How can I extract a predetermined range of lines from a text file on Unix?

Asked 16 years, 3 months ago   Modified 4 months ago

Viewed 676k times

683

I have a `~23000` line SQL dump containing several databases worth of data. I need to extract a certain section of this file (i.e. the data for a single database) and place it in a new file. I know both the start and end line numbers of the data that I want.

Does anyone know a Unix command (or series of commands) to extract all lines from a file between say line `16224` and `16482` and then redirect them into a new file?

bash   text-processing

Share

Improve this question

Follow

edited Jul 6 at 7:36

Penny Liu

17.2k ●5 ●86 ●108

asked Sep 17, 2008 at 13:40

Adam J. Forster

18.6k ●9 ●27 ●21

Since you mention large files, I suggest checking comment stackoverflow.com/questions/83329/…
– sancho.s ReinstateMonicaCellio Dec 13, 2015 at 12:42

28 Answers

Sorted by: Highest score (default)

979

```
sed -n '16224,16482p;16483q' filename > newfile
```

From the sed manual:

> **p** - Print out the pattern space (to the standard output). This command is usually only used in conjunction with the -n command-line option.
>
> **n** - If auto-print is not disabled, print the pattern space, then, regardless, replace the pattern space with the next line of input. If there is no more input then sed exits without processing any more commands.
>
> **q** - Exit `sed` without processing any more commands or input. Note that the current pattern space is printed if auto-print is not disabled with the -n option.

[and](and)

> Addresses in a sed script can be in any of the following forms:
>
> **number** Specifying a line number will match only that line in the input.
>
> An address range can be specified by specifying two addresses separated by a comma (,). An address range matches lines starting from where the first address matches, and continues until the second address matches (inclusively).

Share Improve this answer Follow

edited Jun 13, 2018 at 18:09
avandeursen
8,638 ● 3 ● 43 ● 53

answered Sep 17, 2008 at 13:42
boxxar
11.4k ● 2 ● 22 ● 7

3 I was curious if this modifies the original file. I backed it up just in case and it appears this did NOT modify the original, as expected. – Andy Groff Aug 6, 2012 at 19:54

2 @AndyGroff. To modify the file in place use "-i" parameter. Otherwise it will not modify the file. – youri Jan 25, 2013 at 13:06

217 If, like me, you need to do this on a VERY large file, it helps if you add a quit command on the next line. Then it's `sed -n '16224,16482p;16483q' filename`. Otherwise sed will keep scanning till the end (or at least my version does). – wds Feb 1, 2013 at 13:40

2 @wds - Your comment well deserves an answer that climbs to the top. It can make the difference between day and night. – sancho.s ReinstateMonicaCellio Dec 13, 2015 at 12:41

sed is powerful, I found this tutorial easy to read. Have fun :)..... grymoire.com/Unix/Sed.html#uh-15b – suhao399 Dec 5, 2016 at 2:39

239

```
sed -n '16224,16482 p' orig-data-file > new-file
```

Where 16224,16482 are the start line number and end line number, inclusive. This is 1-indexed. `-n` suppresses echoing the input as output, which you clearly don't want; the numbers indicate the range of lines to make the following command operate on; the command `p` prints out the relevant lines.

Share Improve this answer Follow

edited Oct 19, 2010 at 10:55

answered Sep 17, 2008 at 13:46

JXG
7,393 ● 8 ● 36 ● 63

9 On large files, the above command will continue walking the entire file after the desired range has been found. Is there a way to have sed stop processing the file once the range has been output? – Gary Dec 14, 2011 at 17:21

46 Well, from the answer here, it seems that stopping at the end of the range could be accomplished with: `sed -n`

`'16224,16482p;16482q' orig-data-file > new-file` .
– Gary Dec 14, 2011 at 17:43

8 Why would you put in an unnecessary space, and then have to quote? (Of course, making unnecessary problems and solving them is the essence of half of computer science, but I mean beside that reason ...) – Kaz Oct 16, 2013 at 18:36

