Regular Expressions

Daniel KNITTL-FRANK
p23687@fh-hagenberg.at

May 9, 2024

A Bit of Background

- Originated in 1951
- “Regular Events” by Stephen C. Kleene
- Equivalent to “finite automata” (Kleene’s theorem)
- Can be used to describe “regular languages”
- Type-3 grammar in Chomsky hierarchy

General

- Regular Expressions, RegExp, RegEx, RE
- Literals often marked with slashes: /regex/
- Slightly different syntax and features (BRE, ERE, PCRE)

BRE

Basic Regular Expressions

Simple Regex (1)

- A simple regex is /R/, which matches a single upper case R
  - RegEx
  - rrRRrr

Simple Regex (2)

- The regex /RE/ matches an R *immediately* followed by an E
  - RegEx [no match]
  - REgex
- /RE/ is composed of two regex /R/ and /E/

Matching Any Character: .

- The regex /./ matches *any* single character (except line breaks)
  - RegEx.
- /e./
  - regex

Escaping Meta Characters: \

- A backslash escapes the next character
  - Meta Characters become literals (\.)
  - Some literals become meta characters (\t)
- To match a dot, use /****./:
  - RegEx.

Repeating Regex: *

- Repeats the regex *immediately* preceding the quantifier
- 0..∞ repetitions
- *Greedy*, matches as much as possible
- /Hal*o/
  - Hao
  - Ha**l**o
  - Ha**ll**o
  - Ha**lll**o
- /a*b/
  - aa*bb

Character Classes: []

- Match a single character from a set
- Sets can contain ranges
- /gr**[ae]**y/
  - gr**a**yhound
  - gr**e**yhound
- /0x**[0-9a-f_]***/
  - 0x**1e_e7**

POSIX Character Classes: [[::]]

[:upper:]	uppercase letters
[:lower:]	lowercase letters
[:alpha:]	upper- and lowercase letters
[:digit:]	digits
[:xdigit:]	hexadecimal digits
[:alnum:]	digits, upper- and lowercase letters
[:punct:]	punctuation (all graphic characters except letters and digits)
[:blank:]	space and TAB characters only
[:space:]	blank (whitespace) characters
[:cntrl:]	control characters
[:graph:]	graphic characters (all characters which have graphic representation)
[:print:]	graphic characters and space

Negative Character Classes: [^]

- Match a single character not in a set
- Complementary sets can contain ranges
- /**[^**0-9**]**/
  - I am 42 years old
- /**[^**[:lower:]**]**/
  - I am 42 years old

Anchors: ^ $

- Anchor pattern at beginning or end of text/line
- /**^**a/
  - aaaaa
- /a**$**/
  - aaaaa

Examples

- Matching identifiers in a programming language:
  - [A-Za-z_][A-Za-z0-9_]*

ERE

Extended Regular Expressions

Repetitions: {m,n} (1)

- Repeat preceding regex
- {n}   exactly n repetitions
- {0,n} max n repetitions
- {n,} min n repetitions
- {m,n} min m, max n repetitions

Repetitions: {m,n} (2)

- /w**{3}**/
  - **www**.example.com
- /[0-9]**{0,3}**/
  - 0,42,1337
- /[0-9]**{3,}**/
  - 0,42,1337
- /[0-9]**{2,3}**/
  - 0,42,1337

Repetitions: +

- 1..∞ repetitions (equivalent to {1,})
- /Ha**l+**o/
  - Ha**l**o
  - Ha**ll**o
  - Ha**lll**o

Optional: ?

