CS 241: Systems Programming
Lecture 24. Regular Expressions II

Spring 2024
Prof. Stephen Checkoway

From last time: Extended regex

`.`	any char
`*`	zero or more
`+`	one or more
`?`	zero or one
`^`	start of a line
`$`	end of the line
`[ ]`	one of the chars
`{m,n}`	at least `m`, but at most `n`
`( )`	group
`|`	alternation

`\d`	digits
`\D`	nondigit
`\w`	word
`\W`	nonword
`\s`	space
`\S`	nonspace

Enhanced regex

char classes (used inside `[ ]`):

- `[:alpha:]`
- `[:digit:]`
- `[:xdigit:]`
- `[:space:]`
- etc.

sed(1) – stream editor

Usage: `$ sed [OPTIONS] command file`

- if no file, use **`stdin`**
- original file is not altered unless `-i` option is used
- `-E` option uses extended (modern) regular expressions
- multiple commands can be given using `-e command`
- `-n` option causes sed to not print each line

Sed as a regex find & replace

`$ sed 's/regex/replacement/' file`

- For each line of `file`, find the first portion of the line that matches `regex` and replace it with `replacement`

`$ sed 's/regex/replacement/g' file`

- For each line of `file`, find each portion of the line that matches `regex` and replace them all with `replacement`

Example: Replace the first "colour" with "color" in a file or stdin

-
  ```
  $ echo 'I like the colour blue.' | sed 's/colour/color/'
  I like the color blue.
  ```

Sed commands

Command format: `[address[,address]]function[arguments]`

- `addresses` are optional

Addresses are

- line number
- **`$`** is the last line of input
- **`/regex/`** lines matching the regex

Functions are applied to

- each line of input if no addresses are given
- each line of input matching the `address` if one is given, or
- between the two `addresses` (inclusive) if two are given

Sed functions

Functions

- `d` – delete line
- `s` – substitute string
- `p` – print line
- and many others (check the man page)

Sed print/delete examples

Sed print/delete examples

```
sed 'd' lines.txt
```

- delete all lines

Sed print/delete examples

```
sed 'd' lines.txt
```

- delete all lines

```
sed '2d' lines.txt
```

- delete second line

Sed print/delete examples

```
sed 'd' lines.txt
```

- delete all lines

```
sed '2d' lines.txt
```

- delete second line

```
sed -e '1,5d' -e '7d' lines.txt
```

- delete first 5 lines and line 7

Sed print/delete examples

```
sed 'd' lines.txt
```

- delete all lines

```
sed '2d' lines.txt
```

- delete second line

```
sed -e '1,5d' -e '7d' lines.txt
```

- delete first 5 lines and line 7

```
sed '/^#/d' lines.txt
```

- delete all lines starting with an # sign

Sed print/delete examples

```
sed 'd' lines.txt
```

- delete all lines

```
sed '2d' lines.txt
```

- delete second line

```
sed -e '1,5d' -e '7d' lines.txt
```

- delete first 5 lines and line 7

```
sed '/^#/d' lines.txt
```

- delete all lines starting with an # sign

```
sed -n '/.sh$/p' lines.txt
```

- only print lines ending in .sh

Sed print/delete examples

```
sed 'd' lines.txt
```

- delete all lines

```
sed '2d' lines.txt
```

- delete second line

```
sed -e '1,5d' -e '7d' lines.txt
```

- delete first 5 lines and line 7

```
sed '/^#/d' lines.txt
```

- delete all lines starting with an # sign

```
sed -n '/.sh$/p' lines.txt
```

- only print lines ending in .sh

```
sed -n '/^begin/,/^end/p' lines.txt
```

Sed print/delete examples

```
sed 'd' lines.txt
```

- delete all lines

```
sed '2d' lines.txt
```

- delete second line

```
sed -e '1,5d' -e '7d' lines.txt
```

- delete first 5 lines and line 7

```
sed '/^#/d' lines.txt
```

- delete all lines starting with an # sign

```
sed -n '/.sh$/p' lines.txt
```

- only print lines ending in .sh

```
sed -n '/^begin/,/^end/p' lines.txt
```

- only print lines between begin and end markers

Sed substitution

`s/regex/replacement/flags`

- The first regex match is replaced with the replacement
- Groups `( )` are called captures and can be referred to by number in the replacement: `s/Hello (\w+)!/Goodbye \1!/`

Flags

- `N` Substitution only the Nth match, e.g., `s/regex/replace/3`
- `g` Replace all matches in the line, not just the first
- `p` Print the line if a substitution was performed (often used with -n)
- `w file` Append the line to `file`

more sed examples

more sed examples

```
sed 's/foo/bar/' lines.txt
```

- replace the first foo with bar on each line (foofoo -> barfoo)

more sed examples

```
sed 's/foo/bar/' lines.txt
```

- replace the first foo with bar on each line (foofoo -> barfoo)

```
sed 's/foo/bar/g' lines.txt
```

