Command	Short Description
awk '*cmds*' **file(s)**	Invokes the awk commands (*cmds*) on the file or files (**file(s)**)
$1 $2 $3...	Denotes the first, second, third, and so on fields respectively in a file
$0	Denotes an entire line in a field
{.....}	Whatever is inside these brackets is treated as an executable step (i.e., print, x=3, n=5+$32, getline).
{print...}	Prints whatever is designated to the screen unless the output is redirected to a file
(...)	Whatever is inside these brackets is used to test for patterns (if--then...else, while, etc.)
awk -f *prog inputf*	If the awk command line is very long, it may be placed in a program file (*prog*), and the input file(s) is shown as *inputf*.
NF	Awk automatically counts the fields for each input line and gives the variable NF that value.
{printf(...)}	Prints using a user-supplied format
BEGIN{...}	Executes whatever is inside the brackets before starting to view the input file
END{...}	Executes whatever is inside the brackets after awk is finished reading the input file
length(*field*)	Counts the number of characters in a word or field (i.e., $5 or even $0)
#	Used to comment out statements in an awk program file
array[*countr*]	An array with the counting variable *countr* (note this didn't have to be predefined!)
/*string*/	Matches the current input line for *string*
~/*string*/	Matches current input line for *string* by itself or as a substring
!~ /*string*/	Matches current input line for anything *not* containing *string*

Control Flow Statements

Command	Short Description
{*statements*}	Execute all the *statements* grouped in the brackets
if (*expression*) *statement*	If *expression* is true, execute *statement.*
if (*expression*) *statement1* **else** *statement2*	If *expression* is true, execute *statement1;* otherwise, execute *statement2.*
while (*expression*) *statement*	If *expression* is true, execute *statement* and repeat.
for (*expression1*; *expression2*; *expression3*) *statement*	Equivalent to *expression1*; while (*expression2*) {*statement*; *expression3*}
for (*variable in array*) *statement*	Execute *statement* with *variable* set to each subscript in *array* in turn
do *statement* while (*expression*)	Execute *statement*; if *expression* is true, repeat
break	Immediately leave innermost enclosing *while, for,* or *do*
continue	Start next iteration of innermost enclosing *while, for,* or *do*
next	Start next iteration of main input loop
exit	Exit
exit *expression*	Go immediately to the END action; if within the END action, exit program entirely.

Expression	Meaning
Metacharacters	
\	Used in an escape sequence to match a special symbol (e.g., \t matches a tab and * matches * literally)
^	Matches the beginning of a string
$	Matches the end of a string
.	Matches any single character
[ABDU]	Matches either character A, B, D, or U; may include ranges like [a-e-B-R]
A|B	Matches A or B
DF	Matches D immediately followed by an F
R*	Matches zero or more Rs
R+	Matches one or more Rs
R?	Matches a null string or R
NR==10, NR==25	Matches all lines from the 10th read to the 25th read

Escape Sequences	Meaning
\b	Backspace
\f	Form feed
\n	Newline (line feed)
\r	Carriage return
\t	Tab
ddd	Octal value *ddd*, where *ddd* is 1 to 3 digits between 0 and 7
\c	Any other character literally (e.g., \\ for backslash, \" for ", * for *, and so on)

Operator	Meaning
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to
>=	Greater than or equal to
>	Greater than
~	Matched by (used when comparing strings)
!~	Not matched by (used when comparing strings)

Built-In Variables

Variable	Meaning	Default
ARGC	Number of command line arguments	–
ARGV	Array of command line arguments	–
FILENAME	Name of current input file	–
FNR	Record number in current file	–
FS	Controls the input field separator	one space
NF	Number of fields in current record	–
NR	Number of records read so far	–
OFMT	Output format for numbers	%.6g
OFS	Output field separator	one space
ORS	Output record separator	\n
RLENGTH	Length of string matched by match function	–
RS	Controls the input record separator	\n
RSTART	Start of string matched by match function	–
SUBSEP	Subscript separator	\034

