Notes on `sed`

B.S. Bharath Saiguhan

April 4, 2021

Substitutions

Done with the `s` command, which requires:

- search regex/string
- replacement string
- file on which to act & other optional flags

like so: `sed 's/<regex>/<replacement>/<clause> /path/to/file`.

Although `/` is the standard delimiter, any printable character may be used for convenience, say if you are going to work with file pathnames as an example. If replacement is empty, then sed simply 'deletes' the patterns matching the regex.

Some of the most commonly used flags are:

- `-e` followed by one or more `editing-commands`. Necessary when running multiple commands.
- `-f` followed by a `script-file`. Simply read commands from the file.
- `-n`. Suppresses the usual printing of the final modified line. In this case, use `p` explicitly to print. To do this in a script, put `#n` anywhere in the script file. Note that after this line you will have to use a `p` clause to tell sed to print from the pattern space.

sed also understands backreferences. Use `&` to substitute the entire matched pattern at that point, which means that `'s/Atlanta/&, the capital of the South/'` replaces all occurences of "Atlanta" with "Atlanta, the capital. . . ".

The `g` clause given to sed at the end of an editing command (i.e. after the third delimiter) replaces all pattern matches *globally*. Similarly, the `n` clause, where n is a number, replaces the n-th occurence only.

sed remembers the regexs given to it, and if you don't give it a regex in the editing command, it will repeat the last regex.

To substitute only on specific lines, prefix the `editing-command` with an `address`, which can be :

- a regex. Substitutions will only be performed on those lines with matching patterns. Eg: `sed /oldfunc/ s/$/# XXX: migrate to newfunc/` will annotate some code. If you leave the regex empty, then the address regex will be used again (as is usual for an empty regex).
- the last line (via `$`). Hence `sed -n '$p' "$1"` in a script prints the last line of the file given as the first arg.
- a line number.
- ranges. Here you can either specify in terms of line numbers such as `sed -n '10,42p' foo.xml` or in terms of regex matches such as `sed '/foo/,/bar/ s/baz/quux/g'`.
- negated regex. Eg: `'/used/!s/new/used/g'` replaces new with used on all lines which don't match with `used`. Note that for maximum portability DON'T include a whitespace after the `!`.

Note: `sed 10q file` reads *till* the 10th line and quits immediately once it reads that line.

Note: In `sed -n '\:tolstoy: s;;Tolstoy;g' /etc/passwd`, the `\` makes the `:` delimit the search pattern, and the `;` acts as a delimiter for the actual substitute command.

Note: regexs match the *longest, leftmost substring*, and naturally, a match of the NULL string is longer than no match at all. Carefully analyse the following to understand what we just said:

- `echo Tolstoy is worldly | sed 's/T.*y/Camus'`
  - `output >> Camus`
- `echo Tolstoy is worldly | sed 's/T[[:alpha:]]*y/Camus'`
  - `output >> Camus is worldly.`
- `echo abc | sed 's/b*/1'`
  - `output >> 1abc`
- `echo abc | sed 's/b*/1/g'`
  - `output >> 1a1c1`