UNIX & LINUX

SPONSORED BY aws

How can I replace a string in a file(s)?

Asked 8 years, 4 months ago Modified 5 months ago Viewed 1.8m times

914

502

Replacing strings in files based on certain search criteria is a very common task. How can I

- replace string `foo` with `bar` in all files in the current directory?
- do the same recursively for sub directories?
- replace only if the file name matches another string?
- replace only if the string is found in a certain context?
- replace if the string is on a certain line number?
- replace multiple strings with the same replacement
- replace multiple strings with different replacements

text-processing awk sed perl

Share Improve this question Follow

edited Jan 21, 2017 at 0:25
Braiam
33.9k 25 103 162

asked Feb 1, 2014 at 17:05
terdon ♦
216k 56 407 610

2 This is intended to be a canonical Q&A on this subject (see this meta discussion), please feel free to edit my answer below or add your own. – terdon ♦ Feb 1, 2014 at 17:08

Great `grep -rl` (then piped to `sed`) answer here: unix.stackexchange.com/questions/472476/… – Gabriel Staples Mar 20, 2020 at 0:24

10 Answers

Sorted by: Highest score (default)

1230

1. Replacing all occurrences of one string with another in all files in the current directory:

These are for cases where you *know* that the directory contains only regular files and that you want to process all non-hidden files. If that is not the case, use the approaches in 2.

All `sed` solutions in this answer assume GNU `sed`. If using FreeBSD or macOS, replace `-i` with `-i ''`. Also note that the use of the `-i` switch with any version of `sed` has certain filesystem security implications and is inadvisable in any script which you plan to distribute in any way.

- Non recursive, files in this directory only:

```
sed -i -- 's/foo/bar/g' *
perl -i -pe 's/foo/bar/g' ./*
```

(the `perl` one will fail for file names ending in `|` or space)).

- Recursive, regular files (**including hidden ones**) in this and all subdirectories

```
find . -type f -exec sed -i 's/foo/bar/g' {} +
```

If you are using zsh:

```
sed -i -- 's/foo/bar/g' **/*(D.)
```

(may fail if the list is too big, see `zargs` to work around).

Bash can't check directly for regular files, a loop is needed (braces avoid setting the options globally):

```
( shopt -s globstar dotglob;
    for file in **; do
        if [[ -f $file ]] && [[ -w $file ]]; then
            sed -i -- 's/foo/bar/g' "$file"
        fi
    done
)
```

The files are selected when they are actual files (-f) and they are writable (-w).

2. Replace only if the file name matches another string / has a specific extension / is of a certain type etc:

- Non-recursive, files in this directory only:

```
sed -i -- 's/foo/bar/g' *baz*    ## all files whose name contains baz
sed -i -- 's/foo/bar/g' *.baz    ## files ending in .baz
```

- Recursive, regular files in this and all subdirectories

```
find . -type f -name "*baz*" -exec sed -i 's/foo/bar/g' {} +
```

If you are using bash (braces avoid setting the options globally):

```
( shopt -s globstar dotglob
    sed -i -- 's/foo/bar/g' **baz*
    sed -i -- 's/foo/bar/g' **.baz
)
```

If you are using zsh:

```
sed -i -- 's/foo/bar/g' **/*baz*(D.)
sed -i -- 's/foo/bar/g' **/*.baz(D.)
```

The `--` serves to tell `sed` that no more flags will be given in the command line. This is useful to protect against file names starting with `-` .

- If a file is of a certain type, for example, executable (see `man find` for more options):

```
find . -type f -executable -exec sed -i 's/foo/bar/g' {} +
```

`zsh` :

```
sed -i -- 's/foo/bar/g' **/*(D*)
```

3. Replace only if the string is found in a certain context

- Replace `foo` with `bar` only if there is a `baz` later on the same line:

```
sed -i 's/foo\(.*baz\)/bar\1/' file
```

In `sed` , using `\( \)` saves whatever is in the parentheses and you can then access it with `\1` . There are many variations of this theme, to learn more about such regular expressions, see here.

