Writing Tools with Ruby

William Mitchell (whm)
Mitchell Software Engineering (.com)

About the speaker...

- William Mitchell

- Consultant/contractor specializing in software development and training of software developers.   Practice includes Java, C++, C, C#, Icon, Ruby, object-oriented methods, and test-driven development.

- Occasionally teach courses about programming languages at The University of Arizona (CSc 328, 352, 372, 451).

- BSCS, North Carolina State University, 1981
  MSCS, The University of Arizona, 1984

Writing Tools with Ruby

This one-hour talk has two goals:

- Encourage creation of tools to automate tasks.
- Teach enough Ruby to make Ruby a viable choice for writing some tools.
  — We'll see only a tiny amount of the language.

In many ways this talk is not about Ruby but rather why you should know Ruby or a language like it.

Automation is the name of the game

- Every working person on the planet regularly encounters tasks that can be automated.

- Knowledge workers regularly encounter time-consuming, tedious, and/or error-prone tasks that can be automated by software.

- Programmers are a unique class of knowledge worker: they encounter tasks that can be automated <u>and</u> have the skill to create tools to automate those tasks.

- Ironically, programmers often don't automate those tasks and do them manually again and again, just like the billions of poor slobs that don't know how to program!

What's wrong?

Why do programmers who work all day to automate a task for an employer not take time to automate tasks they encounter in the workplace?

- Lazy?
- Don't really like to program?
- Paid by the hour?
- Don't have the time to automate?

When to automate?

Imagine a document that contains a bunch of numbers intermixed with other text:

```
x=(10, 20) [40, 12,  1231]
up 1000, down 1200, right 50, left 90
x9920yzzs23123k335S23423lllds3
...
```

The task is to find the longest run of digits in the text.

- How long would the document need to be before you'd create a tool to do it?
- What if this were a daily activity? What if a mistake could cost you your job?
- Would C be a good language to automate this task? Java? How about a shell pipeline?

A number of tasks can be profitably automated using languages like Perl, Python, Icon, and Ruby. (These languages are often called "scripting languages" but it's hard to precisely define what constitutes a "scripting language".)

What is Ruby?

- "A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write." — ruby-lang.org

- Ruby is commonly described as an "object-oriented scripting language".

- Ruby was invented in 1993 by Yukihiro Matsumoto ("Matz"), a "Japanese amateur language designer" (his words).

- A book, *Programming Ruby* by Thomas, et al. (2000), and Ruby on Rails (2004), a web application framework that provides for astonishing productivity, have caused interest in Ruby to soar in recent years.

Hello, Ruby

Here is hello.rb:

```
puts("Hello, world!")
```

Execution:

```
% ruby hello.rb
Hello, world!
```

Note:

No need for an enclosing class and/or method.

This implementation of Ruby doesn't have the notion of an executable file other than the source file itself.

Line numbering

Problem:

We want to mail a line-numbered listing of source code to a colleague, using the line numbers to conveniently cite code of interest.

Let's write a program that will read lines on "standard input" and write those lines, with numbers, on "standard output".

Example:

```
% ruby numlines.rb < writebytes.c
1: #include <stdio.h>
2: main()
3: {
4:     int byte_value;
5:
6:     while (scanf("%d", &byte_value) == 1)
7:         putchar(byte_value);
8: }
```

We might put the output on the clipboard to then paste into a message:

```
% ruby numlines.rb < writebytes.c >/dev/clipboard
```

numlines.rb

```
line_number = 1

while line = gets()
    print(line_number, ": ", line)
    line_number += 1
end
```

Things to note:

Ruby has no notion of variable declarations.

The gets method reads the next line from standard input and returns an instance of the String class.

When end of file is reached on standard input, the while loop terminates.

Line numbering, continued

Two minor improvements:

Ruby doesn't require parameter lists to be enclosed in parentheses.

We can use the "shebang" mechanism to make this script directly executable, so we don't need to type "ruby ..." to run it.

Here is the new version, named numlines:

```
#!/usr/bin/ruby
line_number = 1
while line = gets
    print line_number, ": ", line
    line_number += 1
end
```

Usage:

```
% numlines < writebytes.c
1: #include <stdio.h>
...
```

The longest line

Situation:

You awake in a cold sweat, unable to remember the longest word in /usr/share/dict/words on lectura.

Response:

You write a Ruby program.

```
longest = nil

while line = gets
    if longest == nil || line.size > longest.size
        longest = line
    end
end

puts longest
```

Notes:

Most Java operators have counterparts in Ruby represented by the same symbol(s).

The String.size method returns the number of characters in a string.

Line order reversal

Here is one of my favorite interview questions:

> "In any language you want, write a program that reads lines from standard input and writes them to standard output in reverse order. Last line first, first line last."

Example:

```
% ruby revlines.rb < writebytes.c
}
        putchar(byte_value);
    while (scanf("%d", &byte_value) == 1)

    int byte_value;
{
main()
#include <stdio.h>
```

revlines.rb

Here is a solution that relies on string concatenation:

```
result = ""

while line = gets
    result = line + result
end

puts result
```

revlines2.rb

Here is a solution that uses Ruby's Array class, which is similar to ArrayList and Vector in Java.

```
lines = [ ]                    Equivalent: lines = Array.new

while line = gets
   lines.push line
end

for line in lines.reverse
   puts line
end
```

Notes:

The subject of a for's in clause must be an object that has an "each" method.

revlines3.rb

If a prospective hire claimed to know Ruby, I'd want to see an answer like this:

```
puts readlines.reverse          # One-liner!
```

Notes:

readlines reads standard input to end of file and returns an array of the lines.

If given an array, puts prints each element on a separate line.

