The Awk Text Processing Language

“”

Contents

Awk is a unix tool, or programming language designed to process and transform text files which are arranged in a series of 'fields' (chunks of text separated by spaces, or any other delimiter) and records. This document is mainly about the 'mawk' variant of 'awk'.

Find out the version of mawk

```
mawk -W version
```

helpful man pages for awk

`man gawk`	The gnu awk man page
`man ed`	Contains regular expression examples
`man mawk`	Contains good examples
`man regex`	Regular expression syntax

`http://sparky.rice.edu/awk.html`
more awk one liners

Section 1

Gotchas

[+] The so-called “gotchas” are small but potentially frustating problems which arise and which stop a program from working or which make the awk program work in an unexpected way. Gotcha derives from the contraction of the english phrase “got you”.

- ⋆ On a unix system the awk phrase <<awk “{print $1}”>> doesnt work as expected because the unix (bash) shell expands or “interpolates” the “$1” variable. It is necessary to write “awk '{print $1'}”
- ⋆

BEGIN and and variables such as FS must be uppercase

```
begin{FS=","}{print $2}     ~(No!! this doesnt work)
```

Simple Usage

Simple usage of awk on different operating systems.

```
   Unix: awk '/pattern/ {print "$1"}'    # standard Unix shells
DOS/Win: awk '/pattern/ {print "$1"}'    # compiled with DJGPP, Cygwin
         awk "/pattern/ {print \"$1\"}"  # GnuWin32, UnxUtils, Mingw
```

Users of MS-DOS or Microsoft Windows must remember that the percent sign (%) is used to indicate environment variables, so this symbol must be doubled (%%) to yield a single percent sign visible to awk.

Run an awk script

```
cat file1 | awk -f a.awk > file2
```

```
awk -f a.awk file1 > file2    ~(the same)
```


Strings

3.1 Concatenation Of String

Concatenation is the fancy term for joining 2 strings (bits of text) together.

Print the first two columns of the space/tab delimited file 'data.txt'

```
awk '{print $1 $2}' data.txt   ~($1 and $2 are printed with no space
  ⇒ between)
```

```
awk '{print $1$2}' data.txt    ~(the same, at least on my mawk version)
```

```
awk '{print $1 $2;}' data.txt    ~(the same again)
```

```
awk '{print $1 "" $2}' data.txt   ~(the same again, but why would you?)
```

Awk doesnt have variable 'interpolation' in strings

```
awk '{print "$1 ..."}' data.txt    ~(this prints '$1 ...' literally)
```

Print the first column of 'data.txt' with 3 dots '...' appended to it

```
awk '{print $1 "..."}' data.txt
```

Append a string to itself (string concatenation)

```
s = s "xxx";   ~(this appends 3 x's to the end of the string 's')
```

3.2 Matching Patterns

Determine if the variable "s" contains the letter "r"

```
s ~ /r/
```

*Print the first field of each line if it does *not* contain "a" or "b"*

```
$1 !~ /(a|b)/ { print $1 }
```

```
$1 !~ /[ab]/ { print $1 }      ~(the same)
```

Add an "X" between every letter of every line

```
{ gsub(//, "X") ; print }
```

Split the string "s" into the array A using the pattern "r"

```
split(s, A, r)
```

3.3 Printing Strings

Make a multiline string..

```
print "\
 <html> \n\
 <head> \n\
"
```

Print multiple expressions

```
print "variable a is " a "."
```

Its not possible to break printing expressions across lines

```
print "variable a is"
      a ".";
```

(this doesnt work, at least not with mawk 1.3.3)

3.4 Newlines

Display the file 'days.txt' with all newline characters removed

```
awk '{ printf "%s", $0 }' days.txt
```

```
cat days.txt | awk '{ printf "%s", $0 }'   ~(the same)
```

Display 'days.txt' with newline characters replaced with spaces

```
awk '{ printf "%s ", $0 }' days.txt
```

```
cat days.txt | awk '{ printf "%s ", $0 }'
```