- Replace `foo` with `bar` only if `foo` is found on the 3d column (field) of the input file (assuming whitespace-separated fields):

```
gawk -i inplace '{gsub(/foo/,"baz",$3); print}' file
```

(needs `gawk` 4.1.0 or newer).

- For a different field just use `$N` where `N` is the number of the field of interest. For a different field separator (`:` in this example) use:

```
gawk -i inplace -F':' '{gsub(/foo/,"baz",$3);print}' file
```

Another solution using `perl` :

Another solution using perl :

```
perl -i -ane '$F[2]=~s/foo/baz/g; $" = " "; print "@F\n"' foo
```

NOTE: both the `awk` and `perl` solutions will affect spacing in the file (remove the leading and trailing blanks, and convert sequences of blanks to one space character in those lines that match). For a different field, use `$F[N-1]` where `N` is the field number you want and for a different field separator use (the `$"=":"` sets the output field separator to `:`):

```
perl -i -F':' -ane '$F[2]=~s/foo/baz/g; $"=":";print "@F"' foo
```

- Replace `foo` with `bar` only on the 4th line:

  ```
  sed -i '4s/foo/bar/g' file
  gawk -i inplace 'NR==4{gsub(/foo/,"baz")};1' file
  perl -i -pe 's/foo/bar/g if $.==4' file
  ```

4. Multiple replace operations: replace with different strings

- You can combine `sed` commands:

  ```
  sed -i 's/foo/bar/g; s/baz/zab/g; s/Alice/Joan/g' file
  ```

Be aware that order matters (`sed 's/foo/bar/g; s/bar/baz/g'` will substitute `foo` with `baz`).

- or Perl commands

  ```
  perl -i -pe 's/foo/bar/g; s/baz/zab/g; s/Alice/Joan/g' file
  ```

- If you have a large number of patterns, it is easier to save your patterns and their replacements in a `sed` script file:

  ```
  #! /usr/bin/sed -f
  s/foo/bar/g
  s/baz/zab/g
  ```

- Or, if you have too many pattern pairs for the above to be feasible, you can read pattern pairs from a file (two space separated patterns, $pattern and $replacement, per line):

  ```
  while read -r pattern replacement; do
      sed -i "s/$pattern/$replacement/" file
  done < patterns.txt
  ```

- That will be quite slow for long lists of patterns and large data files so you might want to read the patterns and create a `sed` script from them instead. The following assumes a *<<!>space<!>>* delimiter separates a list of *MATCH<<!>space<!>>REPLACE* pairs occurring one-per-line in the file *patterns.txt* :

```
sed 's| *\([^ ]*\) *\([^ ]*\).*|s/\1/\2/g|' <patterns.txt |
sed -f- ./editfile >outfile
```

The above format is largely arbitrary and, for example, doesn't allow for a *<<!>space<!>>* in either of *MATCH* or *REPLACE*. The method is very general though: basically, if you can create an output stream which looks like a `sed` script, then you can source that stream as a `sed` script by specifying `sed` 's script file as `-` stdin.

- You can combine and concatenate multiple scripts in similar fashion:

```
SOME_PIPELINE |
sed -e'#some expression script'  \
    -f./script_file -f-          \
    -e'#more inline expressions' \
./actual_edit_file >./outfile
```

A POSIX `sed` will concatenate all scripts into one in the order they appear on the command-line. None of these need end in a `\n` ewline.

- `grep` can work the same way:

```
sed -e'#generate a pattern list' <in |
grep -f- ./grepped_file
```

- When working with fixed-strings as patterns, it is good practice to escape regular expression *metacharacters*. You can do this rather easily:

```
sed 's/[]$&^*\./[]/\\&/g
     s| *\([^ ]*\) *\([^ ]*\).*|s/\1/\2/g|
' <patterns.txt |
sed -f- ./editfile >outfile
```

5. Multiple replace operations: replace multiple patterns with the same string

- Replace any of `foo` , `bar` or `baz` with `foobar`

```
sed -Ei 's/foo|bar|baz/foobar/g' file
```

- or

```
perl -i -pe 's/foo|bar|baz/foobar/g' file
```

Share Improve this answer Follow

edited Mar 13, 2021 at 21:33

community wiki
34 revs, 10 users 59%
terdon

104

A good replacement Linux tool is **rpl**, that was originally written for the Debian project, so it is available with `apt-get install rpl` in any Debian derived distro, and may be for others, but otherwise you can download the `tar.gz` file from SourceForge.

