Summer 2013 Handout 11

We have three versions of awk

> He turned out to be a young, blond SAS guerrilla-warfare expert with the peculiar nickname of Awk, a name said to vaguely resemble the grunting noise he would make on maneuvers.
>
> —George Crile, *Charlie Wilson's War*, Chapter 14

The **-prune** line means ''skip any directory named **home1** and its subdirectories''. The **! -name '*.awk'** means ''don't output the names of files that end with **.awk**''. **-perm -111** means ''at least all three of the execute bits **--x--x--x** must be turned on'' (Handout 10, p. 1).

```
#!/bin/ksh
#Output the full pathname of each of our versions of awk.

find / \
    -type d -name home1 -prune \
    -o \
    -type f \
    -name '*awk*' \
    ! -name '*.awk' \
    -perm -111 \
    -ls 2> /dev/null |
sort -n     #Put identical inode numbers next to each other.

exit 0
```

```
30515   58 -r-xr-xr-x   2 root     bin        103092 Jul  5  2012 /usr/bin/awk
30515   58 -r-xr-xr-x   2 root     bin        103092 Jul  5  2012 /usr/bin/oawk
30627   91 -r-xr-xr-x   1 root     bin        136376 Jul  5  2012 /usr/bin/nawk
88803   62 -r-xr-xr-x   1 root     bin         90456 Aug 27  2012 /usr/xpg4/bin/awk
```

```
1$ man awk      Aho, Weinberger, Kernighan: manual for /usr/bin/awk
2$ man nawk     new awk and /usr/xpg4/bin/awk
```

Regular expressions in the argument of awk

You must have **/**diagonal slashes**/** around a regular expression in the argument of **awk** (Handout 4, p. 17), just as you must have **{**curly braces**}** around a **print** and **"**double quotes**"** around a string. By default, the regular expression is applied to the entire line of input. For example, to list only the files but not the subdirectories:

```
1$ ls -l | tail +2 | grep '^-' | awk '{print $NF}' | more
2$ ls -l | awk 'NR >= 2 && /^-/ {print $NF}' | more
```

Apply a regular expression to one field

To see if any login name contains a character other than a lowercase letter or digit,

```
1$ grep '^[^:]*[^a-z0-9]' /etc/passwd | more
2$ awk -F: '$1 ~ /[^a-z0-9]/' /etc/passwd | more
```

We'll list all the courses given this semester (**20132**) on i5.nyu.edu. Thanks to the **$1 ~**, the **$** in example 4 means ''end of the first field'', not ''end of the line''.

The **+0.0 -0.1** will sort the courses in alphabetical order of the first character on each line. If there are two or more lines that begin with the first character, the **+0.1n** will sort them in increasing numerical order of the remaining characters on each line. See p. 106.

```
3$ grep '^[^:]*20132:' /etc/group | sort +0.0 -0.1 +0.1n | more
4$ awk -F: '$1 ~ /20132$/' /etc/group | sort +0.0 -0.1 +0.1n | more
```

The opposite of **~** is **!~**. To see if anyone has a home directory that is not a descendant of **/home1**,

```
5$ grep -v '^[^:]*:[^:]*:[^:]*:[^:]*:[^:]*:/home1/' /etc/passwd | more
6$ awk -F: '$6 !~ /^\/home1\//' /etc/passwd | more
root:x:0:0:root@i5:/root:/usr/bin/bash
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
```

The C Standard Library is in the dynamically linked ''shared object'' file **/lib/libc.so**. We will display the ''name table'' of this file with **nm** (Handout 3, p. 27) to verify that it contains the function **printf**. **$NF** is the last field on each line; **$NF - 1** is the last field minus 1 (if the last field is a number), and **$(NF - 1)** is the next-to-last field. The operand of this **$** must not be negative; **$(NF - 1)** exists only if **NF > 0**.

```
7$ nm -D /lib/libc.so |
awk -F'|' 'NR == 5 || NR >= 7 && $NF == "printf" && $(NF - 1) !~ /UNDEF/'
[Index]   Value      Size      Type  Bind  Other Shndx   Name
[453]   |    635548|       396|FUNC |GLOB |3    |16     |printf
```