- replace each foo with bar on every line (foofoo -> barbar)

more sed examples

```
sed 's/foo/bar/' lines.txt
```

- replace the first foo with bar on each line (foofoo -> barfoo)

```
sed 's/foo/bar/g' lines.txt
```

- replace each foo with bar on every line (foofoo -> barbar)

```
sed -e '1,5s/foo/bar/g' -e '7d' lines.txt
```

- replaces each foo with bar on lines 1-5 and deletes line 7

more sed examples

```
sed 's/foo/bar/' lines.txt
```

- replace the first foo with bar on each line (foofoo -> barfoo)

```
sed 's/foo/bar/g' lines.txt
```

- replace each foo with bar on every line (foofoo -> barbar)

```
sed -e '1,5s/foo/bar/g' -e '7d' lines.txt
```

- replaces each foo with bar on lines 1-5 and deletes line 7

```
sed -E 's/(a+)(b+)/\2\1/' lines.txt
```

- flips first adjacent groups of a and b characters (`qaaabt` -> `qbaaat`)

more sed examples

```
sed 's/foo/bar/' lines.txt
```

- replace the first foo with bar on each line (foofoo -> barfoo)

```
sed 's/foo/bar/g' lines.txt
```

- replace each foo with bar on every line (foofoo -> barbar)

```
sed -e '1,5s/foo/bar/g' -e '7d' lines.txt
```

- replaces each foo with bar on lines 1-5 and deletes line 7

```
sed -E 's/(a+)(b+)/\2\1/' lines.txt
```

- flips first adjacent groups of a and b characters (`qaaabt` -> `qbaaat`)

```
sed -n -e '/^begin/,/^end/s/foo/bar/gp' lines.txt
```

- changes all foo to bar between begin & end, then prints just those lines

What is the sed expression to delete all instances of the string `" newfangled"` from from the input? (There's a space before the n.)

```
A. sed -E '/ newfangled/d'

B. sed -E 'd/ newfangled/'

C. sed -E 's/ newfangled/d/'

D. sed -E 's/ newfangled//'

E. sed -E 's/ newfangled//g'
```

What is the sed command that swaps the first two words separated by a space in each line?

`\w` matches a "word" character
`\W` matches a "nonword" character
+ means 1 or more

A. `sed -E 's/(\w+) (\w+)/\2 \1/'`

B. `sed -E 's/(\W+) (\W+)/\2 \1/'`

C. `sed -e 's/(\w+) (\w+)/\2 \1/'`

D. `sed -e 's/\(w+\) \(\w+\)/\2 \1/'`

Other software

less(1)

- search (type a /) searches for a regex

vim(1)

- search (type a / in command mode) searches for a basic regex
- substitution `:[range] s/regex/replacement/flags`
- Vim's regex are strange, it has a "magic mode" and a "very magic mode"

Most other programmer-oriented editors have regex find and replace

- VS Code has a regex find and replace

Regex in Python

`re` module contains all of the regular expression functions and classes

`r = re.compile(pattern)` # returns an object that can be used to

- `r.match(string)` # tries to match the whole string
- `r.search(string)` # finds the first match

`re.match(pattern, string) and re.search(pattern, string)`

- Performs the compilation for you

`match()` and `search()` return a match object m (or **`None`**)

- `m.group()` returns the whole matched string
- `m.group(n)` returns the `n`th matched group

```python
#!/usr/bin/env python3
import re

# A primitive regex for URLs
url_regex = re.compile(r'([^:]+)://([^/]+)(/.*)?')

url = 'https://www.cs.oberlin.edu/classes/department-honors/'
match_obj = url_regex.match(url)
if match_obj:
    print("Scheme:", match_obj.group(1))
    print("Host:", match_obj.group(2))
    print("Path:", match_obj.group(3))
else:
    print("Not a match")
```

```python
#!/usr/bin/env python3
import re

# A primitive regex for URLs
url_regex = re.compile(r'([^:]+)://([^/]+)(/.*)?')

url = 'https://www.cs.oberlin.edu/classes/department-honors/'
match_obj = url_regex.match(url)
if match_obj:
    print("Scheme:", match_obj.group(1))
    print("Host:", match_obj.group(2))
    print("Path:", match_obj.group(3))
else:
    print("Not a match")
```

```
$ ./regex.py
Scheme: https
Host: www.cs.oberlin.edu
Path: /classes/department-honors/
```

