\
/g'
```

```
4$ $S45/giftotext < ˜mm64/public_html/INFO1-CE9545/src/construction.gif | head
P3             "plain" PPM format magic number: http://netpbm.sourceforge.net/doc/ppm.html
38 38          width and height, in pixels
255            maximum color value; minimum is zero
255 255 255    Red, green, blue.  Three maximums make white.
255 255 255
255 255 255
255 255 255
255 255 255
255 255 255
255 255 255

5$ $S45/giftotext < ˜mm64/public_html/INFO1-CE9545/src/construction.gif |
awk '97 <= NR && NR <= 98'
255 255 255    Red and green make yellow, slightly mixed with black for anti-aliasing.
               brighter shade of yellow, unmixed with black
```

Render binary data into ASCII, edit it, and render it back into binary

You can remove the **`{print $0}`**; see Handout 4, p. 16, lines 3−5. The numbers output by the shellscript must be whole numbers, not numbers with fractions. If the average **`($1 + $2 + $3) / 3`** is not a whole number, the surrounding **`int()`** will make it a whole number. See p. 119.

http://i5.nyu.edu/~mm64/INFO1-CE9545/src/bw

```
#!/bin/ksh
#Change the text representation of a GIF from color to
#black and white.  Compute the average of the red, green,
#and blue components of each pixel.

awk '
    NR <= 3 {print $0}
    NR >  3 {a = int(($1 + $2 + $3) / 3); print a, a, a}
'

exit 0
```

```
1$ $S45/giftotext < ~mm64/public_html/INFO1-CE9545/src/construction.gif |
$S45/bw |
awk '97 <= NR && NR <= 98'
255 255 255     Yellow became a shade of gray.
0 0 0           a slightly lighter shade of gray
```

http://i5.nyu.edu/~mm64/INFO1-CE9545/src/texttogif

```
#!/bin/ksh
#Convert the text representation of a GIF back to binary.

/opt/sfw/netpbm/bin/ppmtogif
```

```
2$ cd ~/public_html
3$ pwd

4$ $S45/giftotext < ~mm64/public_html/INFO1-CE9545/src/construction.gif |
    $S45/bw |
    $S45/texttogif > construction_bw.gif

5$ chmod 444 construction_bw.gif
6$ ls -l construction_bw.gif
```

To see the before and after images, point your browser at

```
http://i5.nyu.edu/~mm64/INFO1-CE9545/src/construction.gif
http://i5.nyu.edu/~mm64/INFO1-CE9545/src/construction_bw.gif
```

▼ Homework 12.1: write a filter to stretch a GIF horizontally

Write a shellscript named **`stretch`** that will double the width of the text representation of a GIF. The second line of the standard output of **`stretch`** will contain a width that it twice as big as the one on the second line of the standard input. (Keep the height the same.) Then output two consecutive copies of each pixel. Test **`stretch`** like this:

```
1$ cd ~/public_html
2$ pwd
```

```
3$ $S45/giftotext < ˜mm64/public_html/INFO1-CE9545/src/construction.gif |
    stretch |
    $S45/texttogif > construction_stretched.gif

4$ chmod 444 construction_stretched.gif
5$ ls -l construction_stretched.gif
```

and then point your browser at

```
http://i5.nyu.edu/~mm64/INFO1-CE9545/src/construction.gif
http://i5.nyu.edu/~abc1234/construction_stretched.gif
```

▲

Do the same thing for sound files

The sound exchange utilities at **`http://sox.sourceforge.net/`** will render as sound as text.

```
1$ man sox
2$ man -s 5 soxexam                  examples
```

The arguments **`-t dat`** format the output as text; the argument **`-`** directs the output to the standard output.

```
3$ sox $S45/monkey.au -t dat - | head -6 | cat -n

4$ bc                                Handout 2, pp. 20−21
scale = 11
1 / 8012
.00012481278
control-d
5$
```

If necessary, use **`tr`** to remove the carriage return from the end of each line of text output by **`sox`**.

Run a test suite

```
1$ prog < test1 > result1
2$ cmp result1 correct1

3$ prog < test2 > result2
4$ cmp result2 correct2

5$ prog < test3 > result3
6$ cmp result3 correct3
```

```
#!/bin/ksh
#Run all the tests and output the numbers of the incorrect ones.
#Return exit status 0 if all the tests were passed, 1 otherwise.

cd /test/directory
status=0                #innocent till proven guilty

n=1
while [[ $n -le 1000 ]]
do
    prog < test$n > result$n
    if cmp -s result$n correct$n   #Handout 4, p. 27 for -s
    then
        rm result$n
    else
        echo $n
        status=1
    fi
    let n=n+1
done

exit $status
```

Kill every process owned by a person who owns more than 50

Handout 2, p. 15

```
1$ ps -Af | more
     UID   PID  PPID   C    STIME TTY         TIME CMD
    root  2360  1655   0   Mar 29 ?           0:01 /usr/sbin/sh /lib/svc/method/svc-dlm
    root  2761  1655   0   Mar 29 ?           1:14 /usr/lib/ssh/sshd
   yc801 12235 12228   0   Apr 08 pts/7       0:03 /bin/ksh
  daemon  2392  1655   0   Mar 29 ?          27:37 /lib/crypto/kcfd
```

The pipeline in the first pair of back quotes outputs the loginname of each person who owns more than 50 processes. The pipeline in the second pair of back quotes outputs the PID number of each process owned by **`$loginname`**. Note that the **{**curly braces**}** around the **`print`**, and the **"**double quotes**"** around the string, are both within the single-quoted territory.

```
#!/bin/ksh
#Kill every process owned by a person (other than "root" and "daemon")
#who owns more than 50.

ps -Af |
awk ’NR > 1 && $1 != "root" && $1 != "daemon" {print $1, $2}’ > ˜/ps.out

for loginname in ‘awk ’{print $1}’ ˜/ps.out | sort | uniq -c |
    awk ’$1 > 50 {print $2}’‘
do
    kill -9 ‘awk ’$1 == "’$loginname’" {print $2}’ ˜/ps.out‘
done

rm ˜/ps.out
exit 0
```

Here’s the contents of the **`˜/ps.out`** file:

```
yc801 12235
```

□