**AWK one-liner collection**

I love perl and I use it for most scripts but nothing beats awk on the commandline. AWK is a pattern matching and string processing language named after the surnames of the original authors: Alfred Aho, Peter Weinberger and Brian Kernighan.

**Print selected fields (at a fixed position)**

Split up the lines of the file file.txt with ":" (colon) separated fields and print the second field ($2) of each line:

awk -F":" '{print $2}' file.txt

Same as above but print only output if the second field ($2) exists and is not empty:

awk -F":" '{if ($2)print $2}' file.txt

Print selected fields from each line separated by a dash:

awk -F: '{ print $1 "-" $4 "-" $6 }' file.txt

Print the last field in each line:

awk -F: '{ print $NF }' file.txt

Print every line and delete the second field:

awk '{ $2 = ""; print }' file.txt

Good to know:

* The commandline option -F sets the field separator. The default is space.
* $0 the entire line without the newline at the end
* $1 to $9, $10 to ..., the fields
* NF number of fields
* NR currant line number (counting across all files for multiple files)
* FNR line number (just for that file)

**Print selected fields (at a variable position)**

Sometimes you have delimiter separated fields but the number of fields changes. You want to print a certain field based on the content of a previous field. The linux command "ip route get DstIP" can be used to show the computers own source IP that it uses to go to a certain destination. This printout may take different forms but we are looking for the IP address after the word "src":

# ip route get 1.1.1.1 1.1.1.1 via 10.8.0.2 dev wls1 src 10.8.0.3 cache The printout could also look like this: # ip route get 1.1.1.1 1.1.1.1 dev wls1 src 10.8.0.3 cache

Print the content of a field **after** the field that contains the string "src":

ip route get 1.1.1.1 | awk '{for(i=1;i<=NF;i++){if($i=="src") print $(i+1)}}'

This will always print the computers "own IP" used for the communication with the public IP 1.1.1.1 (it will return 10.8.0.3 for the above examples).

**Print matching lines**

Print field number two ($2) only on lines matching "some regexp" (fiel separator is ":"):

awk -F":" '/some regexp/{print $2}' file.txt

Print lines matching "regexp a" and lines matching "regexp b" but the later ones are printed without newline (note the printf):

awk '/regexp a/{print};/regexp b/{printf $0}' file.txt

Print field number two ($2) only on lines **not** matching "some regexp" (fiel separator is ":"):

awk -F":" '!/some regexp/{print $2}' file.txt or awk -F":" '/some regexp/{next;}{print $2}' file.txt

Print field number two ($2) only on lines matching "some regexp" otherwise print field number three ($3) (fiel separator is ":"):

awk -F":" '/some regexp/{print $2;next}{print $3}' file.txt

The "next" command causes awk to continue with the next line and execute "{print $3}" only for non matching lines. This is like   
/regexp/{...if..regexp..matches...;next}{...else...}   
  
Print lines where field number two matches regexp (apply regexp only to field 2, not the whole line):

awk '$2 ~ /regexp/{print;}' file.txt

Here is an example parsing the linux "ps aux" command. It has in the eighth column the process state. To print all processes that are in running or runnable state you would look for the letter "R" in that 8-th column. You want as well to print line 1 of the ps command printout since it contains the column header:

ps aux | awk '$8 ~ /R/{print;}NR==1{print}'

Print the next two (i=2) lines after the line matching regexp:

awk '/regexp/{i=2;next;}{if(i){i--; print;}}' file.txt

Print the line and the next two (i=2) lines after the line matching regexp (this command is the same as: grep -A 2 regexp):

awk '/regexp/{i=2+1;}{if(i){i--; print;}}' file.txt

Print the line matching regexp and 12 following lines. Print also any line matching regexp2:

awk '/regexp/{i=12+1}{if(i){i--; print;}}/regexp2/{print}' file.txt

AWK ranges: Print the lines from a file starting at the line matching "start" until the line matching "stop":

awk '/start/,/stop/' file.txt

Note: make sure that the stop pattern does not match the start line otherwise only that line will be printed.   
  
AWK ranges: Print the lines starting at the line matching "start" until the end of the file:

awk '/start/,0' file.txt

Note: make sure that the stop pattern does not match the start line otherwise only that line will be printed.   
  
Sometimes you have a terminal log and it contains "prompt# command" with printouts of "command" in-between. You can't use: awk '/command/,/prompt/' log.txt   
to print everything from command to the next prompt because the line where the command is has also a prompt. To solve this we can use a state variable and the "next" statement to skip the processing of other statements once we found "command". This will print everything from "prompt.\*ls" and stop printing at the next prompt:

awk '/prompt.\*ls/{s=1;print;next};/prompt/{s=0};s==1{print}' log.txt

Change "prompt" to whatever string appears in your terminal prompt, e.g the hostname.   
  
If you want to include the last prompt where to stop printing then try this:

awk '/prompt.\*ls/{s=1;print;next};s==0{next};{print};/prompt/{s=0};' log.txt

Change "prompt" to whatever string appears in your terminal prompt, e.g the hostname.   
  
