

# Stream editor and regexps

Sicurezza (CT0539) 2020-21  
Università Ca' Foscari Venezia

Riccardo Focardi

[www.unive.it/data/persone/5590470](http://www.unive.it/data/persone/5590470)  
[secgroup.dais.unive.it](http://secgroup.dais.unive.it)



# Stream editor sed

**sed** is a simple but powerful Unix tool to **filter** and **transform** text

sed is invoked as **sed SCRIPT INPUTFILE ...**

**Example:** replace hello with world in file input.txt

```
sed 's/hello/world/' input.txt           (output goes to stdout)  
sed -i 's/hello/world/' input.txt       (in-place, modifies input.txt)
```

The following commands are equivalent:

```
sed 's/hello/world/' input.txt > output.txt  
sed 's/hello/world/' < input.txt > output.txt  
cat input.txt | sed 's/hello/world/' - > output.txt (with or without -)
```

# sed commands

A sed **program** consists of one or more sed commands

sed commands follow this syntax:

[ **addr** ] **X** [ **options** ]

- **X** is a single-letter sed command
- [ **addr** ] line address: single line number, regexp, or range of lines
- [ **options** ] for some commands

**Delete:** command **d** deletes lines

sed '**1d**' input.txt : deletes first line

sed '**1,3d**' input.txt : deletes first three lines

**Print:** command **p** prints lines

sed '**1p**' filename : prints first line

**NOTE:** line is printed twice

# Examples

```
$ sed '1d' test.txt  
line 2  
line 3  
line 4
```

```
$ sed '1,3d' test.txt  
line 4
```

```
$ sed '1p' test.txt  
line 1  
line 1  
line 2  
line 3  
line 4
```

Command line option **-n** tells sed not to print lines unless they are printed explicitly

## Example:

```
$ sed -n '1,2p' test.txt  
line 1  
line 2
```

# Substitution

Command **s** substitutes strings

```
sed 's/hello/hi/' input.txt
```

By default substitution happens **once**  
for each line

```
$ cat input.txt  
hello guys hello everyone
```

```
$ sed 's/hello/hi/' input.txt  
hi guys hello everyone
```

Option **g** makes substitution global

```
$ sed 's/hello/hi/g' input.txt  
hi guys hi everyone
```

Option **i** makes search case  
insensitive and a number specifies  
which occurrence should be replaced

```
$ sed 's/HELLO/hi/i2' input.txt  
hello guys hi everyone
```

# Substitution (more examples)

It is possible to use a **custom separator** with substitution command

```
sed 's:hello:hi:' input.txt
```

As for d and p, it is possible to indicate which **lines** should be examined:

```
sed '6,7s/hello/hi/' input.txt
```

only applies to lines 6 and 7

Only **print** rows that match string:

```
sed -n '/hello/p' input.txt
```

**Delete** rows that match string:

```
sed '/hello/d' input.txt
```

Apply a **mapping**:

```
sed 'y/abc/ABC/'
```

replaces each occurrence of a,b,c with A, B, C, respectively.

# Regular expressions

Regular expressions are **patterns** representing sets of strings

Useful to perform **advanced searches** in which it is necessary to find strings with a particular **structure**

Programs **grep** and **sed** both support regular expressions

**^** is the beginning of line

**Example:** `ls -al | grep '^d'`

matches all **directory** files in the current directory (**d** is the flag that indicates a directory file)

If we omit the **^** symbol, grep will match all lines containing a **d**, not necessarily in the first position

# Regular expressions (2)

**\$** indicates end of the line

**.** represents a single character

**Example:** `grep '.ino'` will match names such as Nino, Pino, Gino, ...

**c\*** represents a possibly empty, arbitrary number of occurrences of character `c`

**Example:** `grep 'smart *card'`

(smartcard, smart card, smart card)

**.\*** matches an arbitrary number of arbitrary characters

**c\+** one or more occurrences of `c`

**c\?** zero or one occurrences of `c`

Notice that `+` and `?` need to be **escaped** prepending a backslash `\` character