Simplest example of use:

```
$ rpl old_string new_string test.txt
```

Note that if the string contains spaces it should be enclosed in quotation marks. By default `rpl` takes care of **capital letters** but not of **complete words**, but you can change these defaults with options `-i` (ignore case) and `-w` (whole words). You can also specify **multiple files**:

```
$ rpl -i -w "old string" "new string" test.txt test2.txt
```

Or even specify the **extensions** (`-x`) to search or even search **recursively** (`-R`) in the directory:

```
$ rpl -x .html -x .txt -R old_string new_string test*
```

You can also search/replace in **interactive mode** with `-p` (prompt) option:

The output shows the numbers of files/string replaced and the type of search (case in/sensitive, whole/partial words), but it can be silent with the `-q` (**quiet mode**) option, or even more verbose, listing line numbers that contain matches of each file and directory with `-v` (**verbose mode**) option.

Other options that are worth remembering are `-e` (honor **e**scapes) that allow `regular expressions`, so you can search also tabs (`\t`), new lines (`\n`),etc. You can use `-f` to **force permissions** (of course, only when the user has write permissions) and `-d` to preserve the modification times`).

Finally, if you are unsure what exactly will happen, use the `-s` (**simulate mode**).

Share Improve this answer Follow

edited Jul 14, 2020 at 16:51
Matthias Braun
6,697 6 38 47

answered Dec 27, 2015 at 8:06
Fran
1,621 1 14 8

3 So much better at the feedback and simplicity than sed. I just wish it allowed acting on file names, and then it'd be perfect as-is. – Kzqai Dec 23, 2016 at 17:12

1 i like the -s (simulate mode) :-) – m3nda Jun 10, 2018 at 11:08

1 sooo much better than `sed` . golly – Marc Compere Aug 3, 2020 at 17:26

thanks so much for this. `sed` is fine for simple replacements, but terrible for more complicated and longer strings – yeah22 Sep 15, 2020 at 19:02

1 For macOS, `rpl` is available from MacPorts. – murray Sep 23, 2020 at 20:49

28

How to do a search and replace over multiple files suggests:

> You could also use find and sed, but I find that this little line of perl works nicely.
>
> ```
> perl -pi -w -e 's/search/replace/g;' *.php
> ```
>
> - -e means execute the following line of code.
> - -i means edit in-place
> - -w write warnings
> - -p loop over the input file, printing each line after the script is applied to it.

My best results come from using perl and grep (to ensure that file have the search expression)

```
perl -pi -w -e 's/search/replace/g;' $( grep -rl 'search' )
```

Share Improve this answer Follow

edited Oct 19, 2015 at 21:38
Community Bot
1

answered Jan 16, 2015 at 14:02
Alejandro Salamanca Mazuelo
433 5 8

19

You can use Vim in Ex mode:

> replace string ALF with BRA in all files in the current directory?

```
for CHA in *
do
  ex -sc '%s/ALF/BRA/g' -cx "$CHA"
done
```

> do the same recursively for sub directories?

```
find -type f -exec ex -sc '%s/ALF/BRA/g' -cx {} ';'
```

> replace only if the file name matches another string?

```
for CHA in *.txt
do
  ex -sc '%s/ALF/BRA/g' -cx "$CHA"
done
```

> replace only if the string is found in a certain context?

```
ex -sc 'g/DEL/s/ALF/BRA/g' -cx file
```

> replace if the string is on a certain line number?

```
ex -sc '2s/ALF/BRA/g' -cx file
```

> replace multiple strings with the same replacement

```
ex -sc '%s/\vALF|ECH/BRA/g' -cx file
```

> replace multiple strings with different replacements

```
ex -sc '%s/ALF/BRA/g|%s/FOX/GOL/g' -cx file
```

Share Improve this answer Follow

edited Apr 17, 2016 at 5:31

answered Apr 17, 2016 at 1:47
Zombo
1

19

I used this:

```
grep -r "old_string" -l | tr '\n' ' ' | xargs sed -i 's/old_string/new_string/g'
```

1. List all files that contain `old_string` .
2. Replace newline in result with spaces (so that the list of files can be fed to `sed` .
3. Run `sed` on those files to replace old string with new.