Arithmetic in the argument of awk

You can write expressions using fields or other variables. Let's list the size of each file in megabytes instead of bytes:

```
1$ cd /var/apache2/2.2/logs
2$ ls -l | more
total 46583
-rw-r--r--   1 root     root     85611770 May 28 16:56 access_log
-rw-r--r--   1 root     root     31888903 May 28 16:55 error_log
-rw-r--r--   1 root     root         602 Jul 16  2012 error_log.0.gz

3$ ls -l | awk 'NR > 1 {print $5 / (1024 * 1024) "M\t" $NF}'
81.6457M access_log
30.4116M error_log
0.000574112Merror_log.0.gz
```


More than one pattern-action pair

We have to use **/usr/xpg4/bin/awk** or **/usr/bin/nawk**, because **/usr/bin/awk** does not have ^ for exponentiation.

http://i5.nyu.edu/~mm64/INFO1-CE9545/src/pairs1

```
#!/bin/ksh


ls -l |
tail +2 |
/usr/xpg4/bin/awk '
    $5 < 1024                   {print $5           "\t" $NF}
    $5 >= 1024   && $5 < 1024^2 {print $5 / 1024   "K\t" $NF} #kilo
    $5 >= 1024^2 && $5 < 1024^3 {print $5 / 1024^2 "M\t" $NF} #mega
    $5 >= 1024^3 && $5 < 1024^4 {print $5 / 1024^3 "G\t" $NF} #giga
    $5 >= 1024^4 && $5 < 1024^5 {print $5 / 1024^4 "T\t" $NF} #tera
    $5 >= 1024^5                {print $5 / 1024^5 "M\t" $NF} #peta
'

exit 0
```

```
1$ cd /var/apache2/2.2/logs
2$ pairs1
81.6457M access_log
30.4116M error_log
602      error_log.0.gz
```

The BEGIN and END patterns

Unlike the Korn shell, **awk** allows whitespace on either side of the **=**. As in C, you can change the statement **sum = sum + $5** to **sum += $5** (p. 118). The **BEGIN** pattern/action pair is optional.

```
#!/bin/ksh
#Output the total size in bytes of all the files in the current
#directory.

ls -la |
awk '
    BEGIN           {sum = 0}
    NR >= 2 && /^-/ {sum = sum + $5}
    END             {print "sum:", sum}
'

exit 0
```

You need a semicolon between two statements on the same line in an action. As in C, you can change the statement **count = count + 1** to **++count** (p. 121).

```
#!/bin/ksh
#Output the total and average size in bytes of all the files in
#the current directory.  Bug: divides by 0 if no files.

ls -la |
awk '
    NR >= 2 && /^-/ {sum = sum + $5; count = count + 1}
    END             {print "sum:", sum, "average:", sum / count}
'

exit 0
```

The **{** that begins an action must be on the same line as the pattern that governs that action, but the rest of the action can be on the following lines. Now that the statements in the actions are each on a separate line, we no longer need the semicolons.

```
#!/bin/ksh
#Output the total and average size in bytes of all the files in
#the current directory.  Bug: divides by 0 if no files.

ls -la |
awk '
    NR >= 2 && /^-/ {
        sum = sum + $5
        count = count + 1
    }

    END {print "sum:", sum, "average:", sum / count}
'

exit 0
```

An action that contains an if statement

```
#!/bin/ksh
#Output the total and average size in bytes of all the files in
#the current directory.

ls -la |
awk '
    NR >= 2 && /^-/ {
        sum = sum + $5
        count = count + 1
    }

    END {
        if (count > 0) { #parens instead of double square brackets
            print "sum:", sum, "average:", sum / count
        } else {
            print "sum:", sum
        }
    }
'

exit 0
```

http://i5.nyu.edu/~mm64/INFO1-CE9545/src/pairs2

```
#!/bin/ksh

ls -l |
/usr/xpg4/bin/awk '
    NR > 1 && /^-/ {
        if ($5 < 1024) {
            print $5 "\t" $NF
        } else if ($5 < 1024^2) {
            print $5 / 1024   "K\t" $NF #kilo
        } else if ($5 < 1024^3) {
            print $5 / 1024^2 "M\t" $NF #mega
        } else if ($5 < 1024^4) {
            print $5 / 1024^3 "G\t" $NF #giga
        } else if ($5 < 1024^5) {
            print $5 / 1024^4 "T\t" $NF #tera
        } else if ($5 < 1024^6) {
            print $5 / 1024^5 "P\t" $NF #peta
        } else {
            print $5 "\t" $NF
        }
    }
'

exit 0
```

Render an HTML table into a format that can be read easily by awk.

