Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

Teams
Q&A for work

Learn More

Looping through the content of a file in Bash

Ask Question

1132

333

How do I iterate through each line of a text file with Bash?

With this script:

```
echo "Start!"
for p in (peptides.txt)
do
    echo "${p}"
done
```

I get this output on the screen:

```
Start!
./runPep.sh: line 3: syntax error near unexpected token `('
./runPep.sh: line 3: `for p in (peptides.txt)'
```

(Later I want to do something more complicated with `$p` than just output to the screen.)

The environment variable **SHELL** is (from env):

```
SHELL=/bin/bash
```

`/bin/bash --version` output:

```
GNU bash, version 3.1.17(1)-release (x86_64-suse-linux-gnu)
Copyright (C) 2005 Free Software Foundation, Inc.
```

```
Linux version 2.6.18.2-34-default (geeko@buildhost) (gcc version 4.1.2 20061115
(prerelease) (SUSE Linux)) #1 SMP Mon Nov 27 11:46:27 UTC 2006
```

The file peptides.txt contains:

```
RKEKNVQ
IPKKLLQK
QYFHQLEKMNVK
IPKKLLQK
GDLSTALEVAIDCYEK
QYFHQLEKMNVKIPENIYR
RKEKNVQ
VLAKHGKLQDAIN
ILGFMK
LEDVALQILL
```

linux bash loops unix io

edited Jun 19 '18 at 6:50
iNet
1

asked Oct 5 '09 at 17:52
Peter Mortensen
14.1k 19 88 114

8 Oh, I see many things have happened here: all the comments were deleted and the question being reopened. Just for reference, the accepted answer in Read a file line by line assigning the value to a variable addresses the problem in a canonical way and should be preferred over the accepted one here. – fedorqui Aug 30 '16 at 9:44

11 Answers

1793

```
while read p; do
  echo "$p"
done <peptides.txt
```

As pointed out in the comments, this has the side effects of trimming leading whitespace, interpretting backslash sequences, and skipping the trailing line if it's missing a terminating linefeed. If these are concerns, you can do:

```
while IFS="" read -r p || [ -n "$p" ]
do
  printf '%s\n' "$p"
done < peptides.txt
```

Exceptionally, if the loop body may read from standard input, you can open the file using a different file descriptor:

```
while read -u 10 p; do
  ...
done 10<peptides.txt
```

Here, 10 is just an arbitrary number (different from 0, 1, 2).

edited Aug 20 '18 at 20:55
that other guy
76.8k 8 88 126

answered Oct 5 '09 at 18:00
Bruno De Fraine
31.2k 8 47 58

7 How should I interpret the last line? File peptides.txt is redirected to standard input and somehow to the whole of

command has something to consume." My "cat" method is similar, sending the output of a command into the while block for consumption by 'read', too, only it launches another program to get the work done. – Warren Young Oct 5 '09 at 18:30

6 This method seems to skip the last line of a file. – xastor Nov 7 '13 at 7:48

3 Double quote the lines !! echo "$p" and the file.. trust me it will bite you if you don't!!! I KNOW! lol – Mike Q Aug 19 '14 at 17:01

1 Both versions fail to read a final line if it is not terminated with a newline. **Always** use `while read p || [[ -n $p ]]; do ...` – dawg Sep 7 '16 at 14:15

346

```
cat peptides.txt | while read line
do
   # do something with $line here
done
```

answered Oct 5 '09 at 17:54
Warren Young
31.6k 7 71 87

59 In general, if you're using "cat" with only one argument, you're doing something wrong (or suboptimal). – JesperE Oct 5 '09 at 18:02

24 Yes, it's just not as efficient as Bruno's, because it launches another program, unnecessarily. If efficiency matters, do it Bruno's way. I remember my way because you can use it with other commands, where the "redirect in from" syntax doesn't work. – Warren Young Oct 5 '09 at 18:12

70 There's another, more serious problem with this: because the

can be very annoying (depending on what you're trying to do in the loop). – Gordon Davisson Oct 6 '09 at 0:57

21 I use "cat file | " as the start of a lot of my commands purely because I often prototype with "head file |" – mat kelcey Feb 26 '14 at 21:33

45 This may be not that efficient, but it's much more readable than other answers. – Savage Reader Dec 22 '14 at 13:02

130

Option 1a: While loop: Single line at a time: Input redirection

```
#!/bin/bash
filename='peptides.txt'
echo Start
while read p; do
    echo $p
done < $filename
```

Option 1b: While loop: Single line at a time:
Open the file, read from a file descriptor (in this case file descriptor #4).

```
#!/bin/bash
filename='peptides.txt'
exec 4<$filename
echo Start
while read -u4 p ; do
    echo $p
done
```