Section 4

Arrays

Delete an array called record

```
delete record
```

Assign a value to an associative style array

```
a["cars"] = 3
```

Section 5

Regular Expressions

*Regular expression meta-characters: ^ $. [ ] — ( ) * + ?*
Print all lines which start with an awk identifier

```
BEGIN { identifier = "[_a-zA-Z][_a-zA-Z0-9]*" }
$0 ~ "^" identifier
```

Section 6

Loops

Loop through each field of each record

```
{ for(i = 1 ; i <= NF ; i++)  print $i }
```

Print each element of an array

```
for ( i in aa ) print aa[i]
```

Section 7

Splitting Data Fields

The field separator variable FS is interpreted as a regular expression

Split fields with any character followed by a colon ":" character

```
BEGIN {FS=".:"}
```

Split quoted comma delimited fields (csv)

```
BEGIN {FS="\" *, *\""}
```

awk built in variables

ARGC	Number of command line arguments.
ARGV	Array of command line arguments, 0..ARGC-1.
CONVFMT	Format for conversion of numbers to string, default "%.6g".
ENVIRON	Array indexed by environment variables. An environment string, var=value is stored as ENVIRON
FILENAME	Name of the current input file.
FNR	Current record number in FILENAME.
FS	Splits records into fields as a regular expression.
NF	Number of fields in the current record.
NR	Current record number in the total input stream.
OFMT	Format for printing numbers; initially = "%.6g".
OFS	Inserted between fields on output, initially = " ".
ORS	Terminates each record on output, initially = "\n".
RLENGTH	Length set by the last call to the built-in function, match().
RS	Input record separator, initially = "\n".
RSTART	Index set by the last call to match().
SUBSEP	Used to build multiple array subscripts, initially = "\034".

Section 8

Range Of Fields

Awk has no simple way to print a range of fields such as $[1-4] A 'for' loop must be used to loop through the range and print each one. One may use cut instead

Use 'cut' to print fields 1 to 5 from a comma delimited file

```
cut -d, -f1-5
```

Section 9

Awk One Line Recipes

These one line scripts were taken from http://www.pement.org/awk/awk1line.txt

30 April 2008, by Eric Pement - eric [at] pement.org, version 0.27

`http://www.pement.org/awk/awk1line.txt`
: Latest version of the Eric Pement one line scripts (in English)

`http://ximix.org/translation/awk1line_zh-CN.txt`
: Chinese version of these one line scripts

Section 10

File Spacing

Double space a file

```
awk '1;{print ""}'
```

```
awk 'BEGIN{ORS="\n\n"};1'    ~(another way)
```

Double space a file which already has blank lines in it. Output file Should contain no more than one blank line between lines of text. NOTE: On Unix systems, DOS lines which have only CRLF (\r\n) are Often treated as non-blank, and thus 'NF' alone will return TRUE.

```
awk 'NF{print $0 "\n"}'
```

Triple space a file

```
awk '1;{print "\n"}'
```

Section 11

Summing Numeric Columns

Sum up all the numbers in column 2 and print out the total at the end

```
awk '{ a+=$2 } END { print "total=" a }' data.txt
```

Sum a column between 2 lines in a file (with help from sed)

```
sed -n '/#1/,/#2/p' data.txt | awk -F, '{a+=$2; print $2, a}' | less
```

Section 12

Line Numbering

Precede each line by its line number FOR THAT FILE (left alignment). Using a tab (\t) instead of space will preserve margins.

```
awk '{print FNR "\t" $0}' files*
```

Precede each line by its line number FOR ALL FILES TOGETHER, with tab.

```
awk '{print NR "\t" $0}' files*
```

Number each line of a file (number on left, right-aligned) Double the percent signs if typing from the DOS command prompt.

```
awk '{printf("%5d : %s\n", NR,$0)}'
```

Number each line of file, but only print numbers if line is not blank Remember caveats about Unix treatment of \r (mentioned above)

```
awk 'NF{$0=++a " :" $0};1'
```

```
awk '{print (NF? ++a " :" :"") $0}'
```

