Search and replace a line in a file in Python

Asked 16 years, 3 months ago Modified 1 year ago Viewed 650k times

382

I want to loop over the contents of a text file and do a search and replace on some lines and write the result back to the file. I could first load the whole file in memory and then write it back, but that probably is not the best way to do it.

What is the best way to do this, within the following code?

```
f = open(file)
for line in f:
    if line.contains('foo'):
        newline = line.replace('foo', 'bar')
        # how to write this newline back to the file
```

python file

Share
Improve this question
Follow

edited Sep 7, 2009 at 10:11
SilentGhost
319k ● 67 ● 310 ● 294

asked Sep 2, 2008 at 9:19
pkit
8,301 ● 7 ● 37 ● 37

Your mental model of how files work is incomplete. You seem to imagine that there is a "grid" and you can replace one row in the grid without touching the rest. But this isn't how it works; when you start writing to a specific position in a file, you replace everything after that point in the file. Python provides some convenience functions for supporting your mental model but down on the filesystem layer, those are just a useful illusion. The software ends up replacing the rest of the file, albeit then with mostly the same contents as it contained before you started writing. – tripleee Nov 10, 2023 at 11:52

13 Answers

Sorted by: Highest score (default)

311

The shortest way would probably be to use the fileinput module. For example, the following adds line numbers to a file, in-place:

```
import fileinput

for line in fileinput.input("test.txt", inplace=True):
    print('{} {}'.format(fileinput.filelineno(), line), end='') # for Python 3
    # print "%d: %s" % (fileinput.filelineno(), line), # for Python 2
```

What happens here is:

1. The original file is moved to a backup file
2. The standard output is redirected to the original file within the loop
3. Thus any `print` statements write back into the original file

`fileinput` has more bells and whistles. For example, it can be used to automatically operate on all files in `sys.args[1:]`, without your having to iterate over them explicitly. Starting with Python 3.2 it also provides a convenient context manager for use in a `with` statement.

While `fileinput` is great for throwaway scripts, I would be wary of using it in real code because admittedly it's not very readable or familiar. In real (production) code it's worthwhile to spend just a few more lines of code to make the process explicit and thus make the code readable.

There are two options:

1. The file is not overly large, and you can just read it wholly to memory. Then close the file, reopen it in writing mode and write the modified contents back.
2. The file is too large to be stored in memory; you can move it over to a temporary file and open that, reading it line by line, writing back into the original file. Note that this requires twice the storage.

Share

Improve this answer

Follow

edited May 19, 2020 at 6:33
Alex Altair
3,665 ● 5 ● 25 ● 39

answered Nov 14, 2008 at 15:47
Eli Bendersky
273k ● 91 ● 362 ● 422

17 I know this only has two lines in it, however I don't think the code is very expressive in itself. Because if you think for a sec, if you didn't know the function, there are very few clues in what is going on. Printing the line number and the line is not the same as writing it ... if you get my gist... – chutsu May 29, 2010 at 19:12

3 i agree. how would one use fileinput to write to the file? – jml Jan 24, 2011 at 4:50

14 This **DOES** write to the file. It redirects stdout to the file. Have a look at the docs – brice Aug 24, 2011 at 16:17

34 The key bit here is the comma at the end of the print statement: it surpresses the print statement adding another newline (as line already has one). It's not very obvious at all, though (which is why Python 3 changed that syntax, luckily enough). – VPeric Oct 21, 2011 at 14:24

6 Please notice this does not work when you provide an opening hook to the file, e.g. when you try to read/write UTF-16 encoded files. – bompf Jul 1, 2013 at 12:19

236

I guess something like this should do it. It basically writes the content to a new file and replaces the old file with the new file:

```
from tempfile import mkstemp
from shutil import move, copymode
from os import fdopen, remove

def replace(file_path, pattern, subst):
    #Create temp file
    fh, abs_path = mkstemp()
    with fdopen(fh,'w') as new_file:
        with open(file_path) as old_file:
            for line in old_file:
                new_file.write(line.replace(pattern, subst))
    #Copy the file permissions from the old file to the new file
    copymode(file_path, abs_path)
    #Remove original file
    remove(file_path)
    #Move new file
    move(abs_path, file_path)
```

Share

Improve this answer

Follow

edited Jan 17, 2020 at 12:59

answered Sep 2, 2008 at 9:42

Thomas Watnedal

4,951 ● 4 ● 25 ● 23

8 Just a minor comment: `file` is shadowing predefined class of the same name.
– ezdazuzena Jan 24, 2013 at 15:24

4 This code changes the permissions on the original file. How can I keep the original permissions? – nic Jul 18, 2013 at 21:35

1 what's the point of fh, you use it in the close call but I don't see the point of creating a file just to close it... – Wicelo Sep 12, 2014 at 6:24

2 @Wicelo You need to close it to prevent leaking of the file descriptor. Here is a decent explanation: logilab.org/17873 – Thomas Watnedal Sep 19, 2014 at 11:52

1 Yes I've discovered that `mkstemp()` is returning a 2-tuple and `(fh, abs_path) = fh, abs_path`, I didn't know that when I asked the question. – Wicelo Sep 20, 2014 at 3:31