Option 2: For loop: Read file into single variable and parse.
This syntax will parse "lines" based on any white space between the tokens. This still works because the given

Also, reading the full file into a single variable is not a good strategy for large files.

```
#!/bin/bash
filename='peptides.txt'
filelines=`cat $filename`
echo Start
for line in $filelines ; do
    echo $line
done
```

edited Jul 6 '18 at 10:42

Gerard Bosch
134 1 14

answered Oct 5 '09 at 18:18

Stan Graves
5,659 2 14 14

For option 1b: does the file descriptor need to be closed again? E.g. the loop could be an inner loop. – Peter Mortensen Oct 5 '09 at 20:03

3 The file descriptor will be cleaned up with the process exits. An explicit close can be done to reuse the fd number. To close a fd, use another exec with the &- syntax, like this: exec 4<&- – Stan Graves Oct 5 '09 at 21:09

1 Thank you for Option 2. I ran into huge problems with Option 1 because I needed to read from stdin within the loop; in such a case Option 1 will not work. – masgo Jun 4 '14 at 13:50

3 You should point out more clearly that Option 2 is strongly discouraged. @masgo Option 1b should work in that case, and can be combined with the input redirection syntax from Option 1a by replacing `done < $filename` with `done 4<$filename` (which is useful if you want to read the file name from a command parameter, in which case you can just replace `$filename` by `$1`). – Egor Hans Nov 12 '17 at 16:44

appears to be the only way? – user5359531 Nov 12 '18 at 23:21

69

This is no better than other answers, but is one more way to get the job done in a file without spaces (see comments). I find that I often need one-liners to dig through lists in text files without the extra step of using separate script files.

```
for word in $(cat peptides.txt); do echo $word; done
```

This format allows me to put it all in one command-line. Change the "echo $word" portion to whatever you want and you can issue multiple commands separated by semicolons. The following example uses the file's contents as arguments into two other scripts you may have written.

```
for word in $(cat peptides.txt); do cmd_a.sh $word; cmd_b.
```

Or if you intend to use this like a stream editor (learn sed) you can dump the output to another file as follows.

```
for word in $(cat peptides.txt); do cmd_a.sh $word; cmd_b.
```

I've used these as written above because I have used text files where I've created them with one word per line. (See comments) If you have spaces that you don't want splitting your words/lines, it gets a little uglier, but the same command still works as follows:

```
OLDIFS=$IFS; IFS=$'\n'; for line in $(cat peptides.txt); do
$line; done > outfile.txt; IFS=$OLDIFS
```

This just tells the shell to split on newlines only, not spaces, then returns the environment back to what it was previously. At this point, you may want to consider putting it all into a shell script rather than squeezing it all into a single line, though.

Best of luck!

edited Dec 22 '13 at 15:47

answered Oct 4 '13 at 13:30

mightypile
4,236 2 22 26

4 The bash $(<peptides.txt) is perhaps more elegant, but it's still wrong, what Joao said correct, you are performing command substitution logic where space or newline is the same thing. If a line has a space in it, the loop executes TWICE or more for that one line. So your code should properly read: for word in $(<peptides.txt); do .... If you know for a fact there are no spaces, then a line equals a word and you're okay. – maxpolk Dec 8 '13 at 17:58

2 @JoaoCosta,maxpolk : Good points that I hadn't considered. I've edited the original post to reflect them. Thanks! – mightypile Dec 22 '13 at 15:49

2 Using `for` makes the input tokens/lines subject to shell expansions, which is usually undesirable; try this: `for l in $(echo '* b c'); do echo "[$l]"; done` - as you'll see, the `*` - even though originally a *quoted* literal - expands to the files in the current directory. – mklement0 Dec 22 '13 at 16:09

2 @dblanchard: The last example, using $IFS, should ignore spaces. Have you tried that version? – mightypile Nov 24 '15 at 0:53

2 The way how this command gets a lot more complex as crucial

circumvented by bringing in escaped quotes, which again makes things more complex and less readable). – Egor Hans Nov 12 '17 at 14:23

56

A few more things not covered by other answers:

Reading from a delimited file

```
# ':' is the delimiter here, and there are three fields on
# IFS set below is restricted to the context of `read`, it
while IFS=: read -r field1 field2 field3; do
  # process the fields
  # if the line has less than three fields, the missing fi
string
  # if the line has more than three fields, `field3` will 
the third field plus the delimiter(s)
done < input.txt
```

Reading from the output of another command, using process substitution

```
while read -r line; do
  # process the line
done < <(command ...)
```

This approach is better than `command ... | while read -r line; do ...` because the while loop here runs in the current shell rather than a subshell as in the case of the latter. See the related post A variable modified inside a while loop is not remembered.

