Домашнее задание № 7. Оболочка и скрипты

6 января 2024 г.

Семён Чайкин, ИУ9-11Б

Задание 1

Реализуйте собственный вариант утилиты `tree`. Пусть ваша программа поддерживает по меньшей мере ключи `-d` и `-o` так же, как реализация утилиты tree в ОС Linux. Поддержку других ключей можно не реализовывать.

Для «рисования» дерева в консоли используйте символы псевдографики.

Программа не должна аварийно завершаться, если права доступа запрещают получение списка файлов какого-либо каталога.

Реализация

```ruby
#!/usr/bin/env ruby

require 'optparse'

def recursive_tree(wd, root, only_directory, outstream)
  dirs, files = [], []
  path = wd.gsub root, ''
  level = path.count '/'
  ident = (level > 0 ? "  |" * (level-1) : "") + "  +- "
  subident = (level > 0 ? "  |" : "") + ident

  begin
    Dir.open(wd).children().each() do |path|
      path = "#{wd}/#{path}"
      dirs << path if File.directory?(path)
      files << path if File.file?(path)
    end
  rescue
    return outstream.puts "#{ident}#{wd.split('/')[-1]} [error opening dir]"
  end

  outstream.puts "#{ident}#{path.split('/')[-1]}"
```

```
  dirs.each{|dir| recursive_tree dir, root, only_directory, outstream}
  only_directory || \
      files.each{|file| outstream.puts "#{subident}#{file.split('/')[-1]}"}
end

def tree(wd, only_directory, outstream)
  recursive_tree("#{wd}", "#{wd}", only_directory, outstream)
end

def open_file(filename)
  raise "Can't open file" if File.file?(filename) and !File.writable?(filename)
  File.open(filename, "w") rescue raise "Don't have permissions to open file"
end

options = {}
OptionParser.new do |opt|
  opt.banner = "Usage: #{$0} [options] [path]"
  opt.on('-d', 'list directories only') \
      { |o| options[:only_directory] = true}
  opt.on('-o FILENAME', 'output to file instead of stdout') \
      { |o| options[:output_file] = o}
end.parse!

stream = options.has_key?(:output_file) ?
         open_file(options[:output_file]) : STDOUT

ARGV.empty? ? tree(Dir.getwd, options.has_key?(:only_directory), stream) :
              ARGV.each { |wd| tree wd, options.has_key?(:only_directory), stream }
```

Тестирование

Результат в консоли:

```
[trololo@trololo 1]$ ls
tree.rb
[trololo@trololo 1]$ chmod +x tree.rb
[trololo@trololo 1]$ mkdir -p {a,b,c}/{d,e,f}
[trololo@trololo 1]$ touch {a,b,c}/{d,e,f}/t{1,2,3}.txt
[trololo@trololo 1]$ ./tree.rb --help
Usage: ./tree.rb [options] [path]
    -d                               list directories only
    -o FILENAME                     output to file instead of stdout
[trololo@trololo 1]$ ./tree.rb
  +-
  +- a
```

```
  |  +- d
  |  |  +- t1.txt
  |  |  +- t2.txt
  |  |  +- t3.txt
  |  +- e
  |  |  +- t1.txt
  |  |  +- t2.txt
  |  |  +- t3.txt
  |  +- f
  |  |  +- t1.txt
  |  |  +- t2.txt
  |  |  +- t3.txt
  +- b
  |  +- d
  |  |  +- t1.txt
  |  |  +- t2.txt
  |  |  +- t3.txt
  |  +- e
  |  |  +- t1.txt
  |  |  +- t2.txt
  |  |  +- t3.txt
  |  +- f
  |  |  +- t1.txt
  |  |  +- t2.txt
  |  |  +- t3.txt
  +- c
  |  +- d
  |  |  +- t1.txt
  |  |  +- t2.txt
  |  |  +- t3.txt
  |  +- e
  |  |  +- t1.txt
  |  |  +- t2.txt
  |  |  +- t3.txt
  |  +- f
  |  |  +- t1.txt
  |  |  +- t2.txt
  |  |  +- t3.txt
  +- tree.rb
[trololo@trololo 1]$ ./tree.rb not found
  +- not [error opening dir]
  +- found [error opening dir]
```