# Regular expressions (3)

Note on **escaping**:

- To find a special character like `.` or `*` it is enough to **escape** it with a backslash `\` character
- For characters that needs to be escaped in regular expression such as `\+` and `\?` it is instead enough to **remove the backslash**

`[0123456789]` or `[0-9]`

represents all digits from 0 to 9

**Example:** `[0-9]\+` is a decimal number of arbitrary length

`[^0-9]` anything that is **not** a digit

**Example:** `grep '^[^0-9]*$'` all lines that do not contain digits

# Classes

**[:alnum:]** Alphanumeric [a-z A-Z 0-9]

**[:alpha:]** Alphabetic [a-z A-Z]

**[:blank:]** Blank characters (spaces or tabs)

**[:cntrl:]** Control characters

**[:digit:]** Numbers [0-9]

**[:graph:]** Printable characters (excluding spaces)

**[:lower:]** Lowercase letters [a-z]

**[:print:]** Printable characters (including spaces)

**[:punct:]** Punctuation characters

**[:space:]** Spaces (including \t \n)

**[:upper:]** Uppercase letters [A-Z]

**[:xdigit:]** Hex digits [0-9 a-f A-F]

# Regular expressions in sed

**sed** supports regular expressions

For substitutions, it is useful to **refer** to the matched text. This can be done in two ways:

**&** is substituted with the **whole** matched string

**Example:** add world after hello  
sed 's/hello/& world/g'

**Back references and brackets:** refer to portions of the matched text

**Ex.:** extract the name from a letter

```
's/Dear \([^ ]*\) .*$/Name = \1/g'
```

Notice that the pattern we refer to is surrounded by **\(** and **\)**, while the reference to it is **\1**

Use **\2**, **\3**, ... for next references

# Exercises

Files for exercises are available at  
`/home/rookie/Shell/`  
in the [testbed](#) host

# Exercise 1: pretty printing

Given a list of telephone numbers of the form **123456789** use **sed** to rewrite them as **(123)456-789**

Anything in the wrong format should be **left unmodified**.

```
$ cat numeri.txt
```

```
123456789
```

```
392948291
```

```
321582923
```

```
321904984
```

```
Not a number
```

```
hello
```

```
$ sed ... numeri.txt
```

```
(123)456-789
```

```
(392)948-291
```

```
(321)582-923
```

```
(321)904-984
```

```
Not a number
```

```
hello
```

# Exercise 2: break ROT13

The following text has been encrypted by replacing each letter with the one 13 positions ahead in the alphabet (modulo 26) aka [ROT13](#)

Break it with **sed**!

**Hint 1:** Check out command **y**

```
jryy qbar thlf, lbh oebxr n
      pvcure jvgu frq!
```

# Exercise 3: filename conversion

Use sed to select and convert all file names with suffix `.html` given as output by `ls` into **capital letters** with suffix `.HTM`

Non-matching files should be **omitted**

**Hint 1:** Check out command **y**

**Hint 2:** You can concatenate commands as: `sed 'cmd1 ;cmd2'`

```
$ ls
document.pdf
myPage.html
test.html
```

```
$ ls | sed ...
MYPAGE.HTM
TEST.HTM
```

# Exercise 4: data extraction

Use **sed** to extract full user names (5th field) from **/etc/passwd/**

```
$ sed ... /etc/passwd
root
daemon
bin
...
Mailing List Manager
ircd
Gnats Bug-Reporting System (admin)
nobody
```



# Extra: GNU extensions

There are some handy GNU extensions that allows for shorter regexps (do not work in BSD unix)

`c\{n\}` repeats `c` `n` times

**Example:**

```
[[:digit:]]\{10\}
```

is a 10 digits number

`\L` and `\U` in `s` commands convert to lowercase and uppercase, respectively

**Example:**

```
$ cat input.txt  
hello guys hello everyone
```

```
$ sed 's/hello/\U&/g' input.txt  
HELLO guys HELLO everyone
```