Print fields 1 and 2 from all lines not matching regexp:

awk '!/regexp/{print $1 " " $2 }' file.txt

Print fields 1 and 2 from lines matching regexp1 and not matching regexp2:

awk '/regexp1/&&!/regexp2/{print $1 " " $2 }' file.txt

Regexp syntax:

* c matches the non-metacharacter c.
* \c matches the literal character c.
* . matches any character including newline.
* ^ matches the beginning of a string (example: ^1 , only lines starting with a one)
* $ matches the end of a string (example: end$ , only lines ending in "end")
* [abc...] character list, matches any of the characters abc....
* [0-9a-zA-Z] range of characters 0-9 and a-z,A-Z
* [^abc...] negated character list, matches any character except abc....
* r1|r2 alternation: matches either r1 or r2.
* r1r2 concatenation: matches r1, and then r2.
* r+ matches one or more r's.
* r\* matches zero or more r's.
* r? matches zero or one r's.
* (r) grouping: matches r.

In languages like Perl you can use the grouping feature to extract a substring from the matching string. Normal AWK can not use a grouping to chapture a string. However gawk has the match function which can be used for that. The string matched by the first bracket will be in arr[1].   
Print the content of the part of the matching regexp that is enclosed by the round brackets:

gawk 'match($0, /length:([0-9]+) cm/,arr){ print arr[1]}' file.txt

If file.txt looks as shown below then the above command would print 12:

width:3 cm

length:12 cm

height:14 cm

**Insert a string after the matching line**

This inserts a new line after the matching line:

awk '/regexp/{print $0; print "text inserted after matching line";next}{print}' file.txt

$0 is the line where the search pattern "regexp" matches without the newline at the end. The awk print command prints the string and appends a new line.   
  
This appends a string to the matching line:

awk '/regexp/{print $0 "text appended at end of the matching line";next}{print}' file.txt

**If matching "do A" else "do B" (if .. then .. else in awk)**

awk '/regexp/{A-here;next}{B-here}' file.txt Example: awk '/regexp/{gsub(/string/,"replacement");print $1;next}{print;}' file.txt

The example would print lines that do not match unchanged (action B is just "print;") while on lines that match /regexp/ it would replace /string/ by replacement and print the first element ($1).

**If matching "A do..." OR if matching "B do.." (if .. then, if .. then, ...., in awk)**

awk '/regexpA/{A-do-here;}/regexpB/{B-do-here}' file.txt Example: awk '/house/{print $1;}/cat/{print;}' file.txt

**Replacement for some common unix commands (useful in a non unix environment)**

Count lines (wc -l):

awk 'END{print NR}'

Search for matching lines (egrep regexp):

awk '/regexp/'

Print non matching lines (egrep -v regexp):

awk '!/regexp/'

Print matching lines with numbers (egrep -n regexp):

awk '/regexp/{print FNR,$0}'

Print matching lines and ignore case (egrep -i regexp):

awk 'BEGIN {IGNORECASE=1};/regexp/'

Number lines (cat -n):

awk '{print FNR "\t" $0}'

Remove duplicate consecutive lines (uniq):

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

Print first 5 lines of file (head -5):

awk 'NR < 6'

Print everything after the first 2 lines (do not print first 2 lines):

awk 'NR > 2'

**Number non empty lines**

This prints all lines and adds a line number to non empty lines:

awk '/^..\*$/{ print FNR ":" $0 ;next}{print}' file.txt

**Remove empty lines**

This prints all lines except empty ones and lines with only space and tab:

awk '/^[ \t]\*$/{next}{print}' file.txt

**Number lines longer than 80 char and show them**

This is useful to find all the lines longer than 80 characters (or any other length):

awk 'length($0)>80{print FNR,$0}' file.txt

**Substitute foo for bar on lines matching regexp**

awk '/regexp/{gsub(/foo/, "bar")};{print}' file.txt

**Delete trailing white space (spaces, tabs)**

awk '{sub(/[ \t]\*$/, "");print}' file.txt

**Delete leading white space**

awk '{sub(/^[ \t]+/, ""); print}' file.txt

**Add some characters at the beginning of matching lines**

Add ++++ at lines matching regexp.

awk '/regexp/{sub(/^/, "++++"); print;next;}{print}' file.txt

**Color gcc warnings in red**

gcc -Wall main.c |& awk '/: warning:/{print "\x1B[01;31m" $0 "\x1B[m";next;}{print}'

The "\x1B" means the ascii character with hex number 1B (ESC).

**Print only lines of less than 80 characters**

awk 'length < 80' file.txt

**Renaming files with AWK**

You can use awk to generate shell commands such as e.g mv commands to rename files according to a given recipe. I suggest to always print the commands before piping them to sh in order to execute them. A small typo can have very significant side effect so double check what would happen by printing the commands first.   
  