- Preceding expression is optional (equivalent to {0,1})
- /colo**u?**r/
  - colorful
  - colo**u**rful
- /http**s?**:/
  - http://example.com
  - http**s**://fhlug.at

Groups: ()

- Use parentheses to group regexs
- Modifiers apply to full group
- /(ma)+/
  - madame
  - mama

Alternatives: |

- /a|b/ matches either “a” or “b”
- /Mr**|**Mrs**|**Ms/
  - Mr Smith
  - Mrs Coulter
  - Ms Monique
- Must be grouped if a subexpression
- /SE**(vz|bb)**/
  - SE**vz**
  - SE**bb**
  - SExy [no match]

grep

Print lines that match a pattern

man grep

- grep 'PATTERNS' [files…] – BRE
- grep -E 'PATTERNS' – ERE
- grep -F 'PATTERNS' – fixed strings

grep

- grep '^root:' /etc/passwd
- seq 100 | grep '^42$'

Examples

Putting it all together

Time

- 12 hour format
  - `/[0-9]{2}:[0-9]{2} [ap]m/`
  - `/(0[0-9]|1[012]):[0-5][0-9] [ap]m/`
- 24 hour format
  - `/([01][0-9]|2[0-3]):[0-5][0-9]/`

Date

- /[0-9]{2}.[0-9]{2}.[0-9]{4}/
- /(0[1-9]|[12][0-9]|3[01]).(0[1-9]|1[012]).[0-9]{4}/

E-Mail validation

- /^[^]+@[^]+\.[^]+$/
- Minimal variant: /@/
- RFC822: http://www.ex-parrot.com/~pdw/Mail-RFC822-Address.html

Numbers

- Integers: `/[+-]?[0-9]+/`
- Decimals: `/[+-]?[0-9]+(\.[0-9]+)?/`
- Scientific notation: `/[+-]?[0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+(\.[0-9]+)?)?/`

Hex numbers

- /0x[0-9a-fA-F]+/

Hyperlinks

- /https?:\/\/[^]+/

Identifiers

- Identifiers in most programming languages:
- /[A-Za-z_][A-Za-z0-9_]*/

IPv4 Addresses

- `/[0-9]{1,3}(\.[0-9]{1,3}){3}/`
- `/(25[0-5]|2[0-4][0-9]|[01]?[0-9]{1,2})(\.(25[0-5]|2[0-4][0-9]|[01]?[0-9]{1,2})){3}/`

Further Reading

- https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html
  IEEE Std 1003.1, 2018 Edition. Chapter 9, Regular Expressions

- https://www.rand.org/content/dam/rand/pubs/research_memoranda/2008/RM704.pdf
  Representation of Events in Nerve Nets and Finite Automata

- https://regexone.com/ Learn Regular Expressions with simple, interactive exercises

- https://regexr.com/ Learn, Build, & Test RegEx

- https://regex101.com/ build, test, and debug regex

- https://www.debuggex.com/ Online visual regex tester

- http://www.regviz.org/ Visual Debugging of Regular Expressions

- https://regex-vis.com/ Regex Vis

- https://regexper.com/ Regexper

grep

Daniel KNITTL-FRANK
p23687@fh-hagenberg.at

May 9, 2024

grep

- Searches patterns (regular expressions) in text files or streams
- By default, prints all matching lines
- Exit code 0 (any line matched) or 1 (no lines matched)

Grep variants

- grep: Basic regular expressions (BRE)
  - Many meta characters have to be “activated”:
    - \? \+ \(\) \| \{\}
- grep -E, egrep: **E**xtended regular expressions (ERE)
  - Meta characters already activated:
    - ? + () | {}
- grep -F, fgrep: match **F**ixed strings
- grep -P: PCRE (perl-compatible regular expression)

Recap: quantifiers

- Quantifiers quantify preceding expression:
  - `*` 0…∞ repetitions
  - `?` 0…1 repetitions
  - `+` 1…∞ repetitions
  - `{n}` n repetitions
  - `{,n}`, `{0,n}` 0…n repetitions
  - `{m,n}` m…n repetitions
  - `{m,}` m…∞ repetitions