Задание 2

Реализуйте собственный вариант утилиты `grep`. Допускается ограничиться работой только с текстовыми файлами. Так же, как и стандартная утилита `grep`, ваша программа должна обрабатывать как стандартный ввод, так и файлы, пути к которым указаны в командной строке. Ключ `-e` должен позволять передать программе регулярное выражение вместо строки для поиска. Пусть ваша реализация также поддерживает ключи `-i`, `-m`, `-n` так же, как это делает стандартная реализация утилиты `grep`. Поддержку других ключей можно не реализовывать.

Программа не должна аварийно завершаться, если какой-либо из файлов, перечисленных в аргументах командной строки, не может быть прочитан.

Сообщения об ошибках и предупреждения должны направляться в стандартный поток вывода ошибок.

Направление таких сообщений в стандартный поток вывода не допускается.

Реализация

```ruby
#!/usr/bin/env ruby

require 'optparse'

def valid?(line, pattern)
  pattern.is_a?(String) ?
      (line.include? pattern) :
      (line =~ pattern)
end

def integer?(v)
  Integer(v) rescue false
end

def open_file(filename, mode)
  if mode == 'r'
    raise "Can't open file" if !File.file?(filename) or \
                               !File.readable?(filename)
  elsif mode == 'w'
    raise "Can't open file" if  File.file?(filename) and \
                               !File.writable?(filename)
  end
  File.open(filename, mode) \
      rescue raise "Don't have permissions to open file"
end
```

```
def grep_from_stream(stream, pattern, re, ignore_case,
                     max_count, show_lines)
  return if max_count == 0
  pattern = pattern.downcase if ignore_case

  begin
    pattern = Regexp.new pattern if re
  rescue Exception => e
    raise "Invalid regular expression"
  end

  counter = 0
  begin
    stream.each_with_index do |line, index|
      temp = ignore_case ? line.downcase : line
      if valid?(temp, pattern)
        counter+=1
        if show_lines
          puts "#{(index+1)} #{line}"
        else
          puts "#{line}"
        end
        break if counter == max_count
      end
    end
    rescue SignalException => e
  end
end

def grep(files, options)
  re = options.has_key?(:re)
  ignore_case = options.has_key?(:ignore_case)
  max_count = options.fetch(:max_count, -1)
  raise "Invalid max count (expected integer, " \
            "got #{max_count})" if !integer?(max_count)
  max_count = max_count.to_i
  pattern = options[:pattern]
  show_lines = options.has_key?(:show_lines)

  files.each do |file|
    if !File.file?(file)
      puts "Can't open file: #{file}"
      next
    end

    puts "#{file}\n#{'=' * file.length}"
```

```
    grep_from_stream File.open(file), pattern, re, \
                     ignore_case, max_count, show_lines
  end

  grep_from_stream STDIN, pattern, re, \
                   ignore_case, max_count, show_lines if files.empty?
end

ARGV << "-h" if ARGV.empty?

options = {}
OptionParser.new do |opt|
  opt.banner = "Usage: #{$0} [option]... [file]..."
  opt.on('-e PATTERN', 'use PATTERNS for matching') \
      { |p| (options[:pattern] = p); options[:re] = true }
  opt.on('-i', 'ignore case distinctions in patterns and data') \
      { |o| options[:ignore_case] = true }
  opt.on('-m NUM', 'stop after NUM selected lines') \
      { |o| options[:max_count] = o }
  opt.on('-n', 'print line number with output lines') \
      { |o| options[:show_lines] = true }
  opt.on_tail('-h', 'show help message') \
      { |o| puts opt; exit }
end.parse!

args = ARGV
if !options.has_key?(:pattern)
  options[:pattern] = args[0]
  args = args[1..]
end

grep args, options
```