Regex in C

```
#include <regex.h>
int regcomp(regex_t *restrict preg, char const *pattern,
            int cflags);
int regexec(regex_t const *preg, char const *string,
            size_t nmatch, regmatch_t pmatch[nmatch],
            int eflags);
void regfree(regex_t *preg);
```

Need to pass in 1 more regmatch_t object than capture groups

- `pmatch[0]` is whole match, `pmatch[n]` is nth matched group
- `pmatch[n].rm_so` is offset to the start of a match
- `pmatch[n].rm_eo` is offset to the first char after the match

```c
#include <regex.h>
#include <stdio.h>

int main(void) {
  regex_t url_regex;
  regmatch_t match[4];
  regcomp(&url_regex, "([^:]+)://([^/]+)(/.*)?", REG_EXTENDED);
  char const *url = "https://www.cs.oberlin.edu/classes/department-honors/";
  if (!regexec(&url_regex, url, 4, match, 0)) {
    int match_len = match[1].rm_eo - match[1].rm_so;
    printf("Scheme: %.*s\n", match_len, &url[match[1].rm_so]);
    match_len = match[2].rm_eo - match[2].rm_so;
    printf("Host: %.*s\n", match_len, &url[match[2].rm_so]);
    if (match[3].rm_so >= 0) {
      match_len = match[3].rm_eo - match[3].rm_so;
      printf("Path: %.*s\n", match_len, &url[match[3].rm_so]);
    }
  } else {
    puts("No match!");
  }
  regfree(&url_regex);
  return 0;
}
```

Regex in Rust

A bunch of regex crates

`regex` is the most popular

- Written by core Rust developers
- Almost 250 MM downloads (as of Map. 2024)!

```rust
use regex::Regex;

fn main() {
    let re = Regex::new("([^:]+)://([^/]+)(/.*)?").unwrap();
    let url = "https://www.cs.oberlin.edu/classes/department-honors/";
    if let Some(captures) = re.captures(url) {
        println!("Scheme: {}", captures.get(1).unwrap().as_str());
        println!("Host: {}", captures.get(2).unwrap().as_str());
        println!("Path: {}", captures.get(3).unwrap().as_str());
    } else {
        println!("Not a match");
    }
}
```

Regex in Bash

`[[ string =~ regex ]]`

- Returns `0` (true) if the string matches the regex
- Matches are stored in the Bash array variable `BASH_REMATCH`
- `${BASH_REMATCH[0]}` is the whole matched string
- `${BASH_REMATCH[n]}` is the `n`th matched group

```
url='https://www.cs.oberlin.edu/classes/department-honors/'
if [[ ${url} =~ ([^:]+)://([^/]+)(/.*)? ]]; then
  echo "Scheme: ${BASH_REMATCH[1]}"
  echo "Host: ${BASH_REMATCH[2]}"
  echo "Path: ${BASH_REMATCH[3]}"
else
  echo "No match!"
fi
```

Regex in Bash are tricky!

This doesn't work

```
course='CS 241'
if [[ ${course} =~ ([[:alpha:]]*) ([[:digit:]]*) ]]; then
```

Regex in Bash are tricky!

This doesn't work

```
course='CS 241'
if [[ ${course} =~ ([[:alpha:]]*) ([[:digit:]]*) ]]; then
```

```
if [[ ${course} =~ ([[:alpha:]]*) ([[:digit:]]*) ]]; then
^-- SC1009: The mentioned parser error was in this if expression.
   ^-- SC1073: Couldn't parse this test expression.
                                  ^-- SC1072: Expected test to end here
```

Regex in Bash are tricky!

So what about quoting the regex?

```
if [[ ${course} =~ '([[:alpha:]]*) ([[:digit:]]*)' ]]; then
```

Regex in Bash are tricky!

So what about quoting the regex?

```
if [[ ${course} =~ '([[:alpha:]]*) ([[:digit:]]*)' ]]; then
```

```
$ ./regex2.sh
No match!
```

Regex in Bash are tricky!

So what about quoting the regex?

```
if [[ ${course} =~ '([[:alpha:]]*) ([[:digit:]]*)' ]]; then
```

```
$ ./regex2.sh
No match!
```

```
if [[ ${course} =~ '([[:alpha:]]*) ([[:digit:]]*)' ]]; then
                   ^-- SC2076: Don't quote rhs of =~,
                               it'll match literally rather than as a regex.
```

Regex in Bash are tricky!

We need to escape the space

```
if [[ ${course} =~ ([[:alpha:]]*)\ ([[:digit:]]*) ]]; then
```

You can also put the regex in a variable

```
regex='([[:alpha:]]*) ([[:digit:]]*)'
if [[ ${course} =~ ${regex} ]]; then
```