140

Quite simple using `head` / `tail` :

```
head -16482 in.sql | tail -258 > out.sql
```

using `sed` :

```
sed -n '16224,16482p' in.sql > out.sql
```

using `awk` :

```
awk 'NR>=16224&&NR<=16482' in.sql > out.sql
```

Share Improve this answer Follow

edited Jul 6 at 7:36
Penny Liu
17.2k ●5 ●86 ●108

answered Sep 17, 2008 at 13:46
manveru
2,860 ●1 ●21 ●17

3 The second and third options are OK, but the first is slower than many alternatives because it uses 2 commands where 1 is sufficient. It also requires computation to get the right

argument to `tail`. – Jonathan Leffler Jan 5, 2015 at 18:42

3 Worth noting that to keep the same line numbers as the question, the sed command should be `sed -n 16224,16482p' in.sql >out.sql` and the awk command should be `awk 'NR>=16224&&NR<=16482' in.sql > out.sql` – sibaz Feb 26, 2015 at 12:39

4 Also worth knowing that in the case of the first example `head -16482 in.sql | tail -$((16482-16224)) >out.sql` leaves the computation down to bash – sibaz Feb 26, 2015 at 12:45

9 The first one with head and tail WAYYYY faster on big files than the sed version, even with q-option added. head-version instant and sed version I Ctrl-C after a minute... Thanks – Miyagi Oct 21, 2016 at 7:59

3 Could also use `tail -n +16224` to reduce computation – SOFe Oct 12, 2018 at 7:13

45

You could use 'vi' and then the following command:

```
:16224,16482w!/tmp/some-file
```

Alternatively:

```
cat file | head -n 16482 | tail -n 258
```

EDIT:- Just to add explanation, you use **head -n 16482** to display first 16482 lines then use **tail -n 258** to get last 258 lines out of the first output.

Share Improve this answer Follow

edited Jul 16, 2016 at 20:37
user2593869
143 ● 2 ● 10

answered Sep 17, 2008 at 13:42
Mark Janssen
15.1k ● 2 ● 18 ● 4

2 And instead of vi you could use ex, that is vi minus interactive console stuff. – Tadeusz A. Kadłubowski Mar 25, 2010 at 6:43

3 You don't need the `cat` command; `head` can read a file directly. This is slower than many alternatives because it uses 2 (3 as shown) commands where 1 is sufficient. – Jonathan Leffler Jan 5, 2015 at 18:41

3 @JonathanLeffler You are quite wrong. It's blazingly fast. I extract 200k lines, about 1G, from a 2G file with 500k lines, in a few seconds (without the `cat`). Other solutions need at least a few minutes. Also the fastest variation on GNU seems to be `tail -n +XXX filename | head XXX`. – Antonis Christofides Feb 5, 2016 at 11:21

The vi solution is a problem with big files. I like using the cat command as one can see the line numbers from it before sending the output to the file. Like `cat -n fileName | head -n16482 | tail -n258`, just append `> outputFile` to get the output to a file. – oceano1970526 Mar 28 at 21:09

34

There is another approach with `awk`:

```
awk 'NR==16224, NR==16482' file
```

If the file is huge, it can be good to `exit` after reading the last desired line. This way, it won't read the following lines unnecessarily:

```
awk 'NR==16224, NR==16482-1; NR==16482 {print; exit}'

awk 'NR==16224, NR==16482; NR==16482 {exit}' file
```

Share Improve this answer Follow

edited Apr 27, 2020 at 4:59

answered Jan 14, 2014 at 16:30

fedorqui
289k ● 108 ● 585 ● 627

2 1+ for saving runtime and resources by using `print; exit`. Thanks ! – Bernie Reiter Apr 4, 2019 at 16:13

dinging for being **`27.6 %`** slower than what `awk` is actually capable of – RARE Kpop Manifesto Oct 1, 2022 at 6:04

21

People trying to wrap their heads around computing an interval for the `head | tail` combo are overthinking it.

Here's how you get the "16224 -- 16482" range without computing anything:

```
cat file | head -n +16482 | tail -n +16224
```

Explanation:

- The `+` instructs the `head`/`tail` command to "*go up to / start from*" (respectively) the specified line number **as counted from the beginning of the file**.
- Similarly, a `-` instructs them to "*go up to / start from*" (respectively) the specified line number **as counted from the end of the file**
- The solution shown above simply uses `head` first, to '**keep everything up to the top number**', and then `tail` second, to '**keep everything from the bottom number upwards**', thus defining our range of interest (with no need to compute an interval).