Rename all .MP3 file to be lower case:

ls \*.MP3 | awk '{ printf("mv \"%s\" \"%s\"\n", $0, tolower($0)) }' The above will just print what would happen. To actually execute it you run: ls \*.MP3 | awk '{ printf("mv \"%s\" \"%s\"\n", $0, tolower($0)) }' | sh

Substitute a regexp pattern with a given replacement string. We can e.g replace " " (spaces in the file names) by "-":

ls | awk '{ printf("mv \"%s\" \"%s\"\n", $0, gensub(/ +/,"-","g")) }' The above will just print what would happen. To actually execute it you run: ls | awk '{ printf("mv \"%s\" \"%s\"\n", $0, gensub(/ +/,"-","g")) }' | sh

The gensub function reads the strings from $0 (=current line) and returns the modified string. The third argument, the "g", means to find and replace everywhere (globally) on the current line.

**AWK as a command-line calculator**

This prints 5.1:

awk 'BEGIN{print 3.1+4/2}'

This prints 1.41421:

awk 'BEGIN{print sqrt(2)}'

This prints 1.41421:

awk 'BEGIN{print 2^(1/2)}'

This prints 3.141592653589793 (PI with a 15 digits behind the decimal point):

awk 'BEGIN{printf "%.15f\n",4\*atan2(1,1)}'

Print decimal number as hex (this prints 0x20):

awk 'BEGIN{printf "0x%x\n", 32}'

Convert hex string to decimal (this prints 32):

awk 'BEGIN{print strtonum(0x20)}'

Math operators in gnu awk:

* + - \* /
* ^ or \*\* Exponentiation
* % Modulo
* exp(), log() Exponential function and natural logarithm
* atan2(y, x), sin(), cos() work all in radians (fraction of PI)
* sqrt() same as \*\*(1/2) Square root
* strtonum() Convert hex (start with 0x) and octal (start with 0) to decimal

If you want to use this frequently then you could put this into your .bashrc file:

# add the awc function to .basrc # use awc like this: awc "3.4+2+8+99.2" (do not forget the quotes) awc(){ awk "BEGIN{ print $\* }" ;}

On the shell you can then type awc "3.4+2+8+99.2" and it will print 112.6.

**AWK minimal web server**

You can't write a web server as a reasonable one-liner in AWK, you can do that with netcat but there are some cases where you don't have a real web server and you don't have netcat but you have a very basic shell environment and that does usally include gawk (note: you need gawk, gnu version of awk). Here is a web server that allows you to serve files at port 8080 (or any port, just change the number):

#!/usr/bin/gawk -f

BEGIN {

if (ARGC < 2) { print "Usage: wwwawk file.html"; exit 0 }

Concnt = 1;

while (1) {

RS = ORS = "\r\n";

HttpService = "/inet/tcp/8080/0/0";

getline Dat < ARGV[1];

Datlen = length(Dat) + length(ORS);

while (HttpService |& getline ){

if (ERRNO) { print "Connection error: " ERRNO; exit 1}

print "client: " $0;

if ( length($0) < 1 ) break;

}

print "HTTP/1.1 200 OK" |& HttpService;

print "Content-Type: text/html" |& HttpService;

print "Server: wwwawk/1.0" |& HttpService;

print "Connection: close" |& HttpService;

print "Content-Length: " Datlen ORS |& HttpService;

print Dat |& HttpService;

close(HttpService);

print "OK: served file " ARGV[1] ", count " Concnt;

Concnt++;

}

}

Copy this code and save it into a file called wwwawk and then make it executable with "chmod 755 wwwawk". Now take some file (e.g somefile.html) and you can serve it via that little web server:

chmod 755 wwwawk ./wwwawk somefile.html from another shell: curl http://localhost:8080 or lynx http://localhost:8080 or firefox -new-tab http://localhost:8080

This is e.g. a great way to serve kickstart files for automated Linux installations. Note that this awk web server requires gawk. Most linux distributions use gawk by default except for raspberry pi which uses mawk and mawk does not support network connections.

**Perl minimal web server, a bit easier to use**

While it's cool to use AWK as a web server it can get get quickly complicated if you try to improve and enhance the above web server. We are reaching the limits of this language. Perl is in many areas pretty similar to AWK but more suitable for advanced uses. Try this little perl based web server if you are looking for a simple single page web server that is easy to change and enhance:

* [wwwperl.txt](http://tuxgraphics.org/~guido/scripts/wwwperl.txt)

**References, other nice pages about AWK**

* The GAWK manual: <http://www.gnu.org/software/gawk/manual/gawk.html>
* AWK wikibooks: <http://en.wikibooks.org/wiki/AWK>
* Examples with awk, a short introduction: <http://www.linuxfocus.org/English/September1999/article103.html>
* AWK 1 liners: <http://www.pement.org/awk/awk1line.txt>, see as well <http://www.pement.org/awk.htm>

© Guido Socher, guidosocher@fastmail.fm   
License: CC BY 4.0, http://creativecommons.org/licenses/by/4.0/ww