Regular expressions

Operating System Practice

João Vicente Ferreira Lima

Universidade Federal de Santa Maria
jvlima@inf.ufsm.br
http://www.inf.ufsm.br/~jvlima

2021/2

Outline

Outline

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. A RE matches a single character or a set of characters. REs were widely used in Unix systems:

- Text editors such `ed`
- `grep` or `g/re/p`
- Lexical analysis to create a *token*, and analyse syntax of programming languages

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters.

1. $*$ matches any number of repeats, or *zero*

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters.

1. $*$ matches any number of repeats, or *zero*
2. . (*dot*) matches any one character, except a newline.

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters.

1. * matches any number of repeats, or *zero*
2. . (*dot*) matches any one character, except a newline.
3. ^ beginning of line, or negates

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters.

1. * matches any number of repeats, or *zero*
2. . (*dot*) matches any one character, except a newline.
3. ^ beginning of line, or negates
4. $ at the end, matches the end of line

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters.

1. `*` matches any number of repeats, or *zero*
2. `.` (*dot*) matches any one character, except a newline.
3. `^` beginning of line, or negates
4. `$` at the end, matches the end of line
5. `[...]` enclose a set of characteres

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters.

1. `*` matches any number of repeats, or *zero*
2. `.` (*dot*) matches any one character, except a newline.
3. `^` beginning of line, or negates
4. `$` at the end, matches the end of line
5. `[...]` enclose a set of characteres
   - `[xyz]` matches any one (x, y or z)
   - `[a-z0-9]` matches any single lowercase letter or any digit
   - `[^b-d]` matches any *except* those in the range `b` to `d`
6. `\` scapes a special character

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters.

1. `*` matches any number of repeats, or *zero*
2. `.` (*dot*) matches any one character, except a newline.
3. `^` beginning of line, or negates
4. `$` at the end, matches the end of line
5. `[...]` enclose a set of characteres
   - `[xyz]` matches any one (x, y or z)
   - `[a-z0-9]` matches any single lowercase letter or any digit
   - `[^b-d]` matches any *except* those in the range `b` to `d`
6. `\` scapes a special character
7. `\<...\>` mark work boundaries

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters.

1. `*` matches any number of repeats, or *zero*
2. `.` (*dot*) matches any one character, except a newline.
3. `^` beginning of line, or negates
4. `$` at the end, matches the end of line
5. `[...]` enclose a set of characteres
   - `[xyz]` matches any one (x, y or z)
   - `[a-z0-9]` matches any single lowercase letter or any digit
   - `[^b-d]` matches any *except* those in the range `b` to `d`
6. `\` scapes a special character
7. `\<...\>` mark work boundaries
8. `?` matches zero or one RE

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters.

1. `*` matches any number of repeats, or *zero*
2. `.` (*dot*) matches any one character, except a newline.
3. `^` beginning of line, or negates
4. `$` at the end, matches the end of line
5. `[...]` enclose a set of characteres
   - `[xyz]` matches any one (x, y or z)
   - `[a-z0-9]` matches any single lowercase letter or any digit
   - `[^b-d]` matches any *except* those in the range `b` to `d`
6. `\` scapes a special character
7. `\<...\>` mark work boundaries
8. `?` matches zero or one RE
9. `+` matches one or more RE

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters.

1. `*` matches any number of repeats, or *zero*
2. `.` (*dot*) matches any one character, except a newline.
3. `^` beginning of line, or negates
4. `$` at the end, matches the end of line
5. `[...]` enclose a set of characteres
   - `[xyz]` matches any one (x, y or z)
   - `[a-z0-9]` matches any single lowercase letter or any digit
   - `[^b-d]` matches any *except* those in the range `b` to `d`
6. `\` scapes a special character
7. `\<...\>` mark work boundaries
8. `?` matches zero or one RE
9. `+` matches one or more RE
10. `\{\}` indicate the number of ocurrences of RE

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters.

1. `*` matches any number of repeats, or *zero*
2. `.` (*dot*) matches any one character, except a newline.
3. `^` beginning of line, or negates
4. `$` at the end, matches the end of line
5. `[...]` enclose a set of characteres
   - `[xyz]` matches any one (x, y or z)
   - `[a-z0-9]` matches any single lowercase letter or any digit
   - `[^b-d]` matches any *except* those in the range `b` to `d`
6. `\` scapes a special character
7. `\<...\>` mark work boundaries
8. `?` matches zero or one RE
9. `+` matches one or more RE
10. `\{\}` indicate the number of ocurrences of RE
11. `( )` enclose a group of REs

Regular expressions

The main uses for Regular Expressions (REs) are text searches and string manipulation. An RE matches a single character or a set of characters.

1. `*` matches any number of repeats, or *zero*
2. `.` (*dot*) matches any one character, except a newline.
3. `^` beginning of line, or negates
4. `$` at the end, matches the end of line
5. `[...]` enclose a set of characteres
   - `[xyz]` matches any one (x, y or z)
   - `[a-z0-9]` matches any single lowercase letter or any digit
   - `[^b-d]` matches any *except* those in the range `b` to `d`
6. `\` scapes a special character
7. `\<...\>` mark work boundaries
8. `?` matches zero or one RE
9. `+` matches one or more RE
10. `\{\}` indicate the number of ocurrences of RE
11. `( )` enclose a group of REs
12. `|` (*or*) matches any of a set of alternate characters

Basic vs Extended Regular Expressions

- The meta-characters `?`, `+`, `{`, `|`, `(`, `)` lose their special meaning
- Instead use the backslashed versions `\?`, `\+`, `\{`, `\|`, `\(`, `\)`