100

Here's another example that was tested, and will match search & replace patterns:

```
import fileinput
import sys

def replaceAll(file,searchExp,replaceExp):
    for line in fileinput.input(file, inplace=1):
        if searchExp in line:
            line = line.replace(searchExp,replaceExp)
        sys.stdout.write(line)
```

Example use:

```
replaceAll("/fooBar.txt","Hello\sWorld!$","Goodbye\sWorld.")
```

Share

Improve this answer

Follow

edited Apr 27, 2011 at 2:25
the Tin Man
160k ● 44 ● 221 ● 306

answered Nov 24, 2008 at 19:02
Jason
2,731 ● 1 ● 27 ● 28

26 The example use provides a regular expression, but neither `searchExp in line` nor `line.replace` are regular expression operations. Surely the example use is wrong. – kojiro Nov 14, 2011 at 18:18

1 Instead of `if searchExp in line: line = line.replace(searchExp, replaceExpr)` you can just write `line = line.replace(searchExp, replaceExpr)`. No exception is generated, the line just remains unchanged. – David Wallace Nov 15, 2017 at 16:07

Worked perfectly for me as well. I had come across a number of other examples that looked very similar to this, but the trick was the use of the `sys.stdout.write(line)`. Thanks again! – Sage Jan 16, 2018 at 17:23

2 If I use this, my file gets blank. Any idea? – Javier Lopez Tomas Sep 25, 2019 at 16:11

I also got a blank file, but only when my implementation crashed at runtime during development. Once I fixed the errors in my code - everything worked fine. – dutoitns Feb 22, 2023 at 10:28

72

This should work: (inplace editing)

```
import fileinput

# Does a list of files, and
# redirects STDOUT to the file in question
for line in fileinput.input(files, inplace = 1):
      print line.replace("foo", "bar"),
```

Share

Improve this answer

Follow

edited Jan 13, 2016 at 0:35
Gringo Suave
31.7k ● 7 ● 94 ● 81

answered Sep 7, 2009 at 10:07
Kinlan
16.6k ● 5 ● 58 ● 88

5 +1. Also if you receive a RuntimeError: input() already active then call the fileinput.close() – geographika Nov 18, 2011 at 9:24

4 Note that `files` should be a string containing the file name, not a file object. – Lee Aug 30, 2013 at 10:00

11 print adds a newline that could already be there. to avoid this, add .rstrip() at the end of your replacements – Guillaume Gendre Dec 21, 2014 at 14:09

Instead use files arg in input(), it could be fileinput.input(inplace=1) and call the script as > python replace.py myfiles*.txt – chespinoza Feb 24, 2017 at 17:45

25

Based on the answer by Thomas Watnedal. However, this does not answer the line-to-line part of the original question exactly. The function can still replace on a line-to-line basis

This implementation replaces the file contents without using temporary files, as a consequence file permissions remain unchanged.

Also re.sub instead of replace, allows regex replacement instead of plain text replacement only.

Reading the file as a single string instead of line by line allows for multiline match and replacement.

```
import re

def replace(file, pattern, subst):
    # Read contents from file as a single string
    file_handle = open(file, 'r')
    file_string = file_handle.read()
    file_handle.close()

    # Use RE package to allow for replacement (also allowing for (multiline)
REGEX)
    file_string = (re.sub(pattern, subst, file_string))

    # Write contents to file.
    # Using mode 'w' truncates the file.
    file_handle = open(file, 'w')
    file_handle.write(file_string)
    file_handle.close()
```

Share
Improve this answer
Follow

edited Nov 30, 2012 at 9:22

answered Nov 30, 2012 at 8:51
Thijs
259 ● 3 ● 4

2 You might want to use `rb` and `wb` attributes when opening files as this will preserve original line endings – Nux Jun 1, 2016 at 14:35

In Python 3, you can't use 'wb' and 'rb' with 're'. It will give the error "TypeError: cannot use a string pattern on a bytes-like object" – user5074403 Oct 24, 2017 at 13:22

`fileinput` is quite straightforward as mentioned on previous answers:

21

```
import fileinput

def replace_in_file(file_path, search_text, new_text):
    with fileinput.input(file_path, inplace=True) as file:
        for line in file:
            new_line = line.replace(search_text, new_text)
            print(new_line, end='')
```

Explanation:

- `fileinput` can accept multiple files, but I prefer to close each single file as soon as it is being processed. So placed single `file_path` in `with` statement.
- `print` statement does not print anything when `inplace=True`, because `STDOUT` is being forwarded to the original file.
- `end=''` in `print` statement is to eliminate intermediate blank new lines.