Update: The above result will fail on filenames that contain whitespaces. Instead, use:

```
grep --null -lr "old_string" | xargs --null sed -i 's/old_string/new_string/g'
```

Share Improve this answer Follow

edited May 9, 2016 at 7:34
shas
2,253 4 15 28

answered Oct 26, 2015 at 16:58
o_o_o--
755 1 7 9

1 Note that this will fail if any of your file names contain spaces, tabs or newlines. Use `grep --null -lr "old_string" | xargs --null sed -i 's/old_string/new_string/g'` will make it deal with arbitrary file names. – terdon ♦ Oct 26, 2015 at 17:07

thanks guys. added update and left the old code cause it's an interesting caveat that could be useful to someone unaware of this behavior. – o_o_o-- Oct 26, 2015 at 20:59

9

From a user's perspective, a nice & simple Unix tool that does the job perfectly is `qsubst`. For example,

```
% qsubst foo bar *.c *.h
```

will replace `foo` with `bar` in all my C files. A nice feature is that `qsubst` will do a *query-replace*, i.e., it will show me each occurrence of `foo` and ask whether I want to replace it or not. [You can replace unconditionally (no asking) with `-go` option, and there are other options, e.g., `-w` if you only want to replace `foo` when it is a whole word.]

How to get it: `qsubst` was invented by der Mouse (from McGill) and posted to *comp.unix.sources 11(7)* in Aug. 1987. Updated versions exist. For example, the NetBSD version `qsubst.c,v 1.8 2004/11/01` compiles and runs perfectly on my mac.

Share Improve this answer Follow

edited Jul 30, 2015 at 11:27
terdon ♦
216k 56 407 610

answered Jul 30, 2015 at 11:25
phs
433 3 11

Searched similar solution for Linux and found `wrg` script wrapper for ripgrep by lydell: github.com/BurntSushi/ripgrep/issues/74#issuecomment-309213936 – d9k Mar 20, 2021 at 23:32

6

ripgrep (command name `rg`) is a `grep` tool, but supports search and replace as well.

```
$ cat ip.txt
dark blue and light blue
light orange
blue sky
$ # by default, line number is displayed if output destination is stdout
$ # by default, only lines that matched the given pattern is displayed
$ # 'blue' is search pattern and -r 'red' is replacement string
$ rg 'blue' -r 'red' ip.txt
1:dark red and light red
3:red sky

$ # --passthru option is useful to print all lines, whether or not it matched
$ # -N will disable line number prefix
$ # this command is similar to: sed 's/blue/red/g' ip.txt
$ rg --passthru -N 'blue' -r 'red' ip.txt
dark red and light red
light orange
red sky
```

`rg` doesn't support in-place option, so you'll have to do it yourself

```
$ # -N isn't needed here as output destination is a file
$ rg --passthru 'blue' -r 'red' ip.txt > tmp.txt && mv tmp.txt ip.txt
$ cat ip.txt
dark red and light red
light orange
red sky
```

See Rust regex documentation for regular expression syntax and features. The `-P` switch will enable PCRE2 flavor. `rg` supports Unicode by default.

```
$ # non-greedy quantifier is supported
$ echo 'food land bark sand band cue combat' | rg 'foo.*?ba' -r 'X'
Xrk sand band cue combat

$ # unicode support
$ echo 'fox:αλεπού,eagle:αετός' | rg '\p{L}+' -r '($0)'
(fox):(αλεπού),(eagle):(αετός)

$ # set operator example, remove all punctuation characters except . ! and ?
$ para='"Hi", there! How *are* you? All fine here.'
$ echo "$para" | rg '[[:punct:]--[.!?]]+' -r ''
Hi there! How are you? All fine here.

$ # use -P if you need even more advanced features
$ echo 'car bat cod map' | rg -P '(bat|map)(*SKIP)(*F)|\w+' -r '[$0]'
[car] bat [cod] map
```