Built-In String Functions

Function	Description
r	Represents a regular expression
s and t	Represent string expressions
n and p	Integers

gsub(*r*,*s*)	Substitute *s* for *r* globally in $0; return number of substitutions made
gsub(*r*,*s*,*t*)	Substitute *s* for *r* globally in string *t*; return number of substitutions made
index(*s*,*t*)	Return first position of string *t* in *s*, or 0 if *t* is not present
length(*s*)	Return number of characters in *s*
match(*s*,*r*)	Test whether s contains a substring matched by r; return index or 0; sets RSTART and RLENGTH
split(*s*,*a*)	Split *s* into array *a* on FS; return number of fields
split(*s*,*a*,*fs*)	Split *s* into array *a* on field separator *fs*; return number of fields
sprintf(*fmt*,*expr-list*)	Return *expr-list* formatted according to format string *fmt*
sub(*r*,*s*)	Substitute *s* for the leftmost longest substring of $0 matched by *r*; return # of subs made
sub(*r*,*s*,*t*)	Substitute *s* for the leftmost longest substring of *t* matched by *r*; return # of subs made
substr(*s*,*p*)	Return suffix of *s* starting at position *p*
substr(*s*,*p*,*n*)	Return substring of *s* of length *n* starting at position *p*

Expression Operators

Operation	Operators	Example	Meaning of Example
assignment	= += -= *= /= %= ^=	x = x * 2	x = x * 2
conditional	?:	x?y:z	If x is true, then y; else z
logical OR	||	x || y	1 if x or y is true; 0 otherwise
logical AND	&&	x && y	1 if x and y are true; 0 otherwise
array membership	in	i in a	1 if a[i] exists; 0 otherwise
matching	~ !~	$1 ~ /x/	1 if the first field contains an x; 0 otherwise
relational	< <= > >= == !=	x == y	1 of x equals y; 0 otherwise
concatenation		“a” “bc”	“abc”; there is no explicit concatenation operator
add, subtract	+ -	x + y	Sum of x and y
multiply, divide, mod	* / %	x % y	Remainder of x is divided by y (fraction)
unary plus and minus	+ -	-x	Negative x
logical NOT	!	!$1	1 if $1 is zero or null; 0 otherwise
exponentiation	^	x ^ y	x^y
increment, decrement	++ --	++x, x++	Add 1 to x
field	$	$i + 1	Value of the ith field, plus 1
grouping	( )	($i)++	Add 1 to the value of the ith field

Output Statements

Command	Short Description
print	Print $0 to the screen.
print *expression*, *expression*, ...	Print *expression*’s, separated by OFS, terminated by ORS.
print *expression*, *expression*, ... > **filename**	Print to **filename** rather than just to the screen.
print *expression*, *expression*, ... >> **filename**	Append to the end of **filename** rather than just to the screen.
print *expression*, *expression*, ... | *command*	Print to standard input of command.
printf(*format*, *expression*, *expression*, ...)	Printf statements are just line print statements except the first argument specifies output format.
printf(*format*, *expression*, *expression*, ...) > **filename**	
printf(*format*, *expression*, *expression*, ...) >> **filename**	
printf(*format*, *expression*, *expression*, ...) | *command*	
close(*filename*), close(*command*)	Break connection between print and *filename* or *command*.
system(*command*)	Execute *command*, value is status return of command.