Share Improve this answer Follow

answered Apr 12, 2021 at 18:42

Tasos Papastylianou
22.2k ● 2 ● 34 ● 63

3 I find it easier to understand with names than with numbers; I wrote it as `head -n +"$last_line" "$full_log_file" | tail -n +"$first_line" > "$cropped_log_file"` – ShadSterling Nov 23, 2021 at 18:44

2 This is wonderful. This version can run significantly faster though depending on the range and file size: `head -n +16482 file | tail -n +16224` – Akaisteph7 Jun 21 at 16:07

@Akaisteph7 ahahah, yes, I was wondering how I managed to avoid a "useless use of cat" comment on this yet! (here I did it mostly to make the symmetry clear, since people above me were doing the same; but yes, one should absolutely avoid it if unnecessary). Thanks for pointing this out! – Tasos Papastylianou Jun 24 at 7:52

```
perl -ne 'print if 16224..16482' file.txt > new_file.t
```

19

Share Improve this answer Follow

edited Jan 5, 2015 at 18:38
Jonathan Leffler
752k ● 145 ● 946 ● 1.3k

answered Sep 17, 2008 at 13:43
mmaibaum
2,499 ● 1 ● 14 ● 11

@JonathanLeffler In several of your responses you wrote "This is slower than many alternatives because it uses 2 (3 as shown) commands where 1 is sufficient." May I ask if the above `perl` oneliner is what you mean by "1 is sufficient" and/or do you mean the `sed` or `awk` oneliners? Which is the fastest? – Setaa Mar 25, 2021 at 6:00

My comments primarily refer to UUoC — Useless Use of `cat`. Using `cat file | something …` where `something` can read directly from a file should always be slower than making `something` read directly from the file because the `cat` command has to read the file and write it to the pipe, and `something` has to read the contents of the pipe and process that. It means more copying of the data in the file than is necessary. That is the basis for my assertion. I've not carried out the formal tests, but it would take something weird to avoid a slowdown. – Jonathan Leffler Mar 25, 2021 at 6:36

That 'something weird' might conceivably be the `splice(2)` (see also `splice(2)`). OTOH, I don't know whether `cat` on Linux uses it — you'd have to study the source code for `cat`. – Jonathan Leffler Mar 25, 2021 at 6:39

I note that my solution is vulnerable to the 'but it reads the whole file' problem. Perl might optimize the code but you'd have to study the generated byte code to know whether it does. But this definitely avoids the UUoC issue. Note that not all uses of a leading `cat` are wrong, but using a single file name argument usually is. Using `cat "$@" | something …` can handle 0, 1 or many command line arguments, and homegenizes the input to `something` as a single file — which can be important. But `something "$@"` probably works equally well (unless `something` is spelled `tr`, etc). – Jonathan Leffler Mar 25, 2021 at 6:57

14

Standing on the shoulders of boxxar, I like this:

```
sed -n '<first line>,$p;<last line>q' input
```

e.g.

```
sed -n '16224,$p;16482q' input
```

The `$` means "last line", so the first command makes `sed` print all lines starting with line `16224` and the second command makes `sed` quit *after* printing line `16428`. (Adding `1` for the `q`-range in boxxar's solution does not seem to be necessary.)

I like this variant because I don't need to specify the ending line number twice. And I measured that using `$` does not have detrimental effects on performance.

Share Improve this answer edited Aug 2 at 22:37

Follow

Benjamin Loison
5,582 ● 4 ● 19 ● 37

answered Feb 14, 2019 at 13:52

Tilman Vogel
9,763 ● 4 ● 34 ● 34

10

```
# print section of file based on line numbers
sed -n '16224 ,16482p'               # method 1
sed '16224,16482!d'                 # method 2
```

Share Improve this answer
Follow

edited Aug 2 at 22:36

Benjamin Loison
5,582 ● 4 ● 19 ● 37

answered Sep 17, 2008 at 13:42

Cetra
2,611 ● 1 ● 22 ● 27

7

```
cat dump.txt | head -16224 | tail -258
```

should do the trick. The downside of this approach is that you need to do the arithmetic to determine the argument for tail and to account for whether you want the 'between' to include the ending line or not.

