Follow me on Twitter for my latest adventures!

Awk Programming **64** Comments January 05, 2009

Awk One-Liners Explained, Part III: Selective Printing and Deleting of Certain Lines

This is the third and final part of a three-part article on the **Awk one-liners**. This part will explain Awk one-liners for **selective printing and deletion of certain lines**. See part one for introduction of the series.

If you just came to my website, then you might wonder, "What are these Awk one-liners and why are they famous?" The answer is very simple - they are small and beautiful Awk programs that do one and only text manipulation task very well. They have been circulating around the Internet as **awk1line.txt** text file and they have been written by Eric Pement.

If you are intrigued by this article series, I suggest that you subscribe to my posts, as I will have a lot more interesting and educational articles this year.

Eric Pement's Awk one-liner collection consists of five sections:

- 1. File spacing (explained in part one).
- 2. Numbering and calculations (explained in part one).
- 3. Text conversion and substitution (explained in part two).
- **4. Selective printing of certain lines (explained in this part).**
- **5. Selective deletion of certain lines (explained in this part).**
- 6. String creation, array creation and update on selective printing of certain lines (explained in bonus article).
- 7. Release of Awk One-Liners Explained e-book.

Awesome news: I have written an e-book based on this article series. Check it out:

Grab my Awk cheat sheet and the local copy of Awk one-liners file awk1line.txt and let's roll.

4. Selective Printing of Certain Lines

45. Print the first 10 lines of a file (emulates "head -10").

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

Awk has a special variable called "NR" that stands for "Number of Lines seen so far in the current file". After reading each line, Awk increments this variable by one. So for the first line it's 1, for the second line 2, ..., etc. As I explained in the very first one-liner, every Awk program consists of a sequence of pattern-action statements "pattern { action statements }". The "action statements" part get executed only on those lines that match "pattern" (pattern evaluates to true). In this one-liner the pattern is "NR < 11" and there are no "action statements". The default action in case of missing "action statements" is to print the line as-is (it's equivalent to "{ print $0 }"). The pattern in this one-liner is an expression that tests if the current line number is less than 11. If the line number is less than 11, Awk prints the line. As soon as the line number is 11 or more, the pattern evaluates to false and Awk skips the line.

A much better way to do the same is to quit after seeing the first 10 lines (otherwise we are looping over lines > 10 and doing nothing):

```
awk '1; NR == 10 { exit }'
```

The "NR == 10 { exit }" part guarantees that as soon as the line number 10 is reached, Awk quits. For lines smaller than 10, Awk evaluates "1" that is always a true-statement. And as we just learned, true statements without the "action statements" part are equal to "{ print $0 }" that just prints the first ten lines!

46. Print the first line of a file (emulates "head -1").

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

This one-liner is very similar to previous one. The "NR > 1" is true only for lines greater than one, so it does not get executed on the first line. On the first line only the "1", the true statement, gets executed. It makes Awk print the line and read the next line. Now the "NR" variable is 2, and "NR > 1" is true. At this moment "{ exit }" gets executed and Awk quits. That's it. Awk printed just the first line of the file.

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

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

Okay, so what does this one do? First of all, notice that "{y=x "\n" $0; x=$0}" action statement group is missing the pattern. When the pattern is missing, Awk executes the statement group for all lines. For the first line, it sets variable "y" to "\nline1" (because x is not yet defined). For the second line it sets variable "y" to "line1\nline2". For the third line it sets variable "y" to "line2\nline3". As you can see, for line N it sets the variable "y" to "lineN-1\nlineN". Finally, when it reaches EOF, variable "y" contains the last two lines and they get printed via "print y" statement.

Thinking about this one-liner for a second one concludes that it is very ineffective - it reads the whole file line by line just to print out the last two lines! Unfortunately there is no seek() statement in Awk, so you can't seek to the end-2 lines in the file (that's what tail does). It's recommended to use "tail -2" to print the last 2 lines of a file.

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

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

This one-liner may or may not work. It relies on an assumption that the "$0" variable that contains the entire line does not get reset after the input has been exhausted. The special "END" pattern gets executed after the input has been exhausted (or "exit" called). In this one-liner the "print" statement is supposed to print "$0" at EOF, which may or may not have been reset.

It depends on your awk program's version and implementation, if it will work. Works with GNU Awk for example, but doesn't seem to work with nawk or xpg4/bin/awk.

The most compatible way to print the last line is:

```
awk '{ rec=$0 } END{ print rec }'
```

Just like the previous one-liner, it's computationally expensive to print the last line of the file this way, and "tail -1" should be the preferred way.

49. Print only the lines that match a regular expression "/regex/" (emulates "grep").

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

This one-liner uses a regular expression "/regex/" as a pattern. If the current line matches the regex, it evaluates to true, and Awk prints the line (remember that missing action statement is equal to "{ print }" that prints the whole line).

50. Print only the lines that do not match a regular expression "/regex/" (emulates "grep -v").

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

Pattern matching expressions can be negated by appending "!" in front of them. If they were to evaluate to true, appending "!" in front makes them evaluate to false, and the other way around. This one-liner inverts the regex match of the previous (#49) one-liner and prints all the lines that do not match the regular expression "/regex/".

51. Print the line immediately before a line that matches "/regex/" (but not the line that matches itself).

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

This one-liner always saves the current line in the variable "x". When it reads in the next line, the previous line is still available in the "x" variable. If that line matches "/regex/", it prints out the variable x, and as a result, the previous line gets printed.

