

## sed-tutorial - a small tutorial about sed

### DESCRIPTION

sed is a non-interactive text editor that comes with UNIX since Version 7 AT&T UNIX. It's main purpose is to be used in scripts.

### SIMPLE TEXT EDITING

sed works as a filter, which makes it particularly useful for scripts. sed works line oriented. The most simple commands are a pattern and an action. If no pattern is given, the action is applied to all lines, otherwise it is applied only to lines matching the pattern.

### Line-oriented Patterns

A sample application of sed would be to delete the first 10 lines of stdin and echo the rest to stdout:

```
sed -e '1,10d'
```

The `-e` tells sed to execute the next command line argument as sed program. Since sed programs often contain regular expressions, they will often contain characters that your shell interprets, so you should get used to put all sed programs in single quotes so your shell won't interpret the sed program. In this case, nothing bad would have happened, but since almost all other examples will contain meta-characters, you really should get used to quoting your programs. This simple sed program contains a pattern (``1,10``) and an action (``d``). What `sed(1)` does is apply all actions whose pattern match and finally print the line unless the action was ``d``. If you don't want `sed(1)` to print each line by default, you can give sed the `-n` option. Then only lines that you print explicitly (with the ``p`` action) appear on stdout.

If we wanted to print only the first ten lines, we would have deleted all the lines starting with 11:

```
sed -e '11,$d'
```

Note that `$` is the last line. Because sed processes the input line by line, it does not keep the whole input in memory. This makes sed very useful for processing large files, but it has its drawbacks, too. For example, we can't use `sed -e '$-10,$d'`, since sed doesn't know `$` before the end of file, so it doesn't know where `$-10` is. This is a major problem, and it limits sed's usefulness, but sed still has a large number of appliances.

Another way to get only the first 10 lines is to use the `-n` option:

```
sed -n -e '1,10p'
```

If we want to delete only one line, the pattern can be `'10,10'` or simple `'10'`.

### More Than One Command

Commands in sed programs are separated by new lines. So if we wanted to delete the lines 1 to 4 and 6 to 9, we could use:

```
sed -e '1,4d
6,9d'
```

Another possibility is to use the `-e` option more than once:

```
sed -e '1,4d' -e '6,9d'
```

That's why we used the `-e` option all the time. In fact, you can omit it if you have only one command in your program. But you should get used to the `-e` option, so you won't have to add it if you want to extend your program later on.

## Regular Expression Oriented Patterns

Often, we don't know the numbers of the lines we want to delete. A good example is a log file. Log files tend to grow until they become too large to handle. Let's assume that you have a large log file called `log` which contains thousands of lines. Now you want to delete all the lines that contain the word `debug`:

```
sed -e '/debug/d' log
```

This works just like `grep -v debug`.

## A Slightly More Complex Example

We are still working with the large log file. Now we not only want to delete lines with the word `debug`, but we only want lines that contain `foo`. The traditional way to handle this would be:

```
grep 'foo' log | grep -v debug
```

Note that this spawns two `grep` processes. The `sed` equivalent would be:

```
sed -n -e '/debug/d' -e '/foo/p'
```

You might wonder why lines with `debug` aren't printed if they contain `foo`. The answer is that the `d` action skips the rest of the patterns and actions, too, it does not just inhibit the print in the end (which is inhibited here due to the `-n`, anyway).

## Putting sed Programs Into Files

Now that your programs are getting a little more advanced, you might want to put them in script files instead of using the command line. To tell `sed(1)` about your program file, you use the `-f` option:

```
sed -f program.sed
```

There is a kludge in `sed` that allows you to set the `-n` option from within your `sed` program if you use `#n` as the first line in your program file. From now on I will assume that you run the examples through `sed -f`.

## Inserting Text

You can insert text with the `a` and `i` actions. The syntax is:

```
10i
string to be inserted
```

The difference between `i` and `a` is that `i` inserts before the current line and `a` appends after the current line. So `1i` will insert before the first line and `$a` will append after the last line.

## Replacing the current line

You can replace the current line with the ```c``` action. The syntax is like ```i``` and ```a```:

```
10c
new contents for line 10
```

## Printing The Current Line Visually Unambiguously

The ```l``` action is very useful when editing files with nonprintable characters. It prints the current line visually unambiguously. For example, long lines are broken, but the lines end with a `^` to show that they were broken. Normal backslashes in the text are escaped, too, tabs are replaced with `^I` and nonprintable characters are printed as escaped three-digit octal numbers. This example is quite useful as shell alias:

```
sed -n -e 'l'
```

## Aborting Processing

The ```q``` action branches to the end of the script and ends the script processing after this line. So, yet another way of printing the first 10 lines would have been:

```
sed -e '10q'
```

## REGULAR EXPRESSION SUBSTITUTION

The ```s/pattern/replacement/[flags]``` action is the most often used `sed(1)` action. In fact, most `sed` programs consist only of substitute commands, since this is so immensely useful. The regular expression pattern is substituted by the replacement string (which can contain several special symbols). The most basic substitution would be

```
sed -e 's/foo/bar/'
```

which would just change the string ```foo``` to ```bar```.

From <http://www.ceri.memphis.edu/computer/docs/unix/sed.htm>

A handy set of one-liners for `sed`