Share Improve this answer
Follow

edited Jan 5, 2015 at 18:39

Jonathan Leffler
752k ● 145 ● 946 ● 1.3k

answered Sep 17, 2008 at 13:49

JP Lodine
509 ● 5 ● 13

4 You don't need the `cat` command; `head` can read a file directly. This is slower than many alternatives because it uses 2 (3 as shown) commands where 1 is sufficient.
– Jonathan Leffler Jan 5, 2015 at 18:31

2 @JonathanLeffler This answer is the easiest to read and to remember. If you really cared about performance you wouldn't have been using a shell in the first place. It is good practice to let specific tools dedicate themselves to a certain task. Furthermore, the "arithmetic" can be resolved using `| tail -$((16482 - 16224))` . – Yeti May 17, 2018 at 11:32

5

`sed -n '16224,16482p' < dump.sql`

Share Improve this answer Follow

answered Sep 17, 2008 at 13:45
rami
1,596 ● 12 ● 13

3

Quick and dirty:

```
head -16428 < file.in | tail -259 > file.out
```

Probably not the best way to do it but it should work.

BTW: 259 = 16482-16224+1.

Share Improve this answer

Follow

answered Sep 17, 2008 at 13:44

jan.vdbergh
2,119 ● 2 ● 20 ● 29

This is slower than many alternatives because it uses 2 commands where 1 is sufficient. – Jonathan Leffler Jan 5, 2015 at 18:29

3

I wrote a Haskell program called [splitter]() that does exactly this: have a [read through my release blog post]().

You can use the program as follows:

```
$ cat somefile | splitter 16224-16482
```

And that is all that there is to it. You will need Haskell to install it. Just:

```
$ cabal install splitter
```

And you are done. I hope that you find this program useful.

Share Improve this answer

Follow

answered Jul 25, 2013 at 22:43

Robert Massaioli
13.5k ● 7 ● 58 ● 74

Does `splitter` only read from standard input? In a sense, it doesn't matter; the `cat` command is superfluous whether it does or does not. Either use `splitter 16224-16482 <`

`somefile` or (if it takes file name arguments) `splitter 16224-16482 somefile`. – Jonathan Leffler Jan 5, 2015 at 18:31

3

Even we can do this to check at command line:

```
cat filename|sed 'n1,n2!d' > abc.txt
```

For Example:

```
cat foo.pl|sed '100,200!d' > abc.txt
```

Share Improve this answer Follow

edited Feb 5, 2014 at 7:02
Ahmed Salman Tahir
1,781 ● 1 ● 17 ● 26

answered Feb 5, 2014 at 6:41
Chinmoy Padhi
39 ● 1

6 You don't need the `cat` command in either of these; `sed` is perfectly capable of reading files on its own, or you could redirect standard input from a file. – Jonathan Leffler Jan 5, 2015 at 18:28

3

Using ruby:

```
ruby -ne 'puts "#{$.}: #{$_}" if $. >= 32613500 && $.
GND.extract.rdf
```

Share Improve this answer

Follow

answered May 21, 2015 at 12:23

Carl Blakeley
31 ● 2

3

I wanted to do the same thing from a script using a variable and achieved it by putting quotes around the $variable to separate the variable name from the p:

```
sed -n "$first","$count"p imagelist.txt >"$imageblock"
```

I wanted to split a list into separate folders and found the initial question and answer a useful step. (split command not an option on the old os I have to port code to).

Share Improve this answer

Follow

answered Oct 28, 2017 at 9:35

KevinY
1,239 ● 1 ● 13 ● 28

3

Just benchmarking 3 solutions given above, that works to me:

- awk
- sed
- "head+tail"

Credits on the 3 solutions goes to:

- @boxxar

- @avandeursen
- @wds
- @manveru
- @sibaz
- @SOFe
- @fedorqui 'SO stop harming'
- @Robin A. Meade

I'm using a huge file I find in my server:

```
# wc fo2debug.1.log
   10421186    19448208 38795491134 fo2debug.1.log
```

38 Gb in 10.4 million lines.

And yes, I have a logrotate problem. :))

Make your bets!

Getting 256 lines from the beginning of the file.

```
# time sed -n '1001,1256p;1256q' fo2debug.1.log | wc -
256

real    0m0,003s
user    0m0,000s
sys     0m0,004s

# time head -1256 fo2debug.1.log | tail -n +1001 | wc
```

```
256

real	0m0,003s
user	0m0,006s
sys	0m0,000s

# time awk 'NR==1001, NR==1256; NR==1256 {exit}' fo2de
256

real	0m0,002s
user	0m0,004s
sys	0m0,000s
```

Awk won. Technical tie in second place between sed and "head+tail".