Like `grep`, the `-F` option will allow fixed strings to be matched, a handy option which I feel `sed` should implement too.

```
$ printf '2.3/[4]*6\nfoo\n5.3-[4]*9\n' | rg --passthru -F '[4]*' -r '2'
2.3/26
foo
5.3-29
```

Another handy option is `-U` which enables multiline matching

```
$ # (?s) flag will allow . to match newline characters as well
$ printf '42\nHi there\nHave a Nice Day' | rg --passthru -U '(?s)the.*ice' -r ''
42
Hi  Day
```

`rg` can handle dos-style files too

```
$ # same as: sed -E 's/\w+(\r?)$/123\1/'
$ printf 'hi there\r\ngood day\r\n' | rg --passthru --crlf '\w+$' -r '123'
hi 123
good 123
```

Another advantage of `rg` is that it is likely to be faster than `sed`

```
$ # for small files, initial processing time of rg is a large component
$ time echo 'aba' | sed 's/a/b/g' > f1
real    0m0.002s
$ time echo 'aba' | rg --passthru 'a' -r 'b' > f2
real    0m0.007s

$ # for larger files, rg is likely to be faster
$ # 6.2M sample ASCII file
$ wget https://norvig.com/big.txt
$ time LC_ALL=C sed 's/\bcat\b/dog/g' big.txt > f1
real    0m0.060s
$ time rg --passthru '\bcat\b' -r 'dog' big.txt > f2
real    0m0.048s
$ diff -s f1 f2
Files f1 and f2 are identical

$ time LC_ALL=C sed -E 's/\b(\w+)(\s+\1)+\b/\1/g' big.txt > f1
real    0m0.725s
$ time rg --no-unicode --passthru -wP '(\w+)(\s+\1)+' -r '$1' big.txt > f2
real    0m0.093s
$ diff -s f1 f2
Files f1 and f2 are identical
```

Share Improve this answer Follow

edited Sep 4, 2020 at 6:47

answered Oct 8, 2019 at 5:29

Sundeep
10.9k 1 23 45

5

I needed something that would provide a dry-run option and would work recursively with a glob, and after trying to do it with `awk` and `sed` I gave up and instead did it in python.

The script searches recursively all files matching a glob pattern (e.g. `--glob="*.html"`) for a regex and replaces with the replacement regex:

```
find_replace.py [--dir=my_folder] \
    --search-regex=<search_regex> \
    --replace-regex=<replace_regex> \
    --glob=[glob_pattern] \
    --dry-run
```

Every long option such as `--search-regex` has a corresponding short option, i.e. `-s` . Run with `-h` to see all options.