Grep options

- Synopsis: grep [option…] PATTERN [file…]
- Options
  - -c (count) Print number of matching lines (similar to grep | wc -l)
  - -h (hide) Suppresses output of filenames if multiple files match
  - -i (ignore/insensitive) Ignore case when matching
  - -l (list) Only print names of matching files
  - -n (number) Prepend line numbers
  - -o (only) Only print the matched part of the line (default: print full line)
  - -s (suppress) suppress error messages about non-existent or unreadable files
  - -v (invert) Only print lines without match
- Exhaustive list: man grep

POSIX Character classes

POSIX class	Equivalent to	Matches
[:alnum:]	[A-Za-z0-9]	digits, uppercase and lowercase letters, e.g. grep ‘[[:alnum:]]’ (**double** brackets!)
[:alpha:]	[A-Za-z]	upper- and lowercase letters
[:ascii:]	[\x00-\x7F]	ASCII characters
[:blank:]	[\t]	space and TAB characters only
[:cntrl:]	[\x00-\x1F\x7F]	Control characters
[:digit:]	[0-9]	digits
[:graph:]	[^ [:cntrl:]]	graphic characters (all characters which have graphic representation)
[:lower:]	[a-z]	lowercase letters
[:print:]	[[:graph:]]	graphic characters and space
[:punct:]	[-!"#$%&'()*+,./:;<=>?@[]^_`{|}~]	all punctuation characters (all graphic characters except letters and digits)
[:space:]	[\t\n\r\f\v]	all blank (whitespace) characters, including spaces, tabs, new lines, carriage returns, form feeds, and vertical tabs
[:upper:]	[A-Z]	uppercase letters
[:word:]	[A-Za-z0-9_]	word characters
[:xdigit:]	[0-9A-Fa-f]	hexadecimal digits

Non-standardized character groups

- Available with PCRE and other regex variants
- Might match non-latin unicode characters
- \d digit
  - \D everything except digit
- \w, \W word character (letters, digits, underscores), non-word character
- \s, \S whitespace character, non-whitespace character

Sed

Daniel KNITTL-FRANK
p23687@fh-hagenberg.at

May 9, 2024

Sed – Stream Editor

- Nicht-interaktiver Editor für Textdateien
- Führt Aktionen auf Zeilen aus
- Ausgabe auf STDOUT (Originaldatei wird standardmäßig *nicht* verändert)
  - -i für “in-place” editing
- Häufigste Aufgabe: Suchen und Ersetzen von Text (mittels regex)

Sed scripts

- sed ‘[addr]X[options]’ file
- addr is a line number, a regular expression, or a range of lines
- X is a single-letter sed command
- Additional options are used by some sed commands
- Example: sed ‘1,10s/SE/SEbb/g’
  - Replace (substitute) all occurrences of ‘SE’ with ‘SEbb’ in the first 10 lines

Simple sed commands

- d delete
- s/regex/replace/[flags] substitute
- q quit
- y transliterate
- { cmd; cmd; } group commands
- # comment

Substitute

- Common form: s/BRE/replace/
  - Also: s#BRE#replace#, s|BRE|replace|, s;BRE;replace;
- Flags:
  - g global: replace all occurrences in line
  - i ignore: ignore case when matching

Examples

- echo ‘Hallo Daniel’ | sed ‘s/[aeiou]/X/’
  - HXllo Daniel
- echo ‘Hallo Daniel’ | sed ‘s/[aeiou]/X/g’
  - HXllX DXnXXl
- echo ‘Hallo Daniel’ | sed ‘s/hallo/ciao/’
  - Hallo Daniel
- echo ‘Hallo Daniel’ | sed ‘s/hallo/ciao/i’
  - ciao Daniel

Examples

- sed ‘/^public/s/void/int/g’
  - Ersetzt void⇒int, aber nur in Zeilen, die mit “public” beginnen
- sed ‘1,3d’
  - Löscht die ersten 3 Zeilen
- sed ‘$d’
  - Löscht letzte Zeile
- sed ‘/XXX/d’
  - Löscht alle Zeilen mit XXX
- sed ‘/^$/d’
  - Löscht leere Zeilen