Count lines (emulates "wc -l")

```
awk 'END{print NR}'
```

Print the sums of the fields of every line

```
awk '{s=0; for (i=1; i<=NF; i++) s=s+$i; print s}'
```

Add all fields in all lines and print the sum

```
awk '{for (i=1; i<=NF; i++) s=s+$i}; END{print s}'
```

Print every line after replacing each field with its absolute value

```
awk '{for (i=1; i<=NF; i++) if ($i < 0) $i = -$i; print }'
```

```
awk '{for (i=1; i<=NF; i++) $i = ($i < 0) ? -$i : $i; print }'
```

Print the total number of fields ("words") in all lines

```
awk '{ total = total + NF }; END {print total}' file
```

Print the total number of lines that contain "Beth"

```
awk '/Beth/{n++}; END {print n+0}' file
```

Print the largest first field and the line that contains it Intended for finding the longest string in field #1

```
awk '$1 > max {max=$1; maxline=$0}; END{ print max, maxline}'
```

Section 13

The Number Of Fields

Print the number of fields in each line, followed by the line

```
awk '{ print NF ":" $0 } '
```

Print the last field of each line

```
awk '{ print $NF }'
```

Print the last field of the last line

```
awk '{ field = $NF }; END{ print field }'
```

Print every line with more than 4 fields

```
awk 'NF > 4'
```

Print every line where the value of the last field is > 4

```
awk '$NF > 4'
```

Section 14

String Creation

Create a string of a specific length (e.g., generate 513 spaces)

```
awk 'BEGIN{while (a++<513) s=s " "; print s}'
```

Insert a string of specific length at a certain character position Example: insert 49 spaces after column #6 of each input line. gawk –re-interval 'BEGIN{while(a++<49)s=s " "};{sub(/^.{6}/,"&" s)};1'

Section 15

Array Creation

These next 2 entries are not one-line scripts, but the technique Is so handy that it merits inclusion here. Create an array named "month", indexed by numbers, so that month[1] Is 'Jan', month[2] is 'Feb', month[3] is 'Mar' and so on.

```
split("Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec", month, " ")
```

Create an array named "mdigit", indexed by strings, so that Mdigit["Jan"] is 1, mdigit["Feb"] is 2, etc. Requires "month" array

```
for (i=1; i<=12; i++) mdigit[month[i]] = i
```

Section 16

Text Conversion And Substitution

IN UNIX ENVIRONMENT: convert DOS newlines (CR/LF) to Unix format

```
awk '{sub(/\r$/,"")};1'   # assumes EACH line ends with Ctrl-M
```

IN UNIX ENVIRONMENT: convert Unix newlines (LF) to DOS format

```
awk '{sub(/$/,"\r")};1'
```

IN DOS ENVIRONMENT: convert Unix newlines (LF) to DOS format

```
awk 1
```

IN DOS ENVIRONMENT: convert DOS newlines (CR/LF) to Unix format Cannot be done with DOS versions of awk, other than gawk

```
gawk -v BINMODE="w" '1' infile >outfile
```

Use "tr" instead.

```
tr -d \r <infile >outfile          ~( GNU tr version 1.22 or higher )
```

Delete leading whitespace (spaces, tabs) from front of each line

```
awk '{sub(/^[ \t]+/, "")};1'       ~(aligns all text flush left)
```

Delete trailing whitespace (spaces, tabs) from end of each line

```
awk '{sub(/[ \t]+$/, "")};1'
```

Delete BOTH leading and trailing whitespace from each line

```
awk '{gsub(/^[ \t]+|[ \t]+$/,"")};1'
```

```
awk '{$1=$1};1'              # also removes extra space between fields
```

Insert 5 blank spaces at beginning of each line (make page offset)

```
awk '{sub(/^/, "     ")};1'
```

Align all text flush right on a 79-column width

```
awk '{printf "%79s\n", $0}' file*
```