It does not work, if the first line of the file matches "/regex/", in that case, we might want to print "match on line 1", for example:

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

This one-liner tests if variable "x" contains something. The only time that x is empty is at very first line. In that case "match on line 1" gets printed. Otherwise variable "x" gets printed (that as we found out contains the previous line). Notice that this one-liner uses a ternary operator "foo?bar:baz" that is short for "if foo, then bar, else baz".

52. Print the line immediately after a line that matches "/regex/" (but not the line that matches itself).

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

This one-liner calls the "getline" function on all the lines that match "/regex/". This function sets $0 to the next line (and also updates NF, NR, FNR variables). The "print" statement then prints this next line. As a result, only the line after a line matching "/regex/" gets printed.

If it is the last line that matches "/regex/", then "getline" actually returns error and does not set $0. In this case the last line gets printed itself.

53. Print lines that match any of "AAA" or "BBB", or "CCC".

```
awk '/AAA|BBB|CCC/'
```

This one-liner uses a feature of extended regular expressions that support the | or alternation meta-character. This meta-character separates "AAA" from "BBB", and from "CCC", and tries to match them separately on each line. Only the lines that contain one (or more) of them get matched and printed.

54. Print lines that contain "AAA" and "BBB", and "CCC" in this order.

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

This one-liner uses a regular expression "AAA.*BBB.*CCC" to print lines. This regular expression says, "match lines containing AAA followed by any text, followed by BBB, followed by any text, followed by CCC in this order!" If a line matches, it gets printed.

55. Print only the lines that are 65 characters in length or longer.

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

This one-liner uses the "length" function. This function is defined as "length([str])" - it returns the length of the string "str". If none is given, it returns the length of the string in variable $0. For historical reasons, parenthesis () at the end of "length" can be omitted. This one-liner tests if the current line is longer than 64 chars, if it is, the "length > 64" evaluates to true and line gets printed.

56. Print only the lines that are less than 64 characters in length.

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

This one-liner is almost byte-by-byte equivalent to the previous one. Here it tests if the length if line less than 64 characters. If it is, Awk prints it out. Otherwise nothing gets printed.

57. Print a section of file from regular expression to end of file.

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

This one-liner uses a pattern match in form 'pattern1, pattern2' that is called "range pattern". The 3rd Awk Tip from article "10 Awk Tips, Tricks and Pitfalls" explains this match very carefully. It matches all the lines starting with a line that matches "pattern1" and continuing until a line matches "pattern2" (inclusive). In this one-liner "pattern1" is a regular expression "/regex/" and "pattern2" is just 0 (false). So this one-liner prints all lines starting from a line that matches "/regex/" continuing to end-of-file (because 0 is always false, and "pattern2" never matches).

58. Print lines 8 to 12 (inclusive).

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

This one-liner also uses a range pattern in format "pattern1, pattern2". The "pattern1" here is "NR==8" and "pattern2" is "NR==12". The first pattern means "the current line is 8th" and the second pattern means "the current line is 12th". This one-liner prints lines between these two patterns.

59. Print line number 52.

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

This one-liner tests to see if current line is number 52. If it is, "NR==52" evaluates to true and the line gets implicitly printed out (patterns without statements print the line unmodified).

The correct way, though, is to quit after line 52:

```
awk 'NR==52 { print; exit }'
```

This one-liner forces Awk to quit after line number 52 is printed. It is the correct way to print line 52 because there is nothing else to be done, so why loop over the whole doing nothing.

60. Print section of a file between two regular expressions (inclusive).

```
awk '/Iowa/,/Montana/'
```

I explained what a range pattern such as "pattern1,pattern2" does in general in one-liner #57. In this one-liner "pattern1" is "/Iowa/" and "pattern2" is "/Montana/". Both of these patterns are regular expressions. This one-liner prints all the lines starting with a line that matches "Iowa" and ending with a line that matches "Montana" (inclusive).

5. Selective Deletion of Certain Lines

There is just one one-liner in this section.

61. Delete all blank lines from a file.

```
awk NF
```

This one-liner uses the special NF variable that contains number of fields on the line. For empty lines, NF is 0, that evaluates to false, and false statements do not get the line printed.

Another way to do the same is:

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

This one-liner uses a regular-expression match "." that matches any character. Empty lines do not have any characters, so it does not match.

Awk one-liners explained e-book

I have written my first e-book called "Awk One-Liners Explained". I improved the explanations of the one-liners in this article series, added new one-liners and added three new chapters - introduction to awk one-liners, summary of awk special variables and idiomatic awk. Please take a look:

Have Fun!

This concludes the article series about Awk one-liners. I hope that you enjoyed this three-part article and it made you a better Awk programmer!

My future plans are to create a **awk1line-explained.txt** that will be a supplementary file to the famous **awk1line.txt**. I am also thinking about publishing a nicely formatted pdf e-book about all the one-liners.

If you liked this article, you may also like a very similar article on Sed One-Liners Explained.

And finally, if you notice anything that you can't understand, please let me know in the comments. Thank you!

Related Posts

- Sed One-Liners Explained, Part II: Selective Printing of Certain Lines
- Update on Awk One-Liners Explained: String and Array Creation
- Perl One-Liners Explained, Part VI: Selective printing and deleting of lines
- Solving Google Treasure Hunt Puzzle 4: Prime Numbers
- Sed One-Liners Explained, Part III: Selective Deletion of Certain Lines and Special Applications
- Announcing my first e-book "Awk One-Liners Explained"

- Awk One-Liners Explained, Part I: File Spacing, Numbering and Calculations
- Perl One-Liners Explained, Part II: Line Numbering
- Bash One-Liners Explained, Part IV: Working with history
- Perl One-Liners Explained, Part I: File Spacing
-

Comments

Eric Wendelin Permalink
January 05, 2009, 17:13

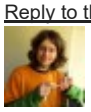
Very, very nice Peter!