For example, this will flip all dates from `2017-12-31` to `31-12-2017` :

```
python replace.py --glob=myfile.txt \
    --search-regex="(\d{4})-(\d{2})-(\d{2})" \
    --replace-regex="\3-\2-\1" \
    --dry-run --verbose
```

```
import os
import fnmatch
import sys
import shutil
import re

import argparse

def find_replace(cfg):
    search_pattern = re.compile(cfg.search_regex)

    if cfg.dry_run:
        print('THIS IS A DRY RUN -- NO FILES WILL BE CHANGED!')

    for path, dirs, files in os.walk(os.path.abspath(cfg.dir)):
        for filename in fnmatch.filter(files, cfg.glob):

            if cfg.print_parent_folder:
                pardir = os.path.normpath(os.path.join(path, '..'))
                pardir = os.path.split(pardir)[-1]
                print('[%s]' % pardir)
            filepath = os.path.join(path, filename)

            # backup original file
            if cfg.create_backup:
                backup_path = filepath + '.bak'

                while os.path.exists(backup_path):
                    backup_path += '.bak'
                print('DBG: creating backup', backup_path)
                shutil.copyfile(filepath, backup_path)

            with open(filepath) as f:
                old_text = f.read()
            all_matches = search_pattern.findall(old_text)

            if all_matches:

                print('Found {} matches in file {}'.format(len(all_matches),
filename))

                new_text = search_pattern.sub(cfg.replace_regex, old_text)

                if not cfg.dry_run:
                    with open(filepath, "w") as f:
                        print('DBG: replacing in file', filepath)
                        f.write(new_text)
                else:
                    for idx, matches in enumerate(all_matches):
                        print("Match #{}: {}".format(idx, matches))

                    print("NEW TEXT:\n{}".format(new_text))

            elif cfg.verbose:
                print('File {} does not contain search regex
"{}"'.format(filename, cfg.search_regex))


if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='''DESCRIPTION:
    Find and replace recursively from the given folder using regular
expressions''',

formatter_class=argparse.RawDescriptionHelpFormatter,
                                     epilog='''USAGE:
    {0} -d [my_folder] -s <search_regex> -r <replace_regex> -g [glob_pattern]

    '''.format(os.path.basename(sys.argv[0])))

    parser.add_argument('--dir', '-d',
                        help='folder to search in; by default current folder',
                        default='.')

    parser.add_argument('--search-regex', '-s',
                        help='search regex',
                        required=True)

    parser.add_argument('--replace-regex', '-r',
                        help='replacement regex',
                        required=True)

    parser.add_argument('--glob', '-g',
                        help='glob pattern, i.e. *.html',
                        default="*.*")

    parser.add_argument('--dry-run', '-dr',
                        action='store_true',
                        help="don't replace anything just show what is going to
be done",
                        default=False)

    parser.add_argument('--create-backup', '-b',
                        action='store_true',
                        help='Create backup files',
                        default=False)

    parser.add_argument('--verbose', '-v',
                        action='store_true',
                        help="Show files which don't match the search regex"
                        help="Show files which don't match the search regex",
                        default=False)

    parser.add_argument('--print-parent-folder', '-p',
                        action='store_true',
                        help="Show the parent info for debug",
                        default=False)

    config = parser.parse_args(sys.argv[1:])

    find_replace(config)
```

`Here` is an updated version of the script which highlights the search terms and replacements with different colors.

Share Improve this answer Follow

edited Dec 12, 2019 at 15:05

answered Nov 15, 2017 at 21:59
ccpizza
1,435 1 18 19

2 I don't understand why you would make something this complex. For recursion, use either bash's (or your shell's equivalent) `globstar` option and `**` globs or `find`. For a dry run, just use `sed`. Unless you use the `-i` option, it won't make any changes. For a backup use `sed -i.bak` (or `perl -i .bak`); for files that don't match, use `grep PATTERN file || echo file`. And why in the world would you have python expand the glob instead of letting the shell do it? Why `script.py --glob=foo*` instead of just `script.py foo*`? – terdon ♦ Nov 23, 2017 at 9:34

2 My *why's* are very simple: (1) above all, ease of debugging; (2) using only a single well documented tool with a supportive community (3) not knowing `sed` and `awk` well and being unwilling to invest extra time on mastering them, (4) readability, (5) this solution will also work on non-posix systems (not that I need that but somebody else might). – ccpizza Nov 23, 2017 at 12:59

2

Here I use `grep` to tell if it *is going to* change a file (so I can count the number of lines changed, and replacements made, to output at the end), then I use `sed` to actually change the file. Notice the single line of `sed` usage at the *very end* of the Bash function below:

`replace_str` Bash function

Update: the below code has been upgraded and is now part of my eRCaGuy_dotfiles project as "find_and_replace.sh" here. <-- I recommend you use this tool now instead.

Usage:

```
gs_replace_str "regex_search_pattern" "replacement_string" "file_path"
```