Getting 256 lines at the end of the first third of the file.

```
# time sed -n '3473001,3473256p;3473256q' fo2debug.1.l
256

real	0m0,265s
user	0m0,242s
sys	0m0,024s

# time head -3473256 fo2debug.1.log | tail -n +3473001
256

real	0m0,308s
user	0m0,313s
sys	0m0,145s

# time awk 'NR==3473001, NR==3473256; NR==3473256 {exi
l
256

real	0m0,393s
user	0m0,326s
sys	0m0,068s
```

Sed won. Followed by "head+tail" and, finally, awk.

Getting 256 lines at the end of the second third of the file.

```
# time sed -n '6947001,6947256p;6947256q' fo2debug.1.l
A256

real    0m0,525s
user    0m0,462s
sys     0m0,064s

# time head -6947256 fo2debug.1.log | tail -n +6947001
256

real    0m0,615s
user    0m0,488s
sys     0m0,423s

# time awk 'NR==6947001, NR==6947256; NR==6947256 {exi
l
256

real    0m0,779s
user    0m0,650s
sys     0m0,130s
```

Same results.

Sed won. Followed by "head+tail" and, finally, awk.

Getting 256 lines near the end of the file.

```
# time sed -n '10420001,10420256p;10420256q' fo2debug.
256

real    1m50,017s
user    0m12,735s
```

```
sys     0m22,926s

# time head -10420256 fo2debug.1.log | tail -n +104200
256

real    1m48,269s
user    0m42,404s
sys     0m51,015s

# time awk 'NR==10420001, NR==10420256; NR==10420256 {
wc -l
256

real    1m49,106s
user    0m12,322s
sys     0m18,576s
```

And suddenly, a twist!

"Head+tail" won. Followed by awk and, finally, sed.

(some hours later...)

Sorry guys!

My analysis above ends up being an example of a basic flaw in doing an analysis.

The flaw is not knowing in depth the resources used for the analysis.

In this case, I used a log file to analyze the performance of a search for a certain number of lines within it.

Using 3 different techniques, searches were made at different points in the file, comparing the performance of the techniques at each point and checking whether the results varied depending on the point in the file where the search was made.

My mistake was to assume that there was a certain homogeneity of content in the log file.

The reality is that long lines appear more frequently at the end of the file.

Thus, the apparent conclusion that longer searches (closer to the end of the file) are better with a given technique, may be biased. In fact, this technique may be better when dealing with longer lines. What remains to be confirmed.

Share Improve this answer Follow

edited Aug 2 at 22:39
Benjamin Loison
5,582 ● 4 ● 19 ● 37

answered Sep 9, 2020 at 19:11
Anselmo Blanco Dominguez
309 ● 2 ● 11

2

I was about to post the head/tail trick, but actually I'd probably just fire up emacs. ;-)

1. esc - x goto-line ret 16224
2. mark (ctrl - space)

3. esc - x goto-line ret 16482
4. esc - w

open the new output file, ctl-y save

Let's me see what's happening.

Share Improve this answer Follow

edited May 26, 2010 at 15:40

guerda
24k ● 28 ● 100 ● 150

answered Sep 17, 2008 at 13:49

sammyo
1,023 ● 1 ● 7 ● 8

4 Emacs doesn't perform very well on very large files in my experience. – Greg Mattes Aug 25, 2011 at 15:31

Can you run that as a scripted action, or is it only an interactive option? – Jonathan Leffler Jan 5, 2015 at 18:37

2

I would use:

```
awk 'FNR >= 16224 && FNR <= 16482' my_file > extracted
```

FNR contains the record (line) number of the line being read from the file.

Share Improve this answer

Follow

answered Jan 1, 2013 at 18:51

Paddy3118

4,752 ● 31 ● 40

2

Using ed:

```
ed -s infile <<<'16224,16482p'
```

`-s` suppresses diagnostic output; the actual commands are in a here-string. Specifically, `16224,16482p` runs the `p` (print) command on the desired line address range.

Share Improve this answer

Follow

edited Feb 22, 2020 at 22:49

answered Sep 12, 2019 at 21:48

Benjamin W.

51.6k ● 19 ● 125 ● 131

I wrote a small bash script that you can run from your command line, so long as you update your PATH to

1

include its directory (or you can place it in a directory that is already contained in the PATH).