Reply to this comment

Muhammad Permalink
January 05, 2009, 17:20

Cool As Usual :D...GO on man i am Periodically checking your Blog.. :)

Reply to this comment



Peter Krumins pkrumins Permalink
January 05, 2009, 17:30

Thanks Eric and Muhammad! :)

Reply to this comment

Jadu Saikia Permalink
January 06, 2009, 09:48

Nice Peter, very useful. Thanks.

One more:
Printing last field of last line:

```
awk '{ f=$NF }; END{ print f }'  file
```

Which is very useful to print the latest updated file in my dir.

```
ls -lrt | awk '{ f=$NF }; END{ print f }'
```

// Jadu

Reply to this comment

anonymous Permalink
May 03, 2013, 14:42

Or pipe to tail -1

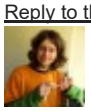
Reply to this comment

Roman Permalink
January 06, 2009, 19:22

"There *are* just one one-liner*s* in this section."

;=]

Reply to this comment



Peter Krumins pkrumins Permalink
January 07, 2009, 01:27

Thanks, Jadu - that is a nice tip!

Roman, fixing it now :) Thanks!

Reply to this comment

thedailyperlonlinr Permalink
January 07, 2009, 23:00

ah you just gave me enough material for 2 months in my new little blog ;-)

Reply to this comment

Yogesh Agrawal Permalink
February 09, 2009, 13:09

#47 can be written in easy way like this
awk ' { rec[NR]=$0} END{printf("%s\n%s\n", rec[NR-1], rec[NR]}' file

can't it?

Reply to this comment

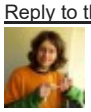
Yogesh Agrawal Permalink
February 09, 2009, 13:38

in #52 if this is the condition then it gives wrong results

foo
foo
foo
bar

According to statement it should "foo foo bar" (in different lines of course) but it prints "foo bar"

Reply to this comment



Peter Krumins pkrumins Permalink
February 11, 2009, 05:42

Hi Yogesh Agrawal. In response to your questions,

#47: No, if you do it that way, you keep all the lines in 'rec' variable. If the file is 2 gigabytes in size, you keep 2 gigabytes in memory.

#52: You're right, it should produce output like that. I did not notice it. It's 6am here. Can you help me fix it?

Reply to this comment

Yogesh Agrawal Permalink
February 11, 2009, 08:07

HI Peter,
Regarding #52 correction. I got one thing.
If our awk command is like this it will work perfectly.
awk 'var-->0;/foo/{var=1}' file

I gave input file as:

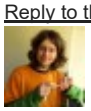
baz
foo
foo
foo
bar

foo
baz

and got output as:
foo
foo
bar
baz
which is correct as per definition.

You may find loophole in this. But till then it is working perfect.
Thanks
yogesh

Reply to this comment



Peter Krumins pkrumins Permalink
February 12, 2009, 01:02

Yogesh, I have three answers for you:

```
awk 'f;{f=/foo/}'

awk 'c&&!--c;/foo/{c=1}'

awk '/foo/{_[NR+1]}NR in _'
```

And my friends from #awk channel suggested a bunch of variations:

a) Print all records from some pattern:

```
awk '/pattern/{f=1}f' file
```

b) Print all records after some pattern:

```
awk 'f;/pattern/{f=1}' file
```

c) Print the Nth record after some pattern:

```
awk 'c&&!--c;/pattern/{c=N}' file
```

d) Print every record except the Nth record after some pattern:

```
awk 'c&&!--c{next}/pattern/{c=N}' file
```

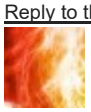
e) Print the N records after some pattern:

```
awk 'c&&c--;/pattern/{c=N}' file
```

f) Print every record except the N records after some pattern:

```
awk 'c&&c--{next}/pattern/{c=N}' file
```

Reply to this comment



Peter Passchier Permalink
February 22, 2009, 17:23

Re. oneliner 61, /./ and NF respond differently to lines that just contain spaces, /./ does pick those up.

Reply to this comment

hunter85 Permalink
June 04, 2009, 12:05

Amazing Blog.

Reply to this comment

Paul Permalink
July 29, 2009, 16:46

Serious error in #45, #46 and many others.
FNR is the line number in the *current* file.
NR is the cumulative line number across all inputs.

awk 'code' six_lines.txt ten_lines.txt will have
NR == 1 and FNR == 1
NR == 7 and FNR == 1 again
and END will see NR == 16 and FNR == 10.

Programs that check for change in FILENAME would be more efficient to check for FNR == 1.

Awk to print first 10 lines of several files:

awk 'FNR == 1 { print "\n.... " FILENAME; } FNR < 11

If your awk has "nextfile" it saves reading lines after first 10.

Reply to this comment

Raj Permalink
October 05, 2009, 09:42

Great awk coding, nice work: Peter, Thanks.

I found another code for 58:

58. Print lines 8 to 12 (inclusive).

```
awk 'NR>7 && NR<13'
```

Reply to this comment

Nik Permalink
December 31, 2009, 00:57

Nice site.

I'm stumped with this. I have a file containing
2 AAA
3 AAA
4 AAA
9 AAA
8 XXX
9 XXX
10 XXX
19 XXX
5 BBB
6 BBB
7 BBB
12 BBB
3 FFF
4 FFF
5 FFF
11 FFF

Can I use awk to return an output like
2|9|AAA
8|10|XXX
5|7|BBB
3|4|5|FFF

That is, to return the FIRST and SECOND-TO-THE-LAST numbers for each unique pattern.