For an HTML table, see Handout 3, pp. 10−11. For **lynx**, see Handout 4, p. 30.

Central Park	blah	blah	57
White Plains	blah	blah	56

The table looks like this after the **egrep**, except that the **<TABLE>** and **</TABLE>** tags were removed by the **egrep**:

```
<TABLE>
<TR><TD>Central Park</TD> <TD>blah</TD> <TD>blah</TD> <TD>57</TD></TR>
<TR><TD>White Plains</TD> <TD>blah</TD> <TD>blah</TD> <TD>56</TD></TR>
</TABLE>
```

The table looks like this after the second **sed**:

```
<TABLE>
<TR>@Central Park@ @blah@ @blah@ @57@</TR>
<TR>@White Plains@ @blah@ @blah@ @56@</TR>
</TABLE>
```

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

```
#!/bin/ksh
#Yonkers temperature is average of Central Park and White Plains.

lynx -source \
http://forecast.weather.gov/obslocal.php\
'?warnzone=NYZ071&local_place=Yonkers+NY&zoneid=EST&offset=18000' |

tr '[A-Z]' '[a-z]' |

#Insert a newline immediately after each </TR>
#to ensure that each row of the HTML table is on a separate line.
sed 's:</tr>:&\
:' |

egrep 'central park|white plains' |

#Change every <TD> or </TD> tag to a @.
sed 's:</*td[^>]*>:@:g' |

awk -F@ '
    BEGIN {sum = 0}
          {sum = sum + $8}
    END   {print sum / NR}
'
```

```
    56.5
```

Look for a pattern that straddles a pair of consecutive lines

grep can search only for a pattern that is confined to one line. The **BEGIN {previous = ""}** and the **$0 ~** are the default and can be removed.

```
#!/bin/ksh
#Output every pair of consecutive lines where the first
#ends with "hel-" and the second starts with "lo".

awk '
    BEGIN {previous = ""}

    previous ~ /hel-$/ && $0 ~ /^lo/ {
        print NR - 1, previous
        print NR, $0
    }

    {previous = $0}
'

exit 0
```

```
#!/bin/ksh
#Another way to format the same program.

awk '
    BEGIN                              {previous = ""}
    previous ~ /hel-$/ && $0 ~ /^lo/ {print NR - 1, previous; print NR, $0}
                                       {previous = $0}
'

exit 0
```

▼ Homework 11.1: nature vs. nurture

Is there a line in **`$S45/Shakespeare.complete`** mentioning the word ‘‘nature’’, followed immediately by a line mentioning the word ‘‘nurture’’? Look for both upper and lowercase.

▲

Print the lines, if any, between two given lines

For the output of **`cvs log`**, see Handout 8, pp. 3 and 4. We will print the first field of every line between the **`locks:`** line and the **`access list:`** line.

```
#!/bin/ksh
#Output the login name of anyone who has locked the jokes file.

cvs log jokesdir/jokes |
awk '
    BEGIN           {between = 0}   #zero
    /^access list:/ {between = 0}
    between != 0    {print $1}
    /^locks:/       {between = 1}   #one
'

exit 0
```

Arrays in awk: pp. 122−123

$1 % 12 is the remainder left over after dividing **$1** by **12**. If **$1** is non-negative, the remainder will be in the range 0 to 11 inclusive.

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

```
#!/bin/ksh
#Each input line has a year.

awk '
    BEGIN {
        a[ 0] = "monkey"
        a[ 1] = "rooster"
        a[ 2] = "dog"
        a[ 3] = "pig"
        a[ 4] = "rat"
        a[ 5] = "ox"
        a[ 6] = "tiger"
        a[ 7] = "hare"
        a[ 8] = "dragon"
        a[ 9] = "snake"
        a[10] = "horse"
        a[11] = "sheep"
    }

    {print $1, a[$1 % 12]}
'

exit 0
```

```
1$ date | awk '{print $4}'
16:58:40

2$ date | awk '{print $4}' | $S45/year_to_animal
16:58:40 monkey
```

An array subscript does not have to be a number. It can be a string.