Center all text on a 79-character width

```
awk '{l=length();s=int((79-l)/2); printf "%"(s+l)"s\n",$0}' file*
```

Substitute (find and replace) "foo" with "bar" on each line

```
awk '{sub(/foo/,"bar")}; 1'           # replace only 1st instance
gawk '{$0=gensub(/foo/,"bar",4)}; 1'  # replace only 4th instance
awk '{gsub(/foo/,"bar")}; 1'          # replace ALL instances in a line
```

Substitute "foo" with "bar" ONLY for lines which contain "baz"

```
awk '/baz/{gsub(/foo/, "bar")}; 1'
```

Substitute "foo" with "bar" EXCEPT for lines which contain "baz"

```
awk '!/baz/{gsub(/foo/, "bar")}; 1'
```

Change "scarlet" or "ruby" or "puce" to "red"

```
awk '{gsub(/scarlet|ruby|puce/, "red")}; 1'
```

Reverse order of lines (emulates "tac")

```
awk '{a[i++]=$0} END {for (j=i-1; j>=0;) print a[j--] }' file*
```

If a line ends with a backslash, append the next line to it (fails if There are multiple lines ending with backslash...)

```
awk '/\\$/ {sub(/\\$/,""); getline t; print $0 t; next}; 1' file*
```

Print and sort the login names of all users

```
awk -F ":" '{print $1 | "sort" }' /etc/passwd
```

Section 17

Rearranging Fields Or Columns

Print the first 2 fields, in opposite order, of every line

```
awk '{print $2, $1}' file
```

Switch the first 2 fields of every line

```
awk '{temp = $1; $1 = $2; $2 = temp}' file
```

Print every line, deleting the second field of that line

```
awk '{ $2 = ""; print }'
```

Print in reverse order the fields of every line

```
awk '{for (i=NF; i>0; i--) printf("%s ",$i);print ""}' file
```

Concatenate every 5 lines of input, using a comma separator between fields

```
awk 'ORS=NR%5?",":"\n"' file
```

Section 18

Selective Printing Of Certain Lines

Print first 10 lines of file (emulates behavior of "head")

```
awk 'NR < 11'
```

Print first line of file (emulates "head -1")

```
awk 'NR>1{exit};1'
```

Print the last 2 lines of a file (emulates "tail -2")

```
awk '{y=x "\n" $0; x=$0};END{print y}'
```

Print the last line of a file (emulates "tail -1")

```
awk 'END{print}'
```

Print only lines which match regular expression (emulates "grep")

```
awk '/regex/'
```

Print only lines which do NOT match regex (emulates "grep -v")

```
awk '!/regex/'
```

Print any line where field #5 is equal to "abc123"

```
awk '$5 == "abc123"'
```

Print only those lines where field #5 is NOT equal to "abc123" This will also print lines which have less than 5 fields.

```
awk '$5 != "abc123"'
```

```
awk '!($5 == "abc123")'
```

Matching a field against a regular expression

```
awk '$7  ~ /^[a-f]/'    # print line if field #7 matches regex
```

```
awk '$7 !~ /^[a-f]/'    # print line if field #7 does NOT match regex
```

Print the line immediately before a regex, but not the line Containing the regex

```
awk '/regex/{print x};{x=$0}'
```

```
awk '/regex/{print (NR==1 ? "match on line 1" : x)};{x=$0}'
```

Print the line immediately after a regex, but not the line Containing the regex

```
awk '/regex/{getline;print}'
```

Grep for AAA and BBB and CCC (in any order on the same line)

```
awk '/AAA/ && /BBB/ && /CCC/'
```

Grep for AAA and BBB and CCC (in that order)

```
awk '/AAA.*BBB.*CCC/'
```

Print only lines of 65 characters or longer

```
awk 'length > 64'
```

Print only lines of less than 65 characters

```
awk 'length < 64'
```

Print section of file from regular expression to end of file

```
awk '/regex/,0'
```

```
awk '/regex/,EOF'
```