Reply to this comment

Balaji Bodicherla Permalink
September 17, 2010, 02:12

I am a vivid fan of awk and sed and always wanted to visit a site who can give one-lines w/good explanation. This site is by far the best one I have seen and it really helped me to thorougly refresh my awk knowledge.

Thanks a lot, keep up the good work. As you mentioned it really helps if you can provide examples for each one-liners, looking forward to get those examples.

Thanks again

Reply to this comment

dj Permalink
February 13, 2011, 21:18

"thinking about publishing a nicely formatted pdf e-book"

A couple sed commands against the text would turn your one-liners explained into asciidoc format, which in turn can product html, pdf, docbook, ebook output, not to mention txt. PS: You can even use it for your math notes :-) See: http://www.methods.co.nz/asciidoc/

Reply to this comment

P_sun Permalink
March 23, 2011, 18:32

Hi,
If I want to get the lines 506, 590 and 600 (from text1.log) written to out.log.
The following command allows me to do that:

awk 'NR==506||NR==590 || NR==600' "text1.log" > out.txt

Let us set a=506, b=590 and c=600.
Now what I need is lines 628 (a+(122)*n), 712 (b+(122)*n),
722 (c+(122)*n) ,where n=1-20 written out from text1.log.
Can awk be suitable for this?
Can someone outline how this can be done. I appreciate your help.

If the above needs to be repeated for a series of outputs, text2.log, text3.log, how can this be done? Any help is greatly appreciated.

Reply to this comment

Dr. Stefan Gruenwald sgruenwald Permalink
March 27, 2011, 04:08

Here is the code that should work for you:

```
for (( n=1; n<21; n++ )); do awk -v n=$n
'NR==a+(122*n)||NR==b+(122*n)||NR==c+(122*n)'
a=506 b=590 c=600 text1.log >>out.txt; done
```

This should be all in one line (a true "oneliner').

Reply to this comment

paddy1 Permalink
June 23, 2011, 21:40

I have this file (using Sun OS) and need to delete rows if fields 1 and 3 are repeated

$ cat tt
yy|red|12|500|55
rr|red|12|500|55
yy|yellow|12|600|55
rr|yellow|13|600|55

Expecting output to be

rr|red|12|500|55
rr|yellow|13|600|55

And using this command - nawk '!x[$1,$3]++' FS="|" tt
But it's just cleaning up the 3rd line !!! I just picked up the command from somewhere so not sure what needs to be changed. Help is appreciated :)

Reply to this comment

yogesh Permalink
August 31, 2011, 07:54

if i have a file master.txt containing 4000 lines,
i want to cut (not print, copy) first 10 lines and paste in a file called data1.txt,
then run a command (submit a job) as
"bsub -K -qio ./runjob.txt" .
the file runjob.txt contains a single line as "./doo data1.txt ."
So, I need to delete (remove content of it) the file data1.txt, so that next 10 lines of master.txt will be in data1.txt to process.
there should be 400 iterations.
is there a way to do it in a loop using awk/sed?

Reply to this comment

Shreya Permalink
September 01, 2011, 21:17

Hey what if I need to print a bunch of lines every time after /regex/ ?

Reply to this comment

Satheesh satheesh_subran Permalink
September 22, 2011, 09:31

Thanks Peter. learned a lot.

Reply to this comment

kurtosis Permalink
March 20, 2012, 07:53

Excellent work! I started learning awk just yesterday, and this mini-series has taught me just about all I'll need to know for most tasks. Or, at least enough to keep me busy playing around for some time. I look forward to reading the perl series soon.

Just one comment: for item 47, you say it's "ineffective," but really "inefficient" is the better word -- it's effective in that it does what it's supposed to, albeit (relatively) slowly. Normally, I'd overlook such things, but since you're in the e-book business I thought I'd point it out. Thanks.

Reply to this comment

vz33 Permalink
May 15, 2012, 13:20

I have a question: if the field $2, $3.....$n can be short for , while I'd like to print the from $2 to $n (n is a very large number,so I cannot write them all one by one)

Reply to this comment

Peter Krumins pkrumins Permalink
May 16, 2012, 13:36

Use a for loop:

```
awk '{ for (i=1;i<5;i++) { field = field $i } } { print field }'
```

Reply to this comment

vapace Permalink
July 11, 2012, 04:00

In addition to Peter's for loop comment, the n in your question is called NF in awk.

Reply to this comment

wb Permalink
May 22, 2012, 01:25

hey, great resource here!! appreciate it

statement: awk '/regex/ { print $1]' /text-input-file.txt

regarding the above, is it possible for the "regex" char expression to be a var, such as $reg_ex ?

statement: awk '/$reg_ex/ { print $1]' /text-input-file.txt

i've tried to escape it and such but no luck.

thanks

Reply to this comment

ms68 Permalink
June 15, 2012, 09:48

I m new to awk and shell scripts and need to modify one of the exysting shell script. I have understand most of the script excpet for following code.

awk '/^CREATE UNIQUE INDEX/{print "-- Create Unique Index #"++x}/^CREATE TABLE/{print "-- Create Table #"++y}/^CREATE (UNIQUE INDEX|TABLE)/,/;/{print}' $ORIGINAL_SCRIPT >> $TEMP_SCRIPT

sed s/'.PSY'/'.PS_'/g < $TEMP_SCRIPT > $MODIFID_SCRIPT

Can you please explain the code by breaking everything to understand. Specialy the awk code i.e. what ^ stands for and how and what awk is doing.