Usage: $ pinch filename start-line end-line

```
#!/bin/bash
# Display line number ranges of a file to the terminal
# Usage: $ pinch filename start-line end-line
# By Evan J. Coon

FILENAME=$1
START=$2
END=$3

ERROR="[PINCH ERROR]"

# Check that the number of arguments is 3
if [ $# -lt 3 ]; then
    echo "$ERROR Need three arguments: Filename Start-
    exit 1
fi

# Check that the file exists.
if [ ! -f "$FILENAME" ]; then
    echo -e "$ERROR File does not exist. \n\t$FILENAME
    exit 1
fi

# Check that start-line is not greater than end-line
if [ "$START" -gt "$END" ]; then
    echo -e "$ERROR Start line is greater than End lin
    exit 1
fi

# Check that start-line is positive.
if [ "$START" -lt 0 ]; then
    echo -e "$ERROR Start line is less than 0."
    exit 1
fi

# Check that end-line is positive.
if [ "$END" -lt 0 ]; then
```

```
    echo -e "$ERROR End line is less than 0."
    exit 1
fi

NUMOFLINES=$(wc -l < "$FILENAME")

# Check that end-line is not greater than the number o
if [ "$END" -gt "$NUMOFLINES" ]; then
    echo -e "$ERROR End line is greater than number of
    exit 1
fi

# The distance from the end of the file to end-line
ENDDIFF=$(( NUMOFLINES - END ))

# For larger files, this will run more quickly. If the
# end of the file to the end-line is less than the dis
# start of the file to the start-line, then start pinc
# bottom as opposed to the top.
if [ "$START" -lt "$ENDDIFF" ]; then
    < "$FILENAME" head -n $END | tail -n +$START
else
    < "$FILENAME" tail -n +$START | head -n $(( END-ST
fi

# Success
exit 0
```

Share Improve this answer Follow

answered Dec 10, 2014 at 17:06

DrNerdfighter
79 ● 5

1 This is slower than many alternatives because it uses 2 commands where 1 is sufficient. In fact, it reads the file twice because of the `wc` command, which wastes disk bandwidth, especially on gigabyte files. In all sorts of ways, this is well documented, but it is also engineering overkill.
– Jonathan Leffler Jan 5, 2015 at 18:35

Love the word "pinch" to describe what this does. :D – Setaa
Jun 14, 2020 at 18:59

1

This might work for you (GNU sed):

```
sed -ne '16224,16482w newfile' -e '16482q' file
```

or taking advantage of bash:

```
sed -n $'16224,16482w newfile\n16482q' file
```

Share Improve this answer Follow

answered Aug 10, 2015 at 13:00

potong
58.3k ● 6 ● 54 ● 90

1

Since we are talking about extracting lines of text from a text file, I will give an special case where you want to extract all lines that match a certain pattern.

```
myfile content:
=====================
line1 not needed
line2 also discarded
[Data]
first data line
second data line
=====================
sed -n '/Data/,$p' myfile
```

Will print the [Data] line and the remaining. If you want the text from line1 to the pattern, you type: sed -n '1,/Data/p' myfile. Furthermore, if you know two pattern (better be unique in your text), both the beginning and end line of the range can be specified with matches.

```
sed -n '/BEGIN_MARK/,/END_MARK/p' myfile
```

Share Improve this answer Follow

answered Feb 14, 2018 at 22:52

Kemin Zhou
6,821 ● 3 ● 53 ● 60

1

I've compiled some of the highest rated solutions for `sed`, `perl`, `head+tail`, plus my own code for `awk`, and focusing on performance via the pipe, while using `LC_ALL=C` to ensure all candidates at their fastest possible, allocating 2-second sleep gap in between.

The gaps are somewhat noticeable :

```
   abs time    awk/app speed ratio
 ----------------------------------
   0.0672 sec :   1.00x mawk-2
   0.0839 sec :   1.25x gnu-sed
   0.1289 sec :   1.92x perl
   0.2151 sec :   3.20x gnu-head+tail
```