Reading from a null delimited input, for

```
while read -r -d '' line; do
  # logic
  # use a second 'read ... <<< "$line"' if we need to toke
done < <(find /path/to/dir -print0)
```

Related read: BashFAQ/020 - How can I find and safely handle file names containing newlines, spaces or both?

Reading from more than one file at a time

```
while read -u 3 -r line1 && read -u 4 -r line2; do
  # process the lines
  # note that the loop will end when we reach EOF on eithe
`&&`
done 3< input1.txt 4< input2.txt
```

Based on @chepner's answer here:

`-u` is a bash extension. For POSIX compatibility, each call would look something like `read -r X <&3`.

Reading a whole file into an array (Bash versions earlier to 4)

```
while read -r line; do
    my_array+=("$line")
done < my_file
```

If the file ends with an incomplete line (newline missing at the end), then:

```
while read -r line || [[ $line ]]; do
    my_array+=("$line")
done < my_file
```

Reading a whole file into an array (Bash versions 4x and later)

```
readarray -t my_array < my_file
```

or

```
mapfile -t my_array < my_file
```

And then

```
for line in "${my_array[@]}"; do
  # process the lines
done
```

- More about the shell builtins `read` and `readarray` commands - GNU
- More about `IFS` - Wikipedia
- BashFAQ/001 - How can I read a file (data stream, variable) line-by-line (and/or field-by-field)?

Related posts:

- Creating an array from a text file in Bash
- What is the difference between thee approaches to reading a file that has just one line?
- Bash while read loop extremely slow compared to cat, why?

edited Oct 26 '18 at 5:40

19.2k 8 42 68

note that instead of `command < input_filename.txt` you can always do `input_generating_command | command` or `command < <(input_generating_command)` – masterxilo Mar 7 at 14:00

42

Use a while loop, like this:

```
while IFS= read -r line; do
   echo "$line"
done <file
```

Notes:

1. If you don't set the `IFS` properly, you will lose indentation.
2. You should almost always use the -r option with read.
3. Don't read lines with `for`

edited Mar 29 '17 at 0:10

answered Jun 9 '15 at 15:09

Jahid
14.1k 4 63 85

2 Why the `-r` option? – David C. Rankin Jun 23 '15 at 2:31

2 @DavidC.Rankin The -r option prevents backslash interpretation. `Note #2` is a link where it is described in detail... – Jahid Jun 23 '15 at 6:01

@FlorinAndrei : The above example doesn't need the `-u` option, are you talking about another example with `-u` ? – Jahid Feb 17 '17 at 5:37

Looked through your links, and was surprised there's no answer that simply links your link in Note 2. That page provides everything you need to know about that subject. Or are link-only answers discouraged or something? – Egor Hans Nov 12 '17 at 16:49

13

If you don't want your read to be broken by newline character, use -

```
#!/bin/bash
while IFS='' read -r line || [[ -n "$line" ]]; do
    echo "$line"
done < "$1"
```

Then run the script with file name as parameter.

answered Mar 8 '16 at 16:10

Anjul Sharma
139 1 3

12

Suppose you have this file:

```
$ cat /tmp/test.txt
Line 1
    Line 2 has leading space
Line 3 followed by blank line

Line 5 (follows a blank line) and has trailing space
```

There are four elements that will alter the meaning of the file output read by many Bash solutions:

1. The blank line 4;
2. Leading or trailing spaces on two lines;
3. Maintaining the meaning of individual lines (i.e., each line is a record);
4. The line 6 not terminated with a CR.

If you want the text file line by line including blank lines and terminating lines without CR, you must use a while loop and you must have an alternate test for the final line.

Here are the methods that may change the file (in comparison to what `cat` returns):

1) Lose the last line and leading and trailing spaces:

```
$ while read -r p; do printf "%s\n" "'$p'"; done </tmp/tes
'Line 1'
'Line 2 has leading space'
'Line 3 followed by blank line'
''
'Line 5 (follows a blank line) and has trailing space'
```

(If you do `while IFS= read -r p; do printf "%s\n" "'$p'"; done </tmp/test.txt` instead, you preserve the leading and trailing spaces but still lose the last line if it is not terminated with CR)

2) Using process substitution with `cat` will reads the entire file in one gulp and loses the meaning of individual lines:

```
$ for p in "$(cat /tmp/test.txt)"; do printf "%s\n" "'$p'"
'Line 1
    Line 2 has leading space
Line 3 followed by blank line
```

(If you remove the `"` from `$(cat /tmp/test.txt)` you read the file word by word rather than one gulp. Also probably not what is intended...)