Printf Format Control Characters

Character	Print Expression as...
c	ASCII character
d	Decimal integer
e	[-]d.dddddd E[+-]dd
f	[-]ddd.dddddd
g	e or f conversion; whichever is shorter, with nonsignificant zeroes suppressed
o	Unsigned octal number
s	String
x	Unsigned hexadecimal number
%	Print a %; no argument is consumed

Examples of Printf

format	$1	printf(format, $1)
modifier 1: – left	Justifies expression	
modifier 2: *width*	Pads field to width as needed; leading 0 pads with zeroes	
modifier 3: *.prec*	Maximum string width, or digits to the right of the decimal point	
%c	97	a
%d	84.23	84
%5d	84.23	___84
%e	45.363	4.536300e+01
%f	36.22	36.220000
%7.2f	30.238	__30.24
%g	97.5	97.5
%.6g	6.23972482	6.239725
%o	97	141
%06o	97	000141
%x	97	61
%s	January	January
%10s	January	___January
%-10s	January	January___
%.3s	January	Jan
%10.3s	January	_____Jan
%-10.3s	January	Jan_____
%%	January	%

SAMPLES (1/2)

Format: ➨ What it does...

→ awk command

Commands:

➨ Print the total number of lines in *filename* to the screen.

→ awk 'END {print NR}' *filename*

➨ Prints the 10th input line to the screen.

→ awk 'NR == 10 {print}' *filename*

➨ The print command is to print only the first field ($1) of every line found in the file *filename*.

→ awk '{print $1}' *filename*

➨ Print the last field of the last input line to the screen.

→ awk '{field=$NR} END {print field}' *filename*

➨ Print all input lines ($0) from *filename* that have more than 4 fields (NF>4).

→ awk 'NF > 4 {print $0}' *filename*

➨ Print the values in the first ($1), fourth ($4), and third ($3) fields from every line in the file *filename* in the listed order to the screen separated by the output field separator (OFS) which is one space by default.

→ awk '{print $1, $4, $3}' *filename*

➨ This searches for fields that start (^) with MDATA (~/MDATA/) in the first field ($1). For every match, it increments linesdata by one (++linesdata). After the entire *filename* has been read, the program prints to the screen the number of lines that met the criteria along with a little sentence quoting the name of the input file ($FILENAME).

→ awk 'BEGIN {linesdata=0} $1 ~/^MDATA/ {++linesdata} END {print linesdata " Lines \
start with MDATA in the first field from " $FILENAME}' *filename*

➨ IF the value in the first field in *filename* is equal to 0, THEN the entire line ($0) will be printed to the screen.

→ awk '($1 == 0) {print $0}' *filename*

➨ This will find the largest time value in the first field in the entire file and after finishing reading the file, it will print the maximum time found in the first field followed by the entire line from which the value came.

→ awk '($1 > timemax) {timemax = $1; maxinput = $0} \
END {print timemax, maxinput}' *filename*

SAMPLES (2/2)

- This will translate a file with: MDATA time value into a file with time value only and separated by tabs (OFS="\t")

  → awk 'while ($1~/MDATA/) {print $2,$3} BEGIN {OFS="\t"}' inputfile

- This will print the fourth field if the first field begins(^) with either "owe" or (|) "debt."

  → awk '($1~/^(owe|debt)/) {print $4}' inputfile

- This will do the same as the previous command except it will also sum the fourth field (sum=sum + $4) and print the total at the end of the list.

  → awk 'BEGIN {sum=0} \
  ($1~/^(owe|debt)/) {sum=sum + $4; print $4} \
  END {print "Your total debt is", sum}' inputfile

- This will filter lines that contain (begins (^) and ends with ($)) capital "R" followed by either a 0 or 1 then followed by a number 0 through 9. Then, these lines will be counted (++n adds 1 to n) and printed with the number order that they occurred (e.g., if there are five lines that match the expression, they will be printed in order and labeled 1 2 3 4 5 respectively).

  → awk 'BEGIN {n=0; OFS="\t"} \
  ($0~/^R[01][0-9]$/) {++n; print n, $0}' inputfile

- All output from these commands will go to the screen, but you can use UNIX redirection commands to "pipe" the output into another command or > "redirect" the output to a file or even >> "append" the output to the and of a pre-existing file.

- Awk is also very useful when you need to put information in a different format for a script. Just include the formatting awk statement in your script.