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

```
#!/bin/ksh
#Each input line has an anmal.

awk '
    BEGIN {
        a["monkey"]  = 2004
        a["rooster"] = 2005
        a["dog"]     = 2006
        a["pig"]     = 2007
        a["rat"]     = 2008
        a["ox"]      = 2009
        a["tiger"]   = 2010
        a["hare"]    = 2011
        a["dragon"]  = 2012
        a["snake"]   = 2013
        a["horse"]   = 2014
        a["sheep"]   = 2015
    }

    {print $1, a[$1]}
'

exit 0
```

```
3$ echo monkey | $S45/animal_to_year
monkey 2004
```

http://i5.nyu.edu/~mm64/INFO1-CE9545/src/pairs3

```
#!/bin/ksh
#exponent is 0 if the file is less than 1K.
#Otherwise, exponent is 1 if the file is less than 1M.
#Otherwise, exponent is 2 if the file is less than 1G.  Et cetera.
#log($5) / log(1024) is the logarithm of $5 to the base 1024.

ls -l |
/usr/xpg4/bin/awk '
    BEGIN {
        a[0] = ""
        a[1] = "K"   #kilo
        a[2] = "M"   #mega
        a[3] = "G"   #giga
        a[4] = "T"   #tera
        a[5] = "P"   #peta
    }

    NR > 1 && /^-/ {
        exponent = $5 == 0 ? 0 : int(log($5) / log(1024))
        print $5 / 1024^exponent a[exponent] "\t" $NF
    }
'

exit 0
```

Store every line of input into an array.

In the ‘‘hel-’’ ‘‘lo’’ example, **awk** temporarily remembered the previous line of input as it advanced to the next line. In the following example, **awk** will remember *every* previous line of input. The first line of input will be stored in **a[1]**. The second line of input will be stored in **a[2]**. By the time we get to the **END**, the last line of input will be stored in **a[NR]**.

See also the **backwards** example on p. 122.

```
#!/bin/ksh
#Input a list of numbers, one per line, and output their median.
#Begin by sorting the numbers.  If there is an odd number of numbers,
#the median is the middle number.
#If there is an even number of numbers, the median is the
#average of the two middle numbers.

sort -n |
awk '
    {a[NR] = $0}              #Copy each line into an array element.

    END {
        if (NR == 0) {
            print 0                                 #no numbers
        } else if (NR % 2 == 1) {
            print a[(NR + 1) /2]                    #odd
        } else {
            print (a[NR/2] + a[1 + NR/2]) / 2   #even
        }
    }
'

exit 0
```

An array subscript can be a long string. It can even be an entire line of input such as **$0**.

```
#!/bin/ksh
#Do the same thing as uniq with no arguments,
#except that the duplicate lines do not need to be consecutive.

awk '
               {a[$0] = a[$0] + 1}
    a[$0] == 1 {print $0}
'

exit 0
```

```
#!/bin/ksh
#Do the same thing as uniq with no arguments,
#except that the duplicate lines do not need to be consecutive.

awk '++a[$0] == 1'
exit 0
```


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

```
#!/bin/ksh
#Trace the ancestry of the given PID: Handout 6, pp. 2-3.

if [[ $# -ne 1 ]]
then
    echo $0: arg must be PID 1>&2
    exit 1
fi

ps -Af |
awk '
    NR == 1 {print}

    NR > 1 {
        line[$2] = $0        #$2 is PID
        parent[$2] = $3      #$3 is PPID
    }

    END {
        for (pid = '$1';; pid = parent[pid]) {
            if (line[pid] == "") {
                print "'$0': PID " pid " not found"
                exit 2
            }
            print line[pid]
            if (pid == parent[pid]) {   #reached the top
                exit 0
            }
        }
    }
'
```

```
1$ ancestry `ps -f | awk 'NR > 1 && $NF == "-ksh" {print $2}'`
     UID   PID  PPID   C    STIME TTY          TIME CMD
    mm64 15978 15977   0 15:11:15 pts/9       0:00 -ksh
    mm64 15977 15976   0 15:11:11 ?           0:00 /usr/lib/ssh/sshd
    root 15976  2761   0 15:11:11 ?           0:00 /usr/lib/ssh/sshd
    root  2761  1655   0   Mar 29 ?           1:14 /usr/lib/ssh/sshd
    root  1655  1655   0   Mar 29 ?           0:00 zsched
```

□