Haven't had chance to test `python` or `BSD` variants of those utilities.

```
 (fg && fg && fg && fg) 2>/dev/null;
 echo;
 ( time ( pvE0 < "${m3t}"
        | LC_ALL=C  mawk2 '

           BEGIN {
                     _=10420001-(\
                    __=10420256)^(FS="^$")
           } _<NR {
                   print

                   if(__==NR) { exit }

     }' ) | pvE9) | tee >(xxh128sum >&2) | LC_ALL=C gw
    sleep 2;
    (fg && fg && fg && fg) 2>/dev/null
    echo;
    ( time ( pvE0 < "${m3t}"
           | LC_ALL=C gsed -n '10420001,10420256p;1042

     ) | pvE9 ) |  tee >(xxh128sum >&2) |  LC_ALL=C gw
     sleep  2; (fg && fg && fg && fg) 2>/dev/null
     echo
    ( time ( pvE0 < "${m3t}"
           | LC_ALL=C perl -ne 'print if 10420001..104

    ) | pvE9 ) |  tee >(xxh128sum >&2) |  LC_ALL=C gwc
    sleep  2; (fg && fg && fg && fg) 2>/dev/null
    echo
    ( time ( pvE0 < "${m3t}"
           | LC_ALL=C ghead -n +10420256
           | LC_ALL=C gtail -n +10420001
    ) | pvE9 ) |  tee >(xxh128sum >&2) |  LC_ALL=C gwc


      in0: 1.51GiB 0:00:00 [2.31GiB/s] [2.31GiB/s] [==
     out9: 42.5KiB 0:00:00 [64.9KiB/s] [64.9KiB/s] [ <
( pvE 0.1 in0 < "${m3t}" | LC_ALL=C mawk2 ; )

   0.43s user 0.36s system 117% cpu 0.672 total
    256   43487   43487

54313365c2e66a48dc1dc33595716cc8  stdin
```

```
      out9: 42.5KiB 0:00:00 [51.7KiB/s] [51.7KiB/s] [ <
       in0: 1.51GiB 0:00:00 [1.84GiB/s] [1.84GiB/s] [==

    ( pvE 0.1 in0 < "${m3t}" |LC_ALL=C gsed -n '1042000

    0.68s user 0.34s system 121% cpu 0.839 total
     256    43487    43487

 54313365c2e66a48dc1dc33595716cc8  stdin


       in0: 1.85GiB 0:00:01 [1.46GiB/s] [1.46GiB/s] [==
      out9: 42.5KiB 0:00:01 [33.5KiB/s] [33.5KiB/s] [

 ( pvE 0.1 in0 < "${m3t}" | LC_ALL=C perl -ne 'print if

    1.10s user 0.44s system 119% cpu 1.289 total
     256    43487    43487

 54313365c2e66a48dc1dc33595716cc8  stdin

       in0: 1.51GiB 0:00:02 [ 728MiB/s] [ 728MiB/s] [==
      out9: 42.5KiB 0:00:02 [19.9KiB/s] [19.9KiB/s] [ <

  ( pvE 0.1 in0 < "${m3t}"
    | LC_ALL=C ghead -n +10420256
    | LC_ALL=C gtail -n ; )

  1.98s user 1.40s system 157% cpu 2.151 total
  256    43487    43487

 54313365c2e66a48dc1dc33595716cc8  stdin
```

Share Improve this answer Follow

answered Apr 22, 2022 at 15:03

RARE Kpop Manifesto

2,758 ● 5 ● 14

I imagine that this looks like the source code for a game in temple OS. Split the different methods into separate code block, and list out the definitions for the pv functions, as it's

far from obvious what those `pvE0` things are.
– Ярослав Рахматуллин Jan 5, 2023 at 10:16

1 @ЯрославРахматуллин : `pvE0` and `pvE9` are just my convenience shortcuts for `PARAM0='-tra'     PARAM1='-peb'; instr0="$( gprintf '%s ' nice pv $PARAM0 $PARAM1 -i "$VAR1" -c -N "'$( gecho "$VAR2" | escapeSQuote3 )'" -L "$VAR3" -s "$VAR4" "'$( gecho "$VAR5" | escapeSQuote3 )'" )"    eval "${instr0}"` —— basically things to create a uniform look and feel of `pv` but using a long timestamp for `-` aka `/dev/stdin`. i use them extensively since `pv` is a great tool for figuring out bottlenecks along the chain of piped commands. – RARE Kpop Manifesto Jan 5, 2023 at 10:32

what is escapeSQuote3 ? – Ярослав Рахматуллин Jan 5, 2023 at 11:22

anyway. my point is your code example is incomplete and unusable, so you may as well not include it. What is `"${m3t}`"? – Ярослав Рахматуллин Jan 5, 2023 at 11:25