Тестирование

```
[trololo@trololo 2]$ ls
grep.rb
[trololo@trololo 2]$ chmod +x grep.rb
[trololo@trololo 2]$ ./grep.rb
Usage: ./grep.rb [option]... [file]...
    -e PATTERN                       use PATTERNS for matching
   -i                    ignore case distinctions in patterns and data
    -m NUM                           stop after NUM selected lines
   -n                              print line number with output lines
    -h                               show help message
[trololo@trololo 2]$ cat >> test.txt << EOF
```

```
> Сложный был год:
> налоги, катастрофы, проституция, бандитизм и недобор в армию.
> С последним мириться было нельзя, и за дело принялся знающий человек —
наш военком.
> Он собрал всех тунеядцев, дураков и калек в районе,
> даже глухих определил в погранотряд «Альпийские тетерева».
> Столько лет уже прошло, а они ещё где-то чудят!
> EOF
[trololo@trololo 2]$ ./grep.rb "военком" test.txt
test.txt
========
С последним мириться было нельзя, и за дело принялся знающий человек —
наш военком.
[trololo@trololo 2]$ ./grep.rb "A" test.txt -m 2 -i -n
test.txt
========
8 налоги, катастрофы, проституция, бандитизм и недобор в армию.
9 С последним мириться было нельзя, и за дело принялся знающий человек —
наш военком.
[trololo@trololo 2]$ ./grep.rb -e "[[:digit:]]" -i
Армия не просто доброе слово, а очень быстрое дело.
Так мы выигрывали вс3 в0йны.
2 Так мы выигрывали вс3 в0йны.
Пока противник рисует карты наступл3ния, мы меняем ландшафты, причём вручную.
3. Пока противник рисует карты наступл3ния, мы меняем ландшафты, причём вручную.
Когда приходит время атаки, противник теряется на незнакомой местности
и приходит в полную небоеготовность.
В эт0м смысл, в эт0м наша страт3гия.
5 В эт0м смысл, в эт0м наша страт3гия.
```

Задание 3

Реализуйте собственный вариант утилиты `wc`. Так же, как и стандартная утилита `wc`, ваша программа должна обрабатывать как стандартный ввод, так и файлы, пути к которым указаны в командной строке. Пусть ваша реализация поддерживает ключи `-c`, `-m`, `-w`, `-l` так же, как это делает стандартная реализация утилиты `wc`. Поддержку других ключей можно не реализовывать.

Сообщения об ошибках и предупреждения должны направляться в стандартный поток вывода ошибок.

Направление таких сообщений в стандартный поток вывода не допускается.

Реализация

```ruby
#!/usr/bin/env ruby

require 'optparse'

def wc_from_stream(stream)
  lines = words = symbols = bytes = 0

  begin
    stream.each do |line|
      symbols += line.length
      bytes += line.bytesize
      lines += 1 if line[-1] == "\n"
      words += line.split(' ').size
      puts "\n" if line[-1] != "\n" and stream == STDIN
    end
    rescue SignalException => e
  end
  [ bytes, symbols, lines, words ]
end

def wc_wrapper(stream, options)
  count_lines = (options.has_key?(:count_lines) or options.empty?)
  count_words = (options.has_key?(:count_words) or options.empty?)
  count_symbols = options.has_key?(:count_symbols)
  count_bytes = (options.has_key?(:count_bytes) or options.empty?)
  bytes, symbols, lines, words = wc_from_stream stream
  result = []
  result << lines if count_lines
  result << words if count_words
  result << symbols if count_symbols
  result << bytes if count_bytes
  result
end

def wc(files, options)
  if files.empty?
    puts wc_wrapper(STDIN, options)
    return
  end

  total = []
  files.each do |file|
    (STDERR.puts "#{file}: Can't open file"; next) if !File.file?(file) or \
                                                      !File.readable?(file)
```

```
    result = wc_wrapper(File.open(file), options)
    total = (total.empty? ? result : [total, result].transpose.map(&:sum))
    puts "#{result.join(' ')} #{file}"
  end
  puts "#{total.join(' ')} total" if files.size > 1
end

options = {}
OptionParser.new do |opt|
  opt.banner = "Usage: #{$0} [option]... [file]..."
  opt.on('-c', 'print the byte counts') \
      { |o| options[:count_bytes] = true}
  opt.on('-m', 'print the character counts') \
      { |o| options[:count_symbols] = true}
  opt.on('-l', 'print the newline counts') \
      { |o| options[:count_lines] = true}
  opt.on('-w', 'print the word counts') \
      { |o| options[:count_words] = true}
end.parse!

wc ARGV, options
```