Print section of file based on line numbers (lines 8-12, inclusive)

```
awk 'NR==8,NR==12'
```

Print line number 52

```
awk 'NR==52'
```

```
awk 'NR==52 {print;exit}'          # more efficient on large files
```

Print section of file between two regular expressions (inclusive)

```
awk '/Iowa/,/Montana/'             # case sensitive
```

Selective Deletion Of Certain Lines

Delete ALL blank lines from a file (same as "grep '.' ")

```
awk NF
```

```
awk '/./'
```

Remove duplicate, consecutive lines (emulates "uniq")

```
awk 'a !~ $0; {a=$0}'
```

Remove duplicate, nonconsecutive lines

```
awk '!a[$0]++'                    # most concise script
```

```
awk '!($0 in a){a[$0];print}'     # most efficient script
```


Pipe Awk Output To The Shell

This technique allows each line generated by an awk script to be executed by the shell

Move files to the "iraf" folder and add .dat to the names

```
ls junk* | awk '{print "mv "$0" ../iraf/"$0".dat"}' | sh
```


More One Line Examples

Print first two fields in opposite order

```
awk '{ print $2, $1 }' file
```

Print lines longer than 72 characters

```
awk 'length > 72' file
```

Print length of string in 2nd column

```
awk '{print length($2)}' file
```

Add up first column, print sum and average

```
{ s += $1 }
```

```
END  { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order

```
awk '{ for (i = NF; i > 0; --i) print $i }' file
```

Print the last line

```
{line = $0}
```

```
END {print line}
```

Print the total number of lines that contain the word Pat

```
/Pat/ {nlines = nlines + 1}
```

```
END {print nlines}
```

Print all lines between start/stop pairs

```
awk '/start/, /stop/' file
```

Print all lines whose first field is different from previous one

```
awk '$1 != prev { print; prev = $1 }' file
```

Print column 3 if column 1 > column 2

```
awk '$1 > $2 {print $3}' file
```

Print line if column 3 > column 2

```
awk '$3 > $2' file
```

Count number of lines where col 3 > col 1

```
awk '$3 > $1 {print i + "1"; i++}' file
```

Print sequence number and then column 1 of file

```
awk '{print NR, $1}' file
```

Print every line after erasing the 2nd field

```
awk '{$2 = ""; print}' file
```

Print hi 28 times

```
yes | head -28 | awk '{ print "hi" }'
```

Print hi.0010 to hi.0099 (NOTE IRAF USERS!)

```
yes | head -90 | awk '{printf("hi00%2.0f \n", NR+9)}'
```

Print out 4 random numbers between 0 and 1

```
yes | head -4 | awk '{print rand()}'
```

Print out 40 random integers modulo 5

```
yes | head -40 | awk '{print int(100*rand()) % 5}'
```

Replace every field by its absolute value

```
{ for (i = 1; i <= NF; i=i+1) if ($i < 0) $i = -$i print}
```

Section 22

The Field Delimiter

The field delimiter determines how awk divides up each line of the text file into 'fields' or 'columns' which can then be accessed with the $1, $2, ... variables. The delimiter can be a regular expression (unlike 'cut' for example) The default awk field delimiter is a space or a tab.

Use '—' as the field delimiter and print the 4th field

```
awk -F"|" '{print $4}' filename
```

```
awk -F'|' '{print $4}' filename            ~(the same)
```

```
awk -F\| '{print $4}' filename             ~(should work)
```

```
awk  'BEGIN {FS="|"} {print $4}' filename  ~(the same)
```

Set the field delimiter to be a comma followed by a space, print 2nd field

```
awk -F', ' '{print $2}' data.txt
```

Set the field delimiter to be a comma followed by any number of spaces

```
awk -F', *' '{print $2}' data.txt
```

```
awk 'BEGIN{FS=", *"}{print $2}' data.txt      ~(the same)
```

```
awk 'BEGIN{FS=", *";};{print $2;}' data.txt   ~(the same again)
```

Set the field delimiter to be the double quote character