Greedy Regex

- Regular expressions are “greedy” by default
  - Match as much as possible
- Example:
  - echo ‘X<em>Emphasized text</em>Y’ | sed ‘s/<.*>//’ # ‘XY’
- Workaround:
  - echo ‘X<em>Emphasized text</em>Y’ | sed ‘s/<[^>]*>//’ # ‘XEmphasized textY’
- Some regex engines implement “lazy regex”: ‘.*?’

Shell Scripting Basics

Daniel KNITTL-FRANK
p23687@fh-hagenberg.at

May 9, 2024

Shell Functions

- Find and execute binaries
- IO redirect
- Expansions (Tilde, parameter, command substitution, arithmetic)
- Globbing (wildcards)
- Conditions, Loops
- Job control

Recap: Comments

- # makes everything a comment until the end of line

Recap: IO redirect

- 3 standard file descriptors: 0, 1, 2 (stdin, stdout, stderr)
- < redirects stdin
- > and 2> redirect stdout and stderr, respectively
- Special file /dev/null to discard any output

Simple text output

- echo outputs its arguments
  - Non-portable, different implementations
  - echo ‘Hello world’ # ‘Hello world’
- printf
  - printf ‘%s %d\n’ ‘--header--’ 42
  - Output: --header-- 42

Recap: Pipes

- Combine output and input of two processes
- Several pipes can be used in a single command line

Recap: Wildcards (Globs)

- Match files (and directories) within a single directory
    - ? a single arbitrary character
    - * 0 or more arbitrary characters
    - […] a single character from list
    - [^…] a single character not in list
- Exceptions:
    - . (hidden files)
    - / (directory separator)
- Globs are evaluated by the shell before a command is executed!
    - The number of parameters to a command is defined by the value after expansion
- If no paths match, a wildcard expands to itself

Shell Variables

- Weakly typed values (usually strings)
- Scoped to current process
- Create and assign variables by following their name immediately with an equal sign (no spaces allowed)
  - answer=42
  - name='Daniel'
- "export" to make visible in child processes
  - myvariable=xyz
    export myvariable
  - export newvariable=abc

Shell Variables (2)

- “set” shows all defined variables
- Variables can be expanded to their value with $, e.g. “$PATH” or “${PATH}”
- Non-existent or unset variables expand to the empty string

Parameter Expansion

- $ expands a parameter (variable)
- Modifiers:
  - `${var:-default value}`
  - `${var:+alternative value}`
  - `${var:=assign default value}`
  - `${var:?error if empty or unset}`
- Substring processing:
  - `${var%pattern}  # Remove smallest suffix`
  - `${var%%pattern} # Remove largest suffix`
  - `${var#pattern}  # Remove smallest prefix`
  - `${var##pattern} # Remove largest prefix`

Parameter Expansion: Examples

-
  ```
  value=42
  empty=
  echo “${value:-default} ${empty:-default} ${empty}” # 42 default
  echo “${value:+alt} ${empty:+alt} ${empty}” # alt
  echo “${value:=default} ${empty:=default} ${empty}” # 42 default default
  empty=
  echo “${empty:?my error message}” # bash: empty: my error message
  echo “$?” # 1
  ```

-
  ```
  filename=‘my-archive-20220314.tar.gz’
  echo “${filename%.*}” # my-archive-20220314.tar
  echo “${filename%%.*}” # my-archive-20220314
  echo “${filename#*-}” # archive-20220314.tar.gz
  echo “${filename##*-}” # 20220314.tar.gz
  ```

Quoting

- Variables are expanded inside “double quotes”, but not inside ‘single quotes’
- Example:
  - X=42
  - echo “$X” # outputs the number 42
  - echo ‘$X’ # outputs the string $X