Тестирование

- Текстовый файл `3-1.txt`:

```
Джордж, ты ковбой. Ты остановись, б..., ты кончай, ты патроны
спрячь подальше на склад и забудь про своего папу. У нас был
один мудак, отомстил за брата, б..., и рухнула Великая Российская
империя. И другой чудак был, за своего дедушку отомстил, и рухнул
Советский Союз. И ты повторишь ту же ошибку. Ты папу забудь, папа
отработал свое. Ты подумай о будущем Америки. Она гибнет.
Твоя молодежь бежит из твоей страны. Там никто не хочет жить
в Америке, никто! Эта барахолка, б...!
Доллар, доллар, доллар, ... Эта грязная зеленая бумажка!
```

- Текстовый файл `3-2.txt`:

```
- Что, товарищ абитуриент, ссымся?!
- Так точно, ссусь!
- Ну это не беда! Такая сегодня экологическая обстановка.
Все ссутся… Я ссусь… И даже замдек пысается, бывает, но по ситуации!
Что ж нам из-за этого, в престижный вуз не поступать?
Твой позорный недуг мы в подвиг определим:
пошлём на кафедру высшей математики. Там ты ещё и сраться начнёшь.
```

Результат в консоли:

```
[trololo@trololo 3]$ ls
3-1.txt 3-2.txt wc.rb
[trololo@trololo 3]$ chmod +x wc.rb
[trololo@trololo 3]$ ./wc.rb --help
Usage: ./wc.rb [option]... [file]...
    -c                               print the byte counts
    -m                               print the character counts
    -l                               print the newline counts
    -w                               print the word counts
[trololo@trololo 3]$ ./wc.rb 3-1.txt 3-2.txt
9 89 945 3-1.txt
7 58 618 3-2.txt
16 147 1563 total
[trololo@trololo 3]$ ./wc.rb 3-1.txt 3-2.txt -w
89 3-1.txt
58 3-2.txt
147 total
[trololo@trololo 3]$ ./wc.rb -wlcm
У нас убеждения.
Какие такие убеждения?
Мы веруем в Господа нашего, Говинду, а он нам в людей стрелять не велит.
Всё, вы нам подходите. И говинда ваша ничего. Жутковато, но ничего.
И стричь вас опять же не надо. И люди вы, видно, выносливые,
4 часа «Харе Кришну» орать— это не каждый сдюжит... Пойдёте в химвойска.^D
5
55
314
556
```

Задание 4

Реализуйте простейшую программу проверки орфографии. Пусть программа принимает на вход словарь и текст на естественном языке и выводит список и координаты слов (строка, колонка), которые не встречаются в словаре.

Считайте, что в проверяемом тексте переносы слов отсутствуют. Различные формы одного слова рассматривайте как разные слова. Апостроф считайте частью слова.

