The Basics of Using the Sed Stream Editor to Manipulate Text in Linux

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=208*

Introduction

The sed stream editor is a text editor that performs editing operations on information coming from standard input or a file. Sed edits line-by-line and in a non-interactive way.

This means that you make all of the editing decisions as you are calling the command and sed will execute the directions automatically. This may seem confusing or unintuitive, but it is a very powerful and fast way to transform text.

This tutorial will cover some basics operations and introduce you to the syntax required to operate this editor. You will almost certainly never replace your regular text editor with sed, but it will probably become a welcomed addition to your text editing toolbox.

Basic Usage

In general, sed operates on a stream of text that it reads from either standard input or from a file.

This means that you can send the output of another command directly into sed for editing, or you can work on a file that you've already created.

You should also be aware that sed outputs everything to standard out by default. That means that, unless redirected, sed will print its output to the screen instead of saving it in a file.

The basic usage is:

```
sed [options] commands [file-to-edit]
```

We will copy some files into our home directory to practice some editing.

```
cd
cp /usr/share/common-licenses/BSD .
cp /usr/share/common-licenses/GPL-3 .
```

Let's use sed to view the contents of the BSD license file we copied.

Since we know that sed sends its results to the screen by default, we can use it as a file reader by passing it no editing commands. Let's try it:

```
sed '' BSD

Copyright (c) The Regents of the University of California.
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
```

```
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
...
...
```

This works because the single quotes contain the editing commands we are passing to sed. We passed it nothing, so it just printed each line it received to standard output.

We will demonstrate how sed can use standard input by piping the output of the "cat" command into sed to produce the same result.

```
cat BSD | sed ''

Copyright (c) The Regents of the University of California.
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
. . .
. . .
```

As you can see, we can operate on files or streams of text (as is produced when piping output with the "|" character) just as easily.

Printing Lines

In the previous example, we saw that input passed into sed without any operations would print the results directly to standard output.

We will now explore sed's explicit "print" command, which is specified by using the "p" character within single quotes.

```
sed 'p' BSD

Copyright (c) The Regents of the University of California.
Copyright (c) The Regents of the University of California.
All rights reserved.
All rights reserved.
Redistribution and use in source and binary forms, with or without
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
```

```
modification, are permitted provided that the following conditions
are met:
are met:
. . .
. . .
```

You can see that sed has printed each line twice now. This is because it automatically prints each line, and then we've told it to print explicitly with the "p" command.

If you examine how the output has the first line twice, followed by the second line twice, etc., you will see that sed operates line by line. It accepts a line, operates on it, and outputs the resulting text before repeating the process on the next line.

We can clean up the results by passing the "-n" option to sed, which suppresses the automatic printing:

```
sed -n 'p' BSD

Copyright (c) The Regents of the University of California.
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
. . .
. . .
```

We now are back to printing each line once.

Address Ranges

The examples so far can hardly be considered editing (unless we wanted to print each line twice...). Let's modify the output by only having sed print the first line.

```
sed -n '1p' BSD

Copyright (c) The Regents of the University of California.
```

By placing the number "1" before the print command, we have told sed the line number to operate on.

We can just as easily print five lines (don't forget the "-n").

```
sed -n '1,5p' BSD

Copyright (c) The Regents of the University of California.
All rights reserved.
Redistribution and use in source and binary forms, with or without
```

```
modification, are permitted provided that the following conditions
```

We've just given an address range to sed. If we give sed an address, it will only perform the commands that follow on those lines. In this example, we've told sed to print line 1 through line 5.

We could have specified this in a different way with by giving the first address and then using an offset to tell sed how many additional lines to travel:

```
sed -n '1,+4p' BSD
```

This will result in the same output, because we've told sed to start at line 1 and then operate on the next 4 lines as well.

If we want to print every other line, we can specify the interval after the "~" character. The following line will print every other line starting with line 1:

```
sed -n '1~2p' BSD

Copyright (c) The Regents of the University of California.
modification, are permitted provided that the following conditions
1. Redistributions of source code must retain the above copyright
2. Redistributions in binary form must reproduce the above copyright
   documentation and/or other materials provided with the distribution.
   may be used to endorse or promote products derived from this software
. . .
. . .
```

Deleting Text

We can easily perform text deletion where we previously were specifying text printing by changing the "p" command to the "d" command.

We no longer need the "-n" command because with the delete command, sed will print everything that is not deleted, which will help us see what's going on.

We can modify the last command from the previous section to make it delete every other line starting with the first. The result is that we should be given every line we were **not** given last time.

```
sed '1~2d' BSD

All rights reserved.
Redistribution and use in source and binary forms, with or without
are met:
   notice, this list of conditions and the following disclaimer.
   notice, this list of conditions and the following disclaimer in the
3. Neither the name of the University nor the names of its contributors
   without specific prior written permission.
. . .
. . .
```

It is important to note here that our source file is not being affected. It is still intact. The edits are output to our screen.

If we want to save our edits, we can redirect standard output to a file like so:

```
sed '1~2d' BSD > everyother.txt
```

If we open the file with cat, we will see the same output that we saw onscreen previously. Sed does not edit the source file by default for our safety.

We can change this behavior though by passing sed the "-i" option, which means perform edits in-place. This will edit the source file.

Let's try it by editing our "everyother.txt" file we just created, in-place. Let's further reduce the file by deleting every other line again:

```
sed -i '1~2d' everyother.txt
```

If you use cat again, you can see that the file has been edited.