Reply to this comment

Jago Permalink
June 20, 2012, 14:27

Hi :)
May be someone could help me

I have a line like this (with or without quotes) (')
02.06.1993 02:10 25'416 BUILTIN\Administrators 3RDPARTY.DOC

How could I print ONLY the files containing field #3 "bigger than a value" ? (ex bigger than 99'000'000)

TIA

Jago

Reply to this comment

Stefan Stefanov Permalink
August 10, 2012, 15:57

Thought it may be useful:

print section of a file between two regular expressions exclusive of the lines with the expressions:

awk 'NR==1,/Pattern1/{next}/Pattern2/,NR==0{next}{print}'

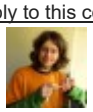
Reply to this comment

Jon marsch Permalink
November 29, 2012, 00:01

I have the following lines
Point
x
y
z
delay

a
b
delay
awk '/Point/,/delay/{print $0}' file.xt seems to include the second delay. How do i stop with the first one?. Thanks

Reply to this comment



Peter Krumins pkrumins Permalink
July 05, 2013, 18:00

This is discussed in the Awk Tips Tricks and Pitfalls article.

In general, here's how:

```
awk '/beginpat/,/endpat/{if (!/beginpat/&&!/endpat/)print}'
```

In your case:

```
awk '/Point/,/delay/{if (!/Point/&&!/delay/)print}'
```

Reply to this comment

Jaypal jaypalsingh Permalink
June 01, 2013, 20:48

Hi Peter,

The 3 books are great and have been of great help in learning idiomatic way of writing. I have a question. Though it is a bit rhetorical. :)

I want to use a ternary operator without the else condition. Something like

(test condition) ? do this : do nothing

Example:

[jaypal:~/Temp] cat test
aa
bb
bb
aa
aa
xx
aa
ff
gg
ss
ff
gg
[jaypal:~/Temp] awk '{ a[$1]++} END { for (k in a) print (a[k]<2) ? k : "" }' test

ss

xx

I don't want awk to print blank lines. So how to do I tell it not to do anything if test is not true. Something like continue is used in loops.

Keep up the good work. Your blogs makes complex things look easy. God bless.

Reply to this comment

Jaypal Singh jaypalsingh Permalink
June 01, 2013, 21:11

I did this but not sure if this is the best way to avoid the else condition.

[jaypal:~/Temp] awk '{ a[$1]++ } END { for (k in a) printf (a[k]<2) ? k"\n" : null }' test
ss
xx

Reply to this comment

Peter Krumins pkrumins Permalink
July 05, 2013, 17:57

Here is how:

```
awk '{ a[$1]++ } END { for (k in a) if (a[k]<2) printf k"\n" }'
```

You simply replace the ternary operator with if.

Reply to this comment

Jaypal jaypalsingh Permalink
March 05, 2014, 22:01

Yep, thats what I usually use. I wanted to know if there is a way we can use ternary operator and not define the else condition. Do we need to have both conditions explicitly stated? I know it sounds a bit rhetorical, but I just wanted to know if there is an idiomatic way of using ternary but not specifying else condition.

Keep up the good work! Your perl and awk ebooks are classic! Can't wait for bash one-liner! :)

Reply to this comment

krishna Permalink
June 04, 2013, 02:32

hi,

I have a file with 20 columns and the 20th column varies from 1 to 6000. Few of the 20th column can also have the same number. I want to have the rows where 20th column |wc is greater than 3 (for the same value)
rows= 8560

```
for (( n=1; n<8560; n++ )); awk -v val=$n ' $20==val {cnt++} END if ( cnt>3 ) print $0' temp.txt
```

there was some syntax problem. Where it was wrong

Reply to this comment

Peter Krumins pkrumins Permalink
July 05, 2013, 18:03

What shell are you using? The syntax for a `for` loop is:

```
for (( exp1; exp2; exp3 )); do
  COMMANDS;
done
```

Reply to this comment

Guy Permalink
August 17, 2013, 22:17

Hi,

Hi,
I'm a non-native Linux/programmer/computer specialist, putting it mildly; I'd be happy if you can advise me w/ the following command: print the most common string in a text file (with columns, but there is only one colums, w/ DNA sequences that always contains ~150 characters) that is bigger than, say, 20 characters.

Thanks a lot,
Guy

Reply to this comment

Bhaskar Permalink
December 02, 2013, 11:48

Hi, I have an input file,and need to replace one number from that file with 4 different values (which is an array) and write output to 4 different files, the name of files are dynamically used from array (along with some fixed name).

```
{
for (i = 1; i <= 4; ++i){
if ( $2 == 200) $2 = arr[i]
print > filename[i];
}
```

however I am having problem that for loop is getting repeated for every single line it reads and finally it write only last value of array as replacement in all 4 files. I am executing this as an awk file.
Thanks

Reply to this comment

GeoffmTurner GeoffmTurner Permalink
April 30, 2014, 09:28

Great article. It brings new meaning to "learn something new every day". Much appreciated.
(Unix 30 year veteran)

Reply to this comment

Deepthi Permalink
May 05, 2014, 14:31

My question is, below is the file contents..
DDDDD
dddd
dddd
block {
ddd
ddd
dd
}
Need to delete the content from block { to till end. How can we do using awk. Please help

Reply to this comment

nani nothing Permalink
August 28, 2014, 21:09

Hi, I have below file,

ABC|info:this is a file|size:3kb|name:sample.txt
SBC|model:Benz|info:This is a CAR|name:class-c
DFR|number:c2105|name:SONY|info: this is a phone

I need to print only "info" column as
ABC;info:this is a file
SBC;info:This is a CAR
DFR;info: this is a phone

Can anybody suggest please

Reply to this comment

deven Permalink
October 08, 2014, 05:45

Hey Guys,

Need some help here. Can someone let me know how I can print the "DATATOBEEXTRACTED" using the awk command here. I Have multiple lines in same format but basically looking for awk to print between 2 values. In this case Print whatever is between "myLogicalpath" and ")"
Eg of a line:
(uname phy:/dev/myLogicalpath/DATATOBEEXTRACTED)

Reply to this comment

Haba Permalink
March 23, 2015, 18:11

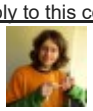
I have several of csv files, I want to remove the first row (header) and first column as well. This is my code, but it overwrites the original files with no content inside:

for x in *.csv;
do
sed '1d' $x | cut -d, -f 1 > "$x";

done

Help would be highly appreciated.