You can use it as follows:

```
file_path = '/path/to/my/file'
replace_in_file(file_path, 'old-text', 'new-text')
```

Share

Improve this answer

Follow

edited Dec 9, 2023 at 15:26

answered Oct 3, 2019 at 10:37

Akif
6,746 ● 3 ● 43 ● 45

If the new text has special characters such as Japanese glyphs, the characters don't appear properly. They're written in a form similar to `\xe8`. – Unknow0059 Nov 13, 2020 at 11:26

1 @Unknow0059 That is simply not true. It sounds like you are perhaps examining the `repr()` of a string; but when you `print()` it instead, the output is exactly equivalent to the input, unless you have done something additional to wreck it yourself. Another common beginner mistake is (to be on Windows and) use some tool which doesn't properly support the file's character encoding, or (being confused about character encodings, and) writing the output file in a different encoding. – tripleee Dec 9, 2023 at 11:15

1 You got my +1 because explain that stdout is redirected. I'm trying without that "print" and it is not working. TY! – jrborba Apr 8 at 18:29

20

As lassevk suggests, write out the new file as you go, here is some example code:

```
fin = open("a.txt")
fout = open("b.txt", "wt")
for line in fin:
    fout.write( line.replace('foo', 'bar') )
fin.close()
fout.close()
```

Share Improve this answer Follow

answered Sep 2, 2008 at 9:42
hamishmcn
7,981 ● 11 ● 42 ● 46

16

A more pythonic way would be to use context managers like the code below:

```
from tempfile import mkstemp
from shutil import move
from os import remove

def replace(source_file_path, pattern, substring):
    fh, target_file_path = mkstemp()
    with open(target_file_path, 'w') as target_file:
        with open(source_file_path, 'r') as source_file:
            for line in source_file:
                target_file.write(line.replace(pattern, substring))
    remove(source_file_path)
    move(target_file_path, source_file_path)
```

You can find the full snippet here.

Share
Improve this answer
Follow

edited Jul 13, 2016 at 19:40
formatkaka
1,358 ● 4 ● 13 ● 27

answered Sep 7, 2013 at 18:39
Kiran
1,077 ● 1 ● 11 ● 25

1 In Python >=3.1 you could open the two context managers on the same line. – florisla Feb 6, 2018 at 11:05

14

If you're wanting a generic function that replaces *any* text with some other text, this is likely the best way to go, particularly if you're a fan of regex's:

```
import re
def replace( filePath, text, subs, flags=0 ):
    with open( filePath, "r+" ) as file:
        fileContents = file.read()
        textPattern = re.compile( re.escape( text ), flags )
        fileContents = textPattern.sub( subs, fileContents )
        file.seek( 0 )
        file.truncate()
        file.write( fileContents )
```

Share Improve this answer Follow

answered Feb 18, 2014 at 14:43
starryknight64
501 ● 7 ● 15

5

Create a new file, copy lines from the old to the new, and do the replacing before you write the lines to the new file.

Share Improve this answer Follow

answered Sep 2, 2008 at 9:24

Lasse V. Karlsen
391k ● 106 ● 646 ● 844

5

Expanding on @Kiran's answer, which I agree is more succinct and Pythonic, this adds codecs to support the reading and writing of UTF-8:

```
import codecs

from tempfile import mkstemp
from shutil import move
from os import remove


def replace(source_file_path, pattern, substring):
    fh, target_file_path = mkstemp()

    with codecs.open(target_file_path, 'w', 'utf-8') as target_file:
        with codecs.open(source_file_path, 'r', 'utf-8') as source_file:
            for line in source_file:
                target_file.write(line.replace(pattern, substring))
    remove(source_file_path)
    move(target_file_path, source_file_path)
```

Share Improve this answer Follow

answered May 2, 2014 at 11:15

igniteflow
9,622 ● 10 ● 40 ● 46

Is it going to preserve the permission of the old file in the new file? – Bidyut Aug 22, 2017 at 10:27

4

Using hamishmcn's answer as a template I was able to search for a line in a file that match my regex and replacing it with empty string.

```
import re

fin = open("in.txt", 'r') # in file
fout = open("out.txt", 'w') # out file
for line in fin:
    p = re.compile('[-][0-9]*[.][0-9]*[,]|[-][0-9]*[,]') # pattern
    newline = p.sub('',line) # replace matching strings with empty string
    print newline
```

```
        fout.write(newline)
    fin.close()
    fout.close()
```

Share Improve this answer Follow

answered Apr 17, 2014 at 2:13

Emmanuel
53 ● 1 ● 5

2 You should compile the regex OUTSIDE the for loop, otherwise is a performance waste – Axel Feb 4, 2016 at 17:49

1

if you remove the indent at the like below, it will search and replace in multiple line. See below for example.

```
def replace(file, pattern, subst):
    #Create temp file
    fh, abs_path = mkstemp()
    print fh, abs_path
    new_file = open(abs_path,'w')
    old_file = open(file)
    for line in old_file:
        new_file.write(line.replace(pattern, subst))
    #close temp file
    new_file.close()
    close(fh)
    old_file.close()
    #Remove original file
    remove(file)
    #Move new file
    move(abs_path, file)
```

Share

Improve this answer

Follow

edited Apr 8, 2013 at 0:41

rowanthorpe
435 ● 3 ● 11

answered Aug 2, 2012 at 19:12

loi
21 ● 1

The formatting of this Python code doesn't look quite right... (I tried to fix, but wasn't sure what was intended) – Andy Hayden Sep 30, 2012 at 18:18

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.