@ЯрославРахматуллин : `"${m3t}"` is just a filename. escapeQuote3 is just to properly escape single quotes for the shell : `echo "abc'xyz" | mawk 'gsub(_,(_)__(_)_)^_' \_='\47' __='\'     abc'\''xyz` .... if it's incomplete, then how did i run the benchmarks ? "Unusable" ? You can't spend 5 seconds adapting your scenario to it ?
– RARE Kpop Manifesto Jan 5, 2023 at 13:26

0

The -n in the accept answers work. Here's another way in case you're inclined.

```
cat $filename | sed "${linenum}p;d";
```

This does the following:

1. pipe in the contents of a file (or feed in the text however you want).
2. sed selects the given line, prints it
3. d is required to delete lines, otherwise sed will assume all lines will eventually be printed. i.e., without the d, you will get all lines printed by the selected line printed twice because you have the ${linenum}p part asking for it to be printed. I'm pretty sure the -n is basically doing the same thing as the d here.

Share Improve this answer Follow

edited Jan 7, 2016 at 18:52
fedorqui
289k ● 108 ● 585 ● 627

answered Dec 10, 2015 at 14:27
ThinkBonobo
16.4k ● 10 ● 71 ● 83

3 note `cat file | sed` is better written as `sed file` – fedorqui Jan 7, 2016 at 18:53

Also this just prints a line, whereas the question is about a range of them. – fedorqui Jan 7, 2016 at 18:54

0

I was looking for an answer to this but I had to end up writing my own code which worked. None of the answers above were satisfactory. Consider you have very large file and have certain line numbers that you want to print out but the numbers are not in order. You can do the following:

My relatively large file `for letter in {a..k} ; do echo $letter; done | cat -n > myfile.txt`

```
  1	a
  2	b
  3	c
  4	d
  5	e
  6	f
  7	g
  8	h
  9	i
 10	j
 11	k
```

Specific line numbers I want: `shuf -i 1-11 -n 4 > line_numbers_I_want.txt`

```
 10
 11
 4
 9
```

To print these line numbers, do the following. `awk '{system("head myfile.txt -n " $0 " | tail -n 1")}' line_numbers_I_want.txt`

What the above does is to head the n line then take the last line using tail

If you want your line numbers in order, sort (is -n numeric sort) first then get the lines.

`cat line_numbers_I_want.txt | sort -n | awk '{system("head myfile.txt -n " $0 " | tail -n 1")}'`

```
 4  d
 9  i
10  j
11  k
```

Share Improve this answer

Follow

answered Feb 27, 2021 at 3:22

Kahiga

33 ● 4

0

Maybe, you would be so kind to give this humble script a chance ;-)

```
#!/usr/bin/bash

# Usage:
#   body n m|-m

from=$1
to=$2

if [ $to -gt 0 ]; then
# count $from the begin of the file $to selected line
    awk "NR >= $from && NR <= $to {print}"
else
# count $from the begin of the file skipping tailing $
    awk '
    BEGIN   {lines=0; from='$from'; to='$to'}
            {++lines}
    NR >= $from {line[lines]=$0}
    END     {for (i = from; i < lines + to + 1; i++) {
                print line[i]
            }
    }'
fi
```

Outputs:

```
$ seq 20 | ./body.sh 5 15
5
6
7
8
9
10
11
12
13
14
15

$ seq 20 | ./body.sh 5 -5
5
6
7
8
9
10
11
12
13
14
15
```

Share Improve this answer Follow

answered Apr 22, 2022 at 13:01

user2928048

4,118 ● 2 ● 14 ● 12

You could use `sed` command in your case and is pretty fast.

0

As mentioned lets assume the range is: between 16224 and 16482 lines

```
#get the lines from 16224 to 16482 and prints the valu
        sed -n '16224 ,16482p' file.txt > filename.txt

#Additional Info to showcase other possible scenarios:

#get the 16224 th line and writes the value to filenam

        sed -n '16224p' file.txt > filename.txt

#get the 16224 and 16300 line values only and write to

        sed -n '16224p;16300p;' file.txt > filename.tx
```

Share Improve this answer Follow

answered Oct 15, 2022 at 5:01

Du-Lacoste
12.7k ● 2 ● 76 ● 57

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.