Reply to this comment



Peter Krumins pkrumins Permalink
March 23, 2015, 19:11

You don't need to loop over files. You can use sed's `-i` argument. This argument replaces data in-place and rewrites the file, and creates a backup .bak version of the file.

Here's how you do it:

```
sed -i.bak '1d; s/^[^,]\+,//; s/^\s\+//' *.csv
```

Explanation:

`1d` delete the first line and goes to next line.

`s/^[^,]\+,//` delete everything from the start to the first comma.

`s/^\s\+//` cleanups any whitespace at the beginning.

Reply to this comment

Fords Permalink
August 28, 2015, 22:13

I have sample data as below, first column is hostname and second is randdom #

host1.net 1
host2.net 14
host30.net 24

host0.net 11

host9.net 5

I need an awk script to check and validate based on column 2 and print entire line to a separate file if it matches criteria. I also need to change the number for different scripts.

I have below code but is not working as expected, any help would be appreciated.

while read line do if [[$(awk '$2 == "1"')]] then echo "$line" >> /var/failed.txt fi done < /var/results.txt

Thanks

Reply to this comment

amr attia Permalink
November 25, 2015, 11:52

I have file with | separator and I need to print first field $1 and other fields need to print part which match the regular expression \[(.*?)\>\
i thin code will be like that

cat x.txt |nawk -F"|" '{print $1"|"match regex $2,"|" match regex $3}'

sample of file

123|aaa[AAA]78->dfd|100
124|bbb[BBB]78->dfd|100
125|ccc[CCC]78->dfd|100
126|ddd[DDD]78->dfd|100
127|eee[EEE]78->dfd|100
128|fff[FFF]78->dfd|100

output
123|[AAA]78|100
124|[BBB]78|100

any on can help please

Reply to this comment

Rafidah_080312 Permalink
December 02, 2015, 02:32

Hi Guys,

Need some help here.
I have 2 files.
File1=list of all pattern that I want.
File2=original file with wanted and unwanted pattern.

I am trying to search for this wanted pattern from File1 in File2 and put the match pattern and its line below into new file but until now I did not find the correct command.
I am using shell script.
Hope someone can help.
Thank you.

Below is the excerpt from both file:
File 1:
>aka:TKWG_16035
>amim:MIM_c14350
>cbaa:SRAA_2322
>cbab:SMCB_2338
>lch:Lcho_3755
>rme:Rmet_1511
>rge:RGE_36020
>rfr:Rfer_1393
>pna:Pnap_1982
>cnc:CNE_2c13480
>reh:H16_B1390
>reh:PHG422_cbbFp
>rpj:N234_28310
>cti:RALTA_B1696
>pdx:Psed_1418
>aza:AZKH_p0228
>bbag:E1O_14830
>mpt:Mpe_A2788
>vap:Vapar_3026
>vpd:VAPA_1c30920
>mph:MLP_19560

File2:
>aka:TKWG_16035 inositol phosphatase/fructose-1,6-bisphosphatase; K01086 fructose-1,6-bisphosphatase I / sedoheptulose-1,7-bisphosphatase [EC:3.1.3.11 3.1.3.37] (A)
MILNIALVCRRISHAIACGALAGVNGSADSTNVQGEDQKKLDILSNDLFIEGNQWGGQLA
GMVSEELDEPYSIPKPYSRGRYLLLFDPLDGSSNIDVNVAVGSIFSVLRTPDSRRTTTVK
DYLQPGVKQVAAGYAIYGPSTMLVLSVGNGVHAFTLCPLVGEFMLTHPNLQVPEDTNEFA
INTSNSRFWDEPVRRYVDECIAGKTGERGKDFNMRWIASLVAEAHRILMRGGVFLYPRDN
KLPARNGRLRLLYEANPIGFIMEQAGGRASTGHEPVLQVVPTEIHQRIGFIFGSKNEVER
IEEYHKHPRRVDSPNPLFHERSLFTESPNML
>amim:MIM_c14350 fbp1; fructose-1,6-bisphosphatase class 1 (EC:3.1.3.11); K01086 fructose-1,6-bisphosphatase I / sedoheptulose-1,7-bisphosphatase [EC:3.1.3.11 3.1.3.37] (A)
MQTPQRPTLTQFLIDERRRFPEAGGDFNSVILNIALVCRRISHAIACGALAGVNGSADST
NVQGEDQKKLDILSNDLFIEGNQWGGQLAGMVSEELDEPYSIPKPYTRGRYLLLFDPLDG
SSNIDVNVSVGSIFSVLRAPDSRRATKAEDYLQPGVEQVAAGYAIYGPATMLVLSVGTGV
HAFTLCPLVGEFMLTHPNLKVPEDTSEFAINTSNSRFWEEPVRRYVDECIAGKTGERGKD