As mentioned previously, the "-i" option can be **dangerous**! Thankfully, sed gives us the ability to create a backup file prior to editing.

To create a backup file prior to editing, add the backup extension directly after the "-i" option:

```
sed -i.bak '1~2d' everyother.txt
```

This will create a backup file with the ".bak" extension, and then edit the regular file in-place.

Substituting Text

Perhaps the most well-known use for sed is substituting text. Sed has the ability to search for text patterns using regular expressions, and then replace the found text.

Click here to learn about regular expressions if needed.

In its simplest form, you can change one word to another word using the following syntax:

```
's/old_word/new_word/'
```

The "s" is the substitute command. The three slashes (/) are used to separate the different text fields. You can use other characters to delimit the fields if it would be more helpful.

For instance, if we were trying to change a website name, using another delimiter would be helpful since URLs contain slashes. We'll use echo to pipe in an example:

```
echo "http://www.example.com/index.html" | sed 's_com/index_org/home_'

http://www.example.org/home.html
```

Do not forget the final delimiter, or sed will complain.

```
echo "http://www.example.com/index.html" | sed 's_com/index_org/home'

sed: -e expression #1, char 22: unterminated `s' command
```

Let's create a file to practice our substitutions on:

```
echo "this is the song that never ends
yes, it goes on and on, my friend
some people started singing it
not knowing what it was
and they'll continue singing it forever
just because..." > annoying.txt
```

Now let's substitute the expression "on" with "forward".

```
sed 's/on/forward/' annoying.txt

this is the sforwardg that never ends
yes, it goes forward and on, my friend
some people started singing it
not knowing what it was
and they'll cforwardtinue singing it forever
just because...
```

You can see a few notable things here. First, is that we are replacing patterns, not words. The "on" within "song" is changed to "forward".

The other thing to notice is that on line 2, the second "on" was not changed to "forward".

This is because by default, the "s" command operates on the first match in a line and then moves to the next line.

To make sed replace every instance of "on" instead of just the first on each line, we can pass an optional flag to the substitute command.

We will provide the "g" flag to the substitute command by placing it after the substitution set.

```
sed 's/on/forward/g' annoying.txt

this is the sforwardg that never ends
yes, it goes forward and forward, my friend
some people started singing it
not knowing what it was
and they'll cforwardtinue singing it forever
just because...
```

Now the substitute command is changing every instance.

If we *only* wanted to change the second instance of "on" that sed finds on each line, then we could use the number "2" instead of the "g".

```
sed 's/on/forward/2' annoying.txt

this is the song that never ends
yes, it goes on and forward, my friend
some people started singing it
not knowing what it was
and they'll continue singing it forever
just because...
```

If we only want to see which lines were substituted, we can use the "-n" option again to suppress automatic printing.

We can then pass the "p" flag to the substitute command to print lines where substitution took place.

```
sed -n 's/on/forward/2p' annoying.text

yes, it goes on and forward, my friend
```

As you can see, we can combine the flags at the end of the command.

If we want the search process to ignore case, we can pass it the "i" flag.

```
sed 's/SINGING/saying/i' annoying.txt

this is the song that never ends
yes, it goes on and on, my friend
some people started saying it
not knowing what it was
and they'll continue saying it forever
just because...
```

Referencing Matched Text

If we wish to find more complex patterns with regular expressions, we have a number of different methods of referencing the matched pattern in the replacement text.

For instance, if we want to match the from the beginning of the line to "at" we can use the expression:

```
sed 's/^.*at/REPLACED/' annoying.txt

REPLACED never ends
yes, it goes on and on, my friend
some people started singing it
REPLACED it was
and they'll continue singing it forever
```

```
just because...
```

You can see that the wildcard expression matches from the beginning of the line to the last instance of "at".

Since you don't know the exact phrase that will match in the search string, you can use the "&" character to represent the matched text in the replacement string.

This example shows how to put parentheses around the matched text:

```
sed 's/^.*at/(&)/' annoying.txt

(this is the song that) never ends
yes, it goes on and on, my friend
some people started singing it
(not knowing what) it was
and they'll continue singing it forever
just because...
```

A more flexible way of referencing matched text is to use escaped parentheses to group sections of matched text.

Every group of search text marked with parentheses can be referenced by an escaped reference number. For instance, the first parentheses group can be referenced with "\1", the second with "\2" and so on.

In this example, we'll switch the first two words of each line:

```
sed 's/\([a-zA-Z0-9][a-zA-Z0-9]*\) \([a-zA-Z0-9][a-zA-Z0-9]*\)/\2 \1/'
annoying.txt

is this the song that never ends
yes, goes it on and on, my friend
people some started singing it
knowing not what it was
they and'll continue singing it forever
because just...
```

As you can see, the results are not perfect. For instance, the second line skips the first word because it has a character not listed in our character set. Similarly, it treated "they'll" as two words in the fifth line.

Let's improve the regular expression to be more accurate:

```
sed 's/\([^ ][^ ]*\) \([^ ][^ ]*\)/\2 \1/' annoying.txt

is this the song that never ends
it yes, goes on and on, my friend
people some started singing it
knowing not what it was
they'll and continue singing it forever
because... just
```

This is much better than last time. This groups punctuation with the associated word.

Notice how we repeat the expression inside the parentheses (once without the "*" character, and then once with it). This is because the "*" character matches the character set that comes before it zero or more times.

This means that the match with the wildcard would be considered a "match" even if the pattern is not found.

To ensure that it is found at least once, we must match it once without the wildcard before employing the wildcard.