Common Environment Variables

- PATH colon-separated list of directories with command binaries
- PS1 custom prompt string
  - Additionally: PS2, PS3, PS4
- USER current user
- LOGNAME logged-in user
- HOME home directory of current user
- LANG active language
- PWD current working directory

Example: Extending PATH

- PATH contains a list of directories, each separated with a colon “:”
- Shells use this variable to find executable binaries (first match wins)
- PATH is already “export”ed by default
- To add a custom directory:
  - PATH=“~/bin:$PATH”
- To make this change persistent, it can be added to one of the following files, which are executed when starting a shell:
  - ~/.profile (system-wide: /etc/profile)
  - ~/.bashrc (system-wide: /etc/bash.bashrc)

Arithmetic Expansion

- $((…)) evaluates an arithmetic expression
- Only integers are supported!
  - Use “bc” or “dc” if floating point arithmetic is required
- echo “$((1+2)) $((21*2)) $((13/2))”
  - 3 42 6

Shell Scripting

Daniel KNITTL-FRANK
p23687@fh-hagenberg.at

May 9, 2024

The Simplest Script

-
  ```
  #!/bin/sh
  echo ‘Hello World’
  ```

Structure

- First line is a special comment “#!”
  - Must be the first 2 bytes of the file
  - “Shebang” or “hashbang”
  - Defines the interpreter used to run the script, e.g.
    - #!/bin/sh
    - #!/bin/bash --posix
    - #!/usr/bin/env python
- Executable bit must be set (chmod +x)
- The rest can be any valid shell command(s)
- Semicolon ; at end of line is optional

Positional Parameters

- Shell scripts can process parameters (like any other executable)
- Parameters can be accessed via $1, $2, …, $9
- $0 contains path to executed script
  - ./script.sh # $0 = ./script.sh
  - /path/to/script.sh # $0 = /path/to/script.sh
  - sh script.sh # $0 = script.sh

Special Parameters

- $* expands to all positional parameters. Quoted, it expands to a single word
- $@ expands to all positional parameters. Quoted, it expands to each parameter being a separate word
- $# expands to the number of positional parameters
- $? expands to the exit status of the previous command
- $! expands to PID of most recent background command

Command Substitution

- $(…) evaluates the inner command(s) in a subshell and substitutes its output
- Examples:
  - echo “You are here: ‘$(pwd)’.”
    # Output: You are here: ‘/home/user’.
  - echo “Type of ~: $(ls -ld ~ | cut -c1)”
    Output: Type of ~: d

Control Flow

- Conditional execution: &&, ||
- Conditions: if, case
- Loops: for, while, until
  - break, continue
- Functions

Conditional execution

- cmd1 && cmd2: second command is only executed if first command was successful
  - test -f file && rm file # deletes file only if it exists
- cmd1 || cmd2: second command is only executed if first command failed
  - test -s file || rm file # deletes file only if it is empty

Conditions: if

- Executes a (list of) command(s) and depending on the exit status, executes the “then” or “else” branch
- “[“ (or “test”) are the most commonly used commands
- “elif” and “else” are optional

-
  ```
  if cond-list; then
    true-list
  elif cond-list; then
    true-list
  else
    false-list
  fi
  ```

“test” and “[“

- “[“ (or “test”) evaluate expressions and exit with the appropriate status code (=0 if expression was true, !=0 if expression was false)
- “[expr]” is equivalent to “test expr”
- Combine multiple expressions with conditional execution:
  - [“$var” -gt 0] && [“$var” -ne 42]
- Invert status code by prepending an exclamation mark
  - ‘! [“$x” -eq 1]’ is equivalent to ‘[“$x” -ne 1]’
- Expressions can compare values or query the file system