Реализация

```ruby
#!/usr/bin/env -S ruby -W0

require 'continuation'
require 'set'
```

```
def whitespace?(ch)
  ch =~ /[[:space:]]/
end

def punctuation?(ch)
  (ch =~ /[[:punct:]]/) and ch != "`"
end

def symbol?(ch)
  (ch =~ /[[:alnum:]]/) or ch == "`"
end

def number?(ch)
  ch =~ /^[[:digit:]]$/
end

def make_stream(s)
  temp_stream = (s+"\0").downcase.chars
  last_newline = l = 0
  stream = []

  temp_stream.each_with_index do |x, i|
    stream << [i+1, l, i-last_newline+1, x]
    if x == "\n"
      last_newline = i+1
      l += 1
    end
  end

  stream
end

def scanner(s)
  stream = make_stream s

  callcc do |error|
    result = tokens(stream, error)
    (peek(stream)[-1] == "\0") and result
  end
end

def peek(stream)
  raise "Invalid stream" if stream.empty?
  stream[0]
end
```

```
def next_(stream)
  ch = peek stream
  stream.replace stream[1..]
  ch
end

def tokens(stream, error)
  pos, l, c, ch = peek stream
  result = []

  if whitespace? ch
    spaces stream, error
    result = tokens stream, error
  elsif (punctuation? ch) or (symbol? ch)
    result << token(stream, error)
    result += tokens(stream, error)
  end
  result
end

def token(stream, error)
  pos, l, c, ch = peek stream
  if punctuation? ch
    next_ stream
  elsif symbol? ch
    [pos, l, c, word(stream, error)]
  else
    error false
  end
end

def word(stream, error)
  pos, l, c, ch = peek stream
  if symbol? ch
    pos, l, c, ch = next_(stream); ch + word(stream, error)
  else
    ''
  end
end

def spaces(stream, error)
  next_ stream if whitespace? peek(stream)[-1]
end

def open_file(filename, mode)
```

```
  if mode == 'r'
    raise "Can't open file" if !File.file?(filename) or \
                               !File.readable?(filename)
  elsif mode == 'w'
    raise "Can't open file" if  File.file?(filename) and \
                               !File.writable?(filename)
  end
  File.open(filename, mode) \
      rescue raise "Don't have permissions to open file"
end

def create_dictionary(text)
  tokens = scanner(text)
  raise "Invalid dictionary content" if !tokens
  result = Set.new
  tokens.each {|pos, l, c, value| result << value \
      if symbol?(value) and not number?(value)}
  result
end

def load_dictionary(filename)
  s = open_file(filename, "r").read
  create_dictionary s
end

def save_dictionary(filename, dictionary)
  f = open_file(filename, "w")
  dictionary.each do |item|
    f.write("#{item}\n")
  end
end

def find_missspelled(filename, dictionary)
  s = open_file(filename, "r").read
  tokens = scanner(s)
  raise "Invalid content" if !tokens

  tokens.each do |pos, l, c, value|
    if symbol?(value) and not number?(value)
      if !dictionary.include?(value)
        puts "#{l},\t#{c}\t#{value}"
      end
    end
  end
end
```

```ruby
if ARGV.size < 1 or ARGV.size > 2
  puts "Usage: #{$0} <dictionary file> <test file>"
  exit
end

d = load_dictionary(ARGV[0])

if ARGV.size > 1
    find_missspelled(ARGV[1], d)
else
    save_dictionary("dict-#{ARGV[0]}", d) rescue Exception => e
end
```

Тестирование

Текстовый файл `4.txt`:

```
Quotes are for dumb people who can't think of something
intelligent to say on their own.
```

Текстовый файл `4-ms.txt`:

```
Quotes aer for dumb peopel who cant think of sumething
intelligent to say on ther own.
```

```
[trololo@trololo 4]$ ls
4-ms.txt 4.txt speller.rb
[trololo@trololo 4]$ chmod +x speller.rb
[trololo@trololo 4]$ ./speller.rb
Usage: ./speller.rb <dictionary file> [test file]
[trololo@trololo 4]$ ./speller.rb 4.txt
[trololo@trololo 4]$ ls
4-ms.txt 4.txt dict-4.txt speller.rb
[trololo@trololo 4]$ ./speller.rb dict-4.txt 4-ms.txt
0,  8   aer
0,  21  peopel
0,  32  cant
0,  46  sumething
1,  23  ther
```

Вывод

В ходе выполнения этой домашней работы я понял, как реализовываются некоторые стандартные скрипты, а также как разрабатывать собственные вспомогательные программы.

В процессе выполнения этого домашнего задания я столкнулся с серьезной

“трудностью”: надо было за короткий срок времени разобраться с новым, неизвестным для меня языком программирования Ruby.

Я думаю, что знания, приобретённые во время выполнения этого домашнего задания, будут полезными и для дальнейшего обучения и развития в области программирования.