For example, replace all three-digit numbers with another specified number in a file, as follows:

```
$ cat sed_data.txt
11 abc 111 this 9 file contains 111 11 88 numbers 0000

$ sed -i 's/\b[0-9]\{3\}\b/NUMBER/g' sed_data.txt
$ cat sed_data.txt
11 abc NUMBER this 9 file contains NUMBER 11 88 numbers 0000
```

The preceding one-liner replaces three-digit numbers only. `\b[0-9]\{3\}\b` is the regular expression used to match three-digit numbers. `[0-9]` is the range of digits; that is, from `0` to `9`. `{3}` is used for matching the preceding character thrice. `\` in `\{3\}` is used to give a special meaning for `{` and `}`. `\b` is the word boundary marker.

It's a useful practice to first try the `sed` command without `-i` to make sure your regex is correct, and once you are satisfied with the result, add the `-i` option to actually make changes to the file. Alternatively, you can use the following form of `sed`:

```
sed -i .bak 's/abc/def/' file
```

In this case, `sed` will not only perform the replacement on the file, but it will also create a file called `file.bak`, which will contain the original contents.

## Matched string notation (&)

In `sed`, we can use `&` as the matched string for the substitution pattern, in such a way that we can use the matched string in the replacement string.

For example:

```
$ echo this is an example | sed 's/\w\+/[&]/g'
[this] [is] [an] [example]
```

Here, regex `\w\+` matches every word. Then, we replace it with `[&]`. `&` corresponds to the word that is matched.

## Substring match notation (\1)

`&` corresponds to the matched string for the given pattern. We can also match the substrings of the given pattern. Let's see how to do it.

```
$ echo this is digit 7 in a number | sed 's/digit \([0-9]\)/\1/'
this is 7 in a number
```

The preceding command replaces `digit 7` with `7`. `\(pattern\)` is used to match the substring. The pattern is enclosed in `()`, and is escaped with slashes. For the first substring match, the corresponding notation is `\1`; for the second, it is `\2`, and so on. Go through the following example with multiple matches:

```
$ echo seven EIGHT | sed 's/\([a-z]\+\) \([A-Z]\+\)/\2 \1/'
EIGHT seven
```

`([a-z]\+\)` matches the first word, and `\([A-Z]\+\)` matches the second word. `\1` and `\2` are used for referencing them. This type of referencing is called **back referencing**. In the replacement part, their order is changed as `\2 \1` and, hence, it appears in reverse order.

## Combination of multiple expressions

The combination of multiple `sed` using a pipe can be replaced as follows:

```
sed 'expression' | sed 'expression'
```

The preceding command is equivalent to the following:

```
$ sed 'expression; expression'
```

Or:

```
$ sed -e 'expression' -e expression'
```

For example,

```
$ echo abc | sed 's/a/A/' | sed 's/c/C/'
AbC
$ echo abc | sed 's/a/A/;s/c/C/'
AbC
$ echo abc | sed -e 's/a/A/' -e 's/c/C/'
AbC
```

## Quoting

Usually, it is seen that the `sed` expression is quoted by using single quotes. But, double quotes can also be used. Double quotes expand the expression by evaluating it. Using double quotes are useful when we want to use a variable string in a `sed` expression.

For example:

```
$ text=hello
$ echo hello world | sed "s/$text/HELLO/"
HELLO world
```

`$text` is evaluated as `hello`.