```
awk -F'"' '{print $2}' data.txt
```

```
awk -F\" '{print $2}' data.txt     ~(the same)
```

Set the field delimiter to be any number of '+' plus signs

```
awk -F'\+*' '{print $2}' data.txt
```

Set the field delimiter to a space following by one or more '' star signs*

```
awk -F' \*+' '{print $2}' data.txt
```

Some looping commands Remove a bunch of print jobs from the queue

```
BEGIN{
    for (i=875;i>833;i--){
              printf "lprm -Plw %d\n", i
     } exit
    }
```

example format strings for 'printf'

e.g. printf("howdy %-8s What it is bro.	%.2f\n"	$1, $2*$3)
	%s	String
	%-8s	8 character string left justified
	%.2f	Number with 2 places after .
	%6.2f	Field 6 chars with 2 chars after .
	\n	Newline
	\t	Tab

Find maximum and minimum values present in column 1

```
NR == 1 {m=$1 ; p=$1}
$1 >= m {m = $1}
$1 <= p {p = $1}
END { print "Max = " m, "   Min = " p }
```

Example of defining variables, multiple commands on one line

```
NR == 1 {prev=$4; preva = $1; prevb = $2; n=0; sum=0}
$4 != prev {print preva, prevb, prev, sum/n; n=0; sum=0; prev = $4;
 ⇒ preva = $1;
 prevb = $2}
$4 == prev {n++; sum=sum+$5/$6}
END {print preva, prevb, prev, sum/n}
```

Example of defining and using a function, inserting values into an array And doing integer arithmetic mod(n). This script finds the number of days Elapsed since Jan 1, 1901. (from http://www.netlib.org/research/awkbookcode/

```
h3)
function daynum(y, m, d,    days, i, n)
{   # 1 == Jan 1, 1901
    split("31 28 31 30 31 30 31 31 30 31 30 31", days)
    # 365 days a year, plus one for each leap year
    n = (y-1901) * 365 + int((y-1901)/4)
    if (y % 4 == 0) # leap year from 1901 to 2099
        days[2]++
    for (i = 1; i < m; i++)
        n += days[i]
    return n + d
}
    { print daynum($1, $2, $3) }
```

Example of using substrings Substr($2,9,7) picks out characters 9 thru 15 of column 2

```
{print "imarith", substr($2,1,7) " - " $3, "out."substr($2,5,3)}
{print "imarith", substr($2,9,7) " - " $3, "out."substr($2,13,3)}
{print "imarith", substr($2,17,7) " - " $3, "out."substr($2,21,3)}
{print "imarith", substr($2,25,7) " - " $3, "out."substr($2,29,3)}
```

Section 23

Books About Awk

"sed & awk, 2nd Edition," by Dale Dougherty and Arnold Robbins (O'Reilly, 1997)
"UNIX Text Processing," by Dale Dougherty and Tim O'Reilly (Hayden Books, 1987)
"GAWK: Effective awk Programming," 3d edition, by Arnold D. Robbins (O'Reilly, 2003) or at http://www.gnu.org/
"Mastering Regular Expressions, 3d edition" by Jeffrey Friedl (O'Reilly, 2006).
The info and manual ("man") pages on Unix systems may be helpful (try "man awk", "man nawk", "man gawk", "man regexp", or the section on regular expressions in "man ed").

Section 24

Awk Contributors

Peter S. Tillier (U.K.); Daniel Jana; Yisu Dong

Section 25

Notes

Convert numbers to SI notation

```
$ awk '{ split(sprintf("%1.3e", $1), b, "e"); p = substr("
  ⇒ yzafpnum_kMGTPEZY", (b[2]/3)+9, 1); o = sprintf("%f", b[1] * (10 ^
  ⇒ (b[2]%3))); gsub(/\./, p, o); print substr( gensub(/_[[:digit:]]*/,
  ⇒  "", "g", o), 1, 4); }' < test.dat
```