The most robust and simplest way to read a file line-by-line and preserve all spacing is:

```
$ while IFS= read -r line || [[ -n $line ]]; do printf "'%s
</tmp/test.txt
'Line 1'
'    Line 2 has leading space'
'Line 3 followed by blank line'
''
'Line 5 (follows a blank line) and has trailing space    '
'Line 6 has no ending CR'
```

If you want to strip leading and trading spaces, remove the `IFS=` part:

```
$ while read -r line || [[ -n $line ]]; do printf "'%s'\n"
'Line 1'
'Line 2 has leading space'
'Line 3 followed by blank line'
''
'Line 5 (follows a blank line) and has trailing space'
'Line 6 has no ending CR'
```

(A text file without a terminating `\n`, while fairly common, is considered broken under POSIX. If you can count on the trailing `\n` you do not need `|| [[ -n $line ]]` in the `while` loop.)

More at the BASH FAQ

edited May 2 '17 at 17:17

dawg
61k 12 88 155

My I ask why the downvote? – dawg Nov 4 '18 at 16:30

4

```
#!/bin/bash
#
# Change the file name from "test" to desired input file
# (The comments in bash are prefixed with #'s)
for x in $(cat test.txt)
do
    echo $x
done
```

edited Mar 24 '14 at 17:57
0zkr PM
585 6 11

answered Nov 14 '13 at 14:23
Sine
99 1 2

6 This answer needs the caveats mentioned in mightypile's answer, and it can fail badly if any line contains shell metacharacters (due to the unquoted "$x"). – Toby Speight Jun 8 '15 at 16:32

6 I'm actually surprised people didn't yet come up with the usual Don't read lines with for... – Egor Hans Nov 12 '17 at 14:17

Here is my real life example how to loop lines of another from variable, use that variable outside of the loop. I guess quite many is asking these questions sooner or later.

```
##Parse FPS from first video stream, drop quotes from fps
## streams.stream.0.codec_type="video"
## streams.stream.0.r_frame_rate="24000/1001"
## streams.stream.0.avg_frame_rate="24000/1001"
FPS=unknown
while read -r line; do
  if [[ $FPS == "unknown" ]] && [[ $line == *".codec_type=
    echo ParseFPS $line
    FPS=parse
  fi
  if [[ $FPS == "parse" ]] && [[ $line == *".r_frame_rate=
    echo ParseFPS $line
    FPS=${line##*=}
    FPS="${FPS%\"}"
    FPS="${FPS#\"}"
  fi
done <<< "$(ffprobe -v quiet -print_format flat -show_form
if [ "$FPS" == "unknown" ] || [ "$FPS" == "parse" ]; then
  echo ParseFPS Unknown frame rate
fi
echo Found $FPS
```

Declare variable outside of the loop, set value and use it outside of loop requires *done <<< "$(...)"* syntax. Application need to be run within a context of current console. Quotes around the command keeps newlines of output stream.

Loop match for substrings then reads *name=value* pair, splits right-side part of last = character, drops first quote, drops last quote, we have a clean value to be used elsewhere.

answered Jun 30 '15 at 8:15

Whome
7,925 3 30 53

many other answers. Plus, it completely drowns in your FPS example. – Egor Hans Nov 12 '17 at 14:14

1

@Peter: This could work out for you-

```
echo "Start!";for p in $(cat ./pep); do
echo $p
done
```

This would return the output-

```
Start!
RKEKNVQ
IPKKLLQK
QYFHQLEKMNVK
IPKKLLQK
GDLSTALEVAIDCYEK
QYFHQLEKMNVKIPENIYR
RKEKNVQ
VLAKHGKLQDAIN
ILGFMK
LEDVALQILL
```

answered Aug 30 '15 at 5:00

Alan Jebakumar
69 1

7 This is very bad! [Why you don't read lines with "for"](). – fedorqui Jun 16 '16 at 10:43

2 This answer is defeating all the principles set by the good answers above! – codeforester Jan 14 '17 at 2:55

2 Please delete this answer. – dawg May 2 '17 at 17:18

2 Now guys, don't exaggerate. The answer is bad, but it seems

1 @EgorHans, I disagree strongly: The point of answers is to teach people how to write software. Teaching people to do things in a way that you *know* is harmful to them and the people who use their software (introducing bugs / unexpected behaviors / etc) is knowingly harming others. An answer known to be harmful has no "right to exist" in a well-curated teaching resource (and curating it is exactly what we, the folks who are voting and flagging, are supposed to be doing here). – Charles Duffy Sep 20 '18 at 16:36

protected by Inian Apr 17 '18 at 7:23

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?