“[“ expressions

- “[expr]” is equivalent to “test expr”
- String comparisons:
  - [string = string]
  - [string1 != string2]
  - [-z string]
  - [-n string]
    - Equivalent to: [string]
- Integer comparisons (floats are not supported):
  - [0 -eq 0] # equal
  - [0 -ne 1] # not equal
  - [0 -lt 1] # less than
  - [1 -gt 0] # greater than
  - [0 -le 1] # less than or equal
  - [1 -ge 0] # greater than or equal

“[“ expressions

- “[expr]” is equivalent to “test expr”
- File comparisons:
  - [file1 -ef file2] # same file
  - [file1 -nt file2] # newer than
  - [file2 -ot file1] # older than
- File checks:
  - test -d dir # is directory?
  - test -e path # exists?
  - test -f file # is (regular) file?
  - test -L link # is symbolic link?
  - test -r file # is readable?
  - test -w file # is writable?
  - test -x file # is executable?
  - And many more … “man [“ (or “man test”) is your friend

Conditions: case

- Compares a value and executes the commands of the first matching (wildcard) pattern
- Patterns are terminated with “)”
- Each case must end with “;;”
-
  ```
  case value in
    pattern1) list ;;
    pattern2|pattern3) list ;;
    ?attern4) list ;;
    pattern?) list ;;
    *) default-list ;;
  esac
  ```

Loops: while

- Executes a list of commands repeatedly, while a condition is true (i.e. exit status = 0)
- Condition can be any command

-
  ```
  while cond-list; do
    body-list
  done
  ```

Loops: until

- Executes a list of commands repeatedly, until a condition becomes true (i.e. exit status = 0)
- Condition can be any command

-
  ```
  until cond-list; do
    body-list
  done
  ```

Loops: for

- Executes a list of commands for each word in turn
- Words are separated by $IFS (defaults to whitespace)

-
  ```
  for x in words; do
    body-list
  done
  ```

Control Flow: Examples

-
  ```
  #!/bin/sh
  if [ $# -lt 1 ]; then
    echo “ERROR: Usage: $0 FILE...” >&2
    exit
  fi
  for file in “$@”; do
    test -e “$file” || { echo “$file does not exist”; continue; }
    case “$file” in
      *.txt) echo “$file has txt extension” ;;
      *.sh) echo “$file has sh extension” ;;
      *.png|*.jpg|*.gif) echo “$file has an image extension” ;;
      *) echo “unknown file extension: ${file#*.}” ;;
    esac
  done
  ```

Functions

- f() { …; } defines a shell function
- Call function by its name, “f”
- Parameters are simply written after the function name
  - f param1 param2 param3
- Parameter values in function accessed via $1, $2, …

Functions (Example)

-
  ```
  # definition:
  fun() {
    echo “First param: $1”;
    echo “Second param: $2”;
    echo “All params: $@”;
  }
  ```

-
  ```
  # call:
  fun with ‘GNU and’ Linux
  ```

-
  ```
  # output:
  First param: with
  Second param: GNU and
  All params: with GNU and Linux
  ```

Read

- read [-p prompt] variable...
- Reads a line from standard input and assigns variables (in turn, from left to right)
- If input contains fewer fields than variables: variables are empty
- If input contains more fields than variables: last variable contains all remaining fields
- -p specifies a prompt which is shown to the user
  - read -p “What is your name? ” username

Examples: Read

-
  ```
  date
  # Fr 1 Jul 2022 13:37:42 CEST
  ```

-
  ```
  date | while read weekday day month year time zone; do
    echo “It is $weekday in $month at $time”
    # It is Fr in Jul at 13:37:42
  done
  ```

-
  ```
  date | while read weekday ignore; do
    echo “Today is $weekday”
    # Today is Fr
  done
  ```

-
  ```
  date | while read weekday day month year time zone too many fields; do
    echo “$too $many $fields”
    # output is 2 space characters: “  ”
  done
  ```

Aliases

- Allow creating shorthands for commands
- Example:

  -
    ```
    alias ll='ls -l'
    ll path/to/dir
    ```