GREP options

- `grep` - default grep
- `egrep` - or `grep -E`, uses extended regular expressions (ERE)
- `fgrep` - or `grep -F`, interprets patterns as fixed strings, not regular expressions
- `rgrep` - or `grep -r`, recursively reads all files under each directory

Regular expressions

Simple grep

```
$ grep 'daemon' /etc/passwd
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
colord:x:113:123:colord colour management daemon,,,:/var/lib/colord:/bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
```

Regular expressions

(caret) begin of line

```
$ grep '^daemon' /etc/passwd
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
```

$ end of line

```
$ grep 'bash$' /etc/passwd
root:x:0:0:root:/root:/bin/bash
jvlima:x:1000:1000:Joao,,,:/home/jvlima:/bin/bash
ddomenico:x:1001:1001:,,,:/home/ddomenico:/bin/bash
gfreytag:x:1002:1002:,,,:/home/gfreytag:/bin/bash
```

Regular expressions

```
$ grep 'Daemon' /etc/passwd
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
```

Lists with []

```
$ grep '[Dd]aemon' /etc/passwd
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false

grep '[aeiou]' /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
.........
```

Regular expressions

. (dot) one character

```
$ grep '^.[aeiou]' /etc/passwd|head -5
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
```

Regular expressions

```
$ grep '^.............................$' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Regular expressions

```
$ grep '^.............................$' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

{} repetition

```
$ grep '^.\{31\}$' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Regular expressions

```
$ grep '^.............................$' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

{} repetition

```
$ grep '^.\{31\}$' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

One or more

```
$ egrep '[0-9]{4,}' /etc/passwd
sync:x:4:65534:sync:/bin:/bin/sync
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
jvlima:x:1000:1000:Joao,,,:/home/jvlima:/bin/bash
```

Regular expressions

.* (AND)

```
$ egrep '^[a-z].*bash$' /etc/passwd
root:x:0:0:root:/root:/bin/bash
jvlima:x:1000:1000:Joao,,,:/home/jvlima:/bin/bash
ddomenico:x:1001:1001:,,,:/home/ddomenico:/bin/bash
gfreytag:x:1002:1002:,,,:/home/gfreytag:/bin/bash
```

Regular expressions

.* (AND)

```
$ egrep '^[a-z].*bash$' /etc/passwd
root:x:0:0:root:/root:/bin/bash
jvlima:x:1000:1000:Joao,,,:/home/jvlima:/bin/bash
ddomenico:x:1001:1001:,,,:/home/ddomenico:/bin/bash
gfreytag:x:1002:1002:,,,:/home/gfreytag:/bin/bash
```

OR

```
$ egrep '^(jvlima|root):' /etc/passwd
root:x:0:0:root:/root:/bin/bash
jvlima:x:1000:1000:Joao,,,:/home/jvlima:/bin/bash
```

Regular expressions

.* (AND)

```
$ egrep '^[a-z].*bash$' /etc/passwd
root:x:0:0:root:/root:/bin/bash
jvlima:x:1000:1000:Joao,,,:/home/jvlima:/bin/bash
ddomenico:x:1001:1001:,,,:/home/ddomenico:/bin/bash
gfreytag:x:1002:1002:,,,:/home/gfreytag:/bin/bash
```

OR

```
$ egrep '^(jvlima|root):' /etc/passwd
root:x:0:0:root:/root:/bin/bash
jvlima:x:1000:1000:Joao,,,:/home/jvlima:/bin/bash
```

NOT

```
$ egrep '^[^a-z]' /etc/passwd
_apt:x:105:65534::/nonexistent:/bin/false
```

Regular expressions

Backreference with () and \ 1

```
$ egrep '([a-z])([a-z])\2\1' /etc/passwd
_flatpak:x:117:125:Flatpak system-wide installation helper,,,:/nonexis...
```

Using Sed

```
$ echo "James Bond" | sed -E 's/(.*) (.*)/The name is \2, \1 \2./'
The name is Bond, James Bond.
```

Sed

Sed means stream editor derived from ed editor. It acts as a text editor for stdin data with stdout as target. The execution model is:

- read input line into pattern space
- apply commands to pattern space
- send pattern space to stdout

Invoking sed

```
$ sed 'p' example.txt

$ cat example.txt | sed p

$ sed 'p' < example.txt
```

Sed

The first command duplicates output since it prints the buffer and aplies the command. To supress the output we use -n option.

Sed output

```
$ sed 'p' example.txt
one
one
two
two
three
three

$ sed -n 'p' example.txt
one
two
three
```

Sed

We can address the text lines by number or/and by pattern.

Simple address

```
$ sed '2d' example.txt
one
three

$ sed '/three/d' example.txt
one
two
```

Sed

We can address the text lines by number or/and by pattern.

Intervals

```
$ sed '1,2d' example.txt
three

$ sed '1,/two/d' example.txt
three

$ sed '2,$d' example.txt
one

$ sed '/^$/d' example.txt # delete empty lines
```

Sed

Abort command

```
$ sed '10q'       # at line 10, stop

$ sed '/^$/q'     # stop at the first blank line
```

Sed

Invert logic

```
$ sed '1,10!d'      # do not delete lines 1 to 10

$ sed -n '11,$!p'   # do not print from line 11 to the end
```

Multiple commands

```
$ sed '5d;10d;/toto/d'  # remove lines 5, 10, and matching toto
```

Sed

One of the most used commands is the substituition s///

Invert logic

```
$ sed 's/toto/tata/' example.txt

$ sed 's:toto:tata:' example.txt # another format

$ echo "James Bond" | sed -E 's/(.*) (.*)/The name is \2, \1 \2./'
The name is Bond, James Bond.

$ sed -E '/IP/!d;s/^.*IP[ ](([0-9]{1,3}\.){3}[0-9]{1,3})\..*$/\1/'  \
    2021-09-10-traffic-analysis-exercise.txt
```