FNMRWIASLVAEAHRILMRGGVFLYPRDNKQPARNGRLRLLYEANPIGFIMEQAGGRAST
GHEPVMQVKPTDIHQRIGFIFGSKNEVERIEQYHKHPRRVDSPNPLFHERSLFTETPNMQ
>app:CAP2UW1_0957 Inositol phosphatase/fructose-16-bisphosphatase; K01086 fructose-1,6-bisphosphatase I / sedoheptulose-1,7-bisphosphatase [EC:3.1.3.11 3.1.3.37] (A)
MLCGRTNVNKFLIEEQRRNPALAGDFSMLISDVVRSCKAIAQGVSRGSLAGVLGDHGSEN
VQGEAQKKLDVLANQAFIHHCEWGGHLAAMASEEMDDIYPIPAENPRGKYLLVFDPLDGS
SNIDVNLSVGSIFSVLRCPEKGSAGTPTAADFLQPGSRQVAAGYALYGPSAMLVLTVGNG
THGFTLDRNIGEFLLTHPNLVIAADTHEFAINASNERFWEKPVQRYVAECLAGKTGVRGK
DFNMRWVASMVAEVHRILMRGGVFLYPRDSKDTSKPGRLRLMYEANPMAMIVGQAGGLAS
TGNERILDVQPTSLHQRVPVILGSRDEVERIERYHREHETGEDQPFRSPLFSTRTLFRDD
>aza:AZKH_p0228 fbp; inositol phosphatase/fructose-1,6-bisphosphatase; K01086 fructose-1,6-bisphosphatase I / sedoheptulose-1,7-bisphosphatase [EC:3.1.3.11 3.1.3.37] (A)
MFDIERTTLTEYLIAQRRLHPGATGEFNALILQVAQACKAISRAVEHGALAGVLGSLDTE
NVQGETQKKLDVLADRIFLRATHWGGELAGMVSEENDAPIPLPEGTRRGKYLLVFDPLDG
SSNIDVNVSVGSIFSVLRAPTPGEDAKPEDFLQPGTRQVAAGYAIYGPSTMLVLTVGDGV
AGFTFDPIIGEFFLTHPEIGIPTTTREFAINAANSRFWEPPVRRYVDECLAGRSGPRESD
FNMRWVASLVAETHRILMRGGVFLYPRDRKDPAKPGRLRLLYECNPIGMIIEQAGGRAST
GEGPVLDVQPTSLHQRIGFVFGSREEVERIERYHAEPAPQPDSDLPLFHSRSLFRQN
>bbag:E1O_14830 fructose-1,6-/sedoheptulose-1,7-bisphosphatase; K01086 fructose-1,6-bisphosphatase I / sedoheptulose-1,7-bisphosphatase [EC:3.1.3.11 3.1.3.37] (A)
MLEKKITLTQFLIEERRRFPQASGAFNSILLNIALACKAISRKVAYGALSGIHGAVGNTN
VQGESQQKLDVLADDTFLHCLQWSGHARGLLSEERDEPWINETEGEGEYLVAFDPLDGSS
NIDVNVSVGSIFSILRAPEGGRRAKIEDFLRPGSEQVAAGYAVYGPTTMLVLTVGRGVQG
FTLDPELGEFVLTHPQMTIPAATKEFAINASNSRFWEAPVRAYIEECLAGVNGPRGKDFN
MRWVASLVAEAHRILTRGGVFLYPRDTKDLKKPGRLRLLYEANPIGMLIEQAGGRASTGY
GPMLAVQPESTHQRIGFVFGSREEVERIERYHAETPAAAGGSDTPLFNARGLFRE
>cbaa:SRAA_2322 fbp1; fructose-1,6-bisphosphatase; K01086 fructose-1,6-bisphosphatase I / sedoheptulose-1,7-bisphosphatase [EC:3.1.3.11 3.1.3.37] (A)
MLQTNRSTLTQFLIEERRRFPGATGDFNALILDVTLACKAIASAVAYGELGGAMGNHAPE
AGGSVNVQGETQKKLDVLSNEVFMQRTEWSGHLAGMASEEMDLPHQIPAQYPRGHYLLVF
DPLDGSSNIDVNVSVGSIFSVLRAPQAVIDSGRDVVEADFLQPGATQMAAGYALYGPTTM
LILTVGRGVNGFTLDPNLGEFMLTHPDMKIPPDTQEFAINASNARFWEAPVKRYVDECLA
GKTGPRGKDFNMRWIASMVAEAHRILMRGGVFLYPRDTKDPLKPGRLRLLYEANPVGMIM
EQAGGRASTGREPMMGVVPTSLHQRIGLVFGSKNEVERIERYHREPAQAKEGEPLFATRS
LFRD

Reply to this comment

Fatih M. KILIÇ fatihmkilic Permalink
January 13, 2016, 07:30

Hello, I want to get the line in file with a specific range. For instance, inclusive 8th. 18th. 28th. print the line .... end of file.
My code is:
awk '{ for (NR=8; NR <= 118; NR=NR+10) print; exit }' input.txt
Where is the my fault? This code repeat same lines six times.

Reply to this comment

vaishali Permalink
January 15, 2016, 13:55

Fatih;
Awk '{a=NR/8;b=int(NR/8); if (a == b){ print;}; }' input.txt

Reply to this comment

Fatih M. KILIÇ fatihmkilic Permalink
June 30, 2016, 11:42

Thanks for your response. But, this code getting the lines like this 8. 16. 24. ....

I'd like to 8 , 18, 28, 38.. lines. So, like 8x+10 formula.

