```
HANDY ONE-LINE SCRIPTS FOR AWK                              30 April 2008
Compiled by Eric Pement - eric [at] pement.org              version 0.27

Latest version of this file (in English) is usually at:
   http://www.pement.org/awk/awk1line.txt

This file will also be available in other languages:
   Chinese  - http://ximix.org/translation/awk1line_zh-CN.txt

USAGE:

   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

Note that the DJGPP compilation (for DOS or Windows-32) permits an awk
script to follow Unix quoting syntax '/like/ {"this"}'. HOWEVER, if the
command interpreter is CMD.EXE or COMMAND.COM, single quotes will not
protect the redirection arrows (<, >) nor do they protect pipes (|).
These are special symbols which require "double quotes" to protect them
from interpretation as operating system directives. If the command
interpreter is bash, ksh or another Unix shell, then single and double
quotes will follow the standard Unix usage.

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.

If a script will not need to be quoted in Unix, DOS, or CMD, then I
normally omit the quote marks. If an example is peculiar to GNU awk,
the command 'gawk' will be used. Please notify me if you find errors or
new commands to add to this list (total length under 65 characters). I
usually try to put the shortest script first. To conserve space, I
normally use '1' instead of '{print}' to print each line. Either one
will work.

FILE SPACING:

 # double space a file
 awk '1;{print ""}'
 awk 'BEGIN{ORS="\n\n"};1'

 # 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"}'

NUMBERING AND CALCULATIONS:

 # 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}'

  # 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'

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'

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

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

# 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

# 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
```

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
```

CREDITS AND THANKS:

Special thanks to the late Peter S. Tillier (U.K.) for helping me with the first release of this FAQ file, and to Daniel Jana, Yisu Dong, and others for their suggestions and corrections.

For additional syntax instructions, including the way to apply editing commands from a disk file instead of the command line, consult:

"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/software/gawk/manual/

To fully exploit the power of awk, one must understand "regular expressions." For detailed discussion of regular expressions, see "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").

USE OF '\t' IN awk SCRIPTS: For clarity in documentation, I have used '\t' to indicate a tab character (0x09) in the scripts. All versions of awk should recognize this abbreviation.

#---end of file---