Bash Function:

```
# Usage: `gs_replace_str "regex_search_pattern" "replacement_string"
"file_path"`
gs_replace_str() {
    REGEX_SEARCH="$1"
    REPLACEMENT_STR="$2"
    FILENAME="$3"

    num_lines_matched=$(grep -c -E "$REGEX_SEARCH" "$FILENAME")
    # Count number of matches, NOT lines (`grep -c` counts lines),
    # in case there are multiple matches per line; see:
    # https://superuser.com/questions/339522/counting-total-number-of-matches-
with-grep-instead-of-just-how-many-lines-match/339523#339523
    num_matches=$(grep -o -E "$REGEX_SEARCH" "$FILENAME" | wc -l)

    # If num_matches > 0
    if [ "$num_matches" -gt 0 ]; then
        echo -e "\n${num_matches} matches found on ${num_lines_matched} lines in
file"\
                "\"${FILENAME}\":"
        # Now show these exact matches with their corresponding line 'n'umbers
in the file
        grep -n --color=always -E "$REGEX_SEARCH" "$FILENAME"
        # Now actually DO the string replacing on the files 'i'n place using the
`sed`
        # 's'tream 'ed'itor!
        sed -i "s|${REGEX_SEARCH}|${REPLACEMENT_STR}|g" "$FILENAME"
    fi
}
```

Place that in your ~/.bashrc file, for instance. Close and reopen your terminal and then use it.

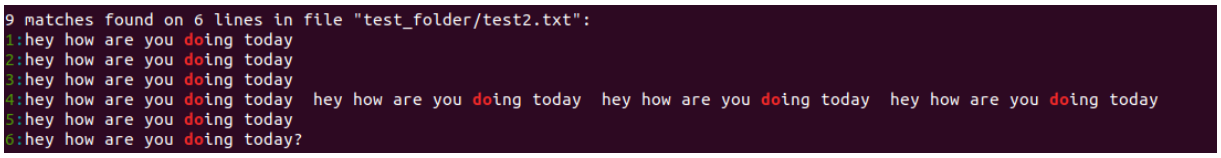
Example:

Replace `do` with `bo` so that "doing" becomes "boing" (I know, we should be fixing spelling errors not creating them :)):

```
$ gs_replace_str "do" "bo" test_folder/test2.txt

9 matches found on 6 lines in file "test_folder/test2.txt":
1:hey how are you doing today
2:hey how are you doing today
3:hey how are you doing today
4:hey how are you doing today  hey how are you doing today  hey how are you
doing today  hey how are you doing today
5:hey how are you doing today
6:hey how are you doing today?
$SHLVL:3
```

Screenshot of the output, to show the matched text being highlighted in red:



```
9 matches found on 6 lines in file "test_folder/test2.txt":
1:hey how are you doing today
2:hey how are you doing today
3:hey how are you doing today
4:hey how are you doing today  hey how are you doing today  hey how are you doing today  hey how are you doing today
5:hey how are you doing today
6:hey how are you doing today?
```

References:

1. https://superuser.com/questions/339522/counting-total-number-of-matches-with-grep-instead-of-just-how-many-lines-match/339523#339523
2. https://stackoverflow.com/questions/12144158/how-to-check-if-sed-has-changed-a-file/61238414#61238414

Share Improve this answer Follow

edited Dec 17, 2021 at 18:56

answered Apr 15, 2020 at 21:12

Gabriel Staples
1,408 16 19

0

I just wrote a wrapper around RipGrep to give it this feature. I call it `rgr` for RipGrep Replace, since it will do the find-and-replace feature on your disk. See installation instructions at the top of the file here: https://github.com/ElectricRCAircraftGuy/eRCaGuy_dotfiles/blob/master/useful_scripts/rg_replace.sh

Example usage from `rgr -h`:

```
EXAMPLE USAGES:

    rgr foo -r boo
        Do a *dry run* to replace all instances of 'foo' with 'boo' in this
folder and down.
    rgr foo -R boo
        ACTUALLY REPLACE ON YOUR DISK all instances of 'foo' with 'boo' in this
folder and down.
    rgr foo -R boo file1.c file2.c file3.c
        Same as above, but only in these 3 files.
    rgr foo -R boo -g '*.txt'
        Use a glob filter to replace on your disk all instances of 'foo' with
'boo' in .txt files
        ONLY, inside this folder and down. Learn more about RipGrep's glob
feature here:
        https://github.com/BurntSushi/ripgrep/blob/master/GUIDE.md#manual-
filtering-globs
    rgr foo -R boo --stats
        Replace on your disk all instances of 'foo' with 'boo', showing detailed
statistics.
```

Share Improve this answer Follow

answered Jan 4 at 7:24

Gabriel Staples
1,408 16 19

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.