Reply to this comment

Gaurav Bansal Permalink

February 16, 2016, 12:27

I have the following problem. There is a file that has a group of lines repeating themselves in some order. So say
AAAAA XXX YYY ZZZ
a=1
b=2
c=3
d=ACCEPT

Then
BBBBB RRR SSS TTT
a=2
b=4
c=8
d=DROP

:
:

XXXXX CCC DDD EEE
a=2
b=4
c=10
d=DROP

This continues. I want to write a script that searches for the pattern on the first line. If it finds then skips next n lines and searches for the substring DROP in the fourth line. If it finds increments a counter. I have the following script but it doesn't work correctly

/AAAA.*/ {
{{for (i=1; i<4; i++) getline var} if (match (substr(var,3,4), "DROP")) print drop; drop++;}
}

Can anyone suggest what is wrong in this?

Reply to this comment

mahesh Permalink
April 23, 2016, 22:01

I have data as below
12345678
98765443232424
98764232353536
98765443232324
98765443232123
12345634
87654432324242
87642323535364
87654432323245
87654432321237

and I want to print as below
==========
12345678 98765443232424 98764232353536
98765443232324 98765443232123

12345634 87654432324242 87642323535364
87654432323245 87654432321237

Is this possible using awk command

Reply to this comment

mahesh Permalink
April 23, 2016, 22:02

==========
12345678 98765443232424 98764232353536
98765443232324 98765443232123

12345634 87654432324242 87642323535364
87654432323245 87654432321237
==========

Reply to this comment

mahesh Permalink
April 23, 2016, 22:02

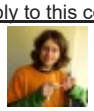
basically all the numbers which are more in length should come in column 2 & 3

Reply to this comment

Casey Permalink
May 10, 2016, 20:52

thanks for posting this. you could probably answer 1000 SO questions with this page.

Reply to this comment



Peter Krumins pkrumins Permalink
May 11, 2016, 00:44

You're welcome. :)

Reply to this comment

Jeff Permalink
August 19, 2016, 11:50

Hi there, I need a little help.

I have a file (hostlist.txt) that contains the following sample,

BA6648 Apps:InforMS
BA12105 MKT-TEST
BA8574 GAB:Global Big Company
BA13398 Test:LnHndlrDev
BA13848 DXRY:Fire
BA28278 HAPA:ds-28
.... etc

and basically wish to look up the BA reference in the file, and lookup the complete string after the space, so I've used initially....

appId=`grep -i ba6648 hostlist.txt | awk '{print $2}'`
echo $appId

which gives me the result "Apps:InforMS"

BUT when I look up...

appId=`grep -i ba8574 hostlist.txt | awk '{print $2}'`

I get "GHP:Global" but I really need "GHP:Global Big Company"

So, how can I extract everything from $2 to the end of the line with awk?

Thank you in advance,

Jeff

Reply to this comment

karn Permalink
February 08, 2017, 16:11

Hello Peter, I am great fan of your awk lines.. i have below question if you could help in a better short way...

I am looking forward to Match all the three pattern from the command and place a space , it should be continue with all 3 matching pattern with the entire output received but in case if it gets only the first match it did not get rest two then don't print that single match at all ....

Below is the real command output from which i want to get this, where if you see first match "dn" searched and it don't contain othe 2 patterns of search. at the same time it would be great it we can have awk or sed itself in place of grep itself ...

$ ldapsearch -h myldapserver -x -LLL -b "ou=profile,o=ferry.com" "cn=*" | grep -Ei "^dn|defaultServerList|preferredServerList"
dn: cn=proxyagent,ou=profile,o=ferry.com
dn: cn=default,ou=profile,o=ferry.com
preferredServerList: 192.68.8.15 192.68.8.16
defaultServerList: 192.68.8.15 192.68.8.16 192.68.88.27
dn: cn=austin, ou=profile, o=ferry.com
defaultServerList: 192.68.63.10 10.209.208.23
preferredServerList: 192.68.88.14 10.28.15.10
dn: cn=sanjose, ou=profile, o=ferry.com
preferredServerList: 192.68.8.15 192.68.8.16
defaultServerList: 192.68.8.15 192.68.8.16 192.68.88.38

Expected output is :

dn: cn=proxyagent,ou=profile,o=ferry.com (--> This is single matched found without 2 others, which i don't want to be printed if its alone without 2 others)

dn: cn=default,ou=profile,o=ferry.com
preferredServerList: 192.68.8.15 192.68.8.16
defaultServerList: 192.68.8.15 192.68.8.16 192.68.88.27

dn: cn=austin, ou=profile, o=ferry.com
defaultServerList: 192.68.63.10 10.209.208.23
preferredServerList: 192.68.88.14 10.28.15.10

Reply to this comment

Leave a new comment

Name:

E-mail: (why do I need your e-mail?)

It would be nice if you left your e-mail address. Sometimes I want to send a private message, or just thank for the great comment. Having your e-mail really helps.

I will never ever spam you.

Twitter: (Your twitter handle, if you have one.)

Website:

Comment:

Comment Help

* use `<pre>...</pre>` to insert a plain code snippet.

* use `<pre lang="lang">...</pre>` to insert a syntax highlighted code snippet.
For example, `<pre lang="python">...</pre>` will insert Python highlighted code.

* use `<code>...</code>` to highlight a variable or a single shell command.

* use `<a href="url" nospam>title</a>` to insert links.

* use other HTML tags, such as, `<b>`, `<i>`, `<blockquote>`, `<sup>`, `<sub>` for text formatting.

Type the word "**network_122**": (just to make sure you're a human)

Preview comment Submit comment

Please **preview** the comment before submitting to make sure it's OK.

Advertisements