Time totaling

Imagine:

An instructor offers one point of extra credit for reporting how much time you spend on each assignment. For the first assignment you've tracked your time in a plain text file, like this:

```
% cat atimes.txt
Saturday morning    2:20
Monday              3:45
Tuesday am, in lab  1:35
Tuesday night--finished! 1:48
```

Now what? Write a program or total them by hand?

timetotal.rb

```
minutes = 0

while line = gets
    line.chomp!

    parts = line.split " "

    parts = parts[-1].split ":"

    if parts.length == 2
        minutes += parts[0].to_i * 60 + parts[1].to_i
    else puts "?" end
end

printf("%d:%02d\n", minutes / 60, minutes % 60)
```

split breaks a string into pieces based on delimiters:

```
>> parts = "Saturday morning    2:20".split " "
=> ["Saturday", "morning", "2:20"]

>> parts[-1]                 => "2:20"

>> parts[-1].split ":"       => ["2", "20"]
```

Usage:

```
% timetotal.rb < atimes.txt
9:28
```

timetotal2.rb

Problem: Extend the program so that it handles times like "4h" and "30m", too.

For reference:

```
minutes = 0

while line = gets

    line.chomp!

    parts = line.split " "

    parts = parts[-1].split ":"

    if parts.length == 2
        minutes += parts[0].to_i * 60 + parts[1].to_i
    else puts "?" end
end

printf("%d:%02d\n", minutes / 60, minutes % 60)
```

Number extraction

```
x=(10, 20) [40, 12,  1231]
up 1000, down 1200, right 50, left 90
x9920yzzs23123k335S23423lllIds3
...
```

A solution for the number extraction problem:

```
numbers = [ ]

while line = gets
   numbers += line.scan /[0-9]+/                         (1)
end

numbers = numbers.sort {|b,a| a.size <=> b.size }        (2)

puts numbers.select {|n| n.size == numbers[0].size}      (3)
```

Notes:

Line (1) uses a regular expression to extract strings that consist only of digits. The strings from each line are added to the array `numbers`.

Line (2) sorts the array based on the size of each string, descending.

Line (3) prints the array elements that have the same size as the first, thus handling ties.

It might be better to skip (3) and let the user pipe the output into `more`.

My first Ruby tool

September 3, 2006:

```
n = 1
d = Date.new(2006, 8, 22)
incs = [2,5]
pos = 0
while d < Date.new(2006, 12, 6)              (1)
    if d != Date.new(2006, 11, 23)           (2)
        printf("%s %s, #%2d\n",
            if d.cwday() == 2: "T"; else "H";end, d.strftime("%m/%d/%y"), n)
        n += 1
    end
    d += incs[pos % 2]                       (3)
    pos += 1
end
```

Note the use of operator overloading at (1) and (2) to compare two dates, and at (3) to add days to a date.

Output:

```
T 08/22/06, # 1
H 08/24/06, # 2
T 08/29/06, # 3
...
```

About the story...

Once upon a time a team of ten little programmers was working on a gigantic Java application. Every day the bug tracking system mailed each of the programmers a long list of his or her "open tickets": bugs that needed to be fixed and features that needed to be added. Sadly, the list of tickets was not ordered by priority and each of the programmers spent about five minutes every day looking through the list to find the high priority tickets.

One day one of the programmers remembered that computers can be used to automate things! In a few minutes she wrote a tiny Ruby program that read the daily ticket mailing and put the tickets in priority order. The program saved her about five minutes every day. She shared the tiny program with the other programmers, putting her company on track to save about a man-month per year—$10,000 based on typical burden rates for software developers. Her boss was so happy that he bought her lunch the very next day!

Speculate: Fact or fiction?

Ticket sorter

Here's what that ticket mailing looked like:

```
Date: Fri, 21 Apr 2006 12:12:09 GMT
Subject: Your open tickets
To: whm@MitchellSoftwareEngineering.com
X-Scanned-By: MIMEDefang 2.54 on 192.81.123.24

William Mitchell,

You have 26 open tickets:


6442: MFG tax data incomplete
        Age     : 4 months ago
        URL     : http://172.23.10.26/Ticket/Display.html?id=6442
        Priority: 1


7134: User walk-away from 2.1.1 leaves scraper unable to fetch same customer
        Age     : 6 weeks ago
        URL     : http://172.23.10.26/Ticket/Display.html?id=7134
        Priority: 1

...24 more...
```

Ticket sorter, continued

```
lines = readlines[15..-1]                (1)

tickets = [ ]

while lines[0] =~ /:/                    (2)
   ticket = lines[0,4] * ""              (3)
   priority = lines[3].split(":")[1].to_i
   tickets.push [priority,ticket]        (4)
   lines = lines[6..-1]
end

tickets.sort {|a,b| a[0] <=> b[0]}.each{ |t| print t[1],"\n"}
```

Notes:

(1) `15..-1` is a `Range`. This assignment discards the first 15 lines of text.

(2) `=~` is a regular expression match. We assume if the line has a colon, it's another ticket.

(3) The expression *Array * String* concatenates the array elements, using the string as a separator.

(4) `tickets` is an array of two-element arrays: `[[p1, t1], [p2, t2], ...]`

(5) Sort the tickets by priority and print the tickets.

Learn more about Ruby

- What we didn't cover about Ruby is just about everything, but perhaps this provides you a foothold and/or motivation to learn more.

- The Ruby home page is ruby-lang.org.

- *Programming Ruby—The Pragmatic Programmers' Guide, 2nd edition,* by Thomas, Fowler, and Hunt, also known as the "pickaxe book", is widely recognized as being the best book on Ruby at present. The first edition of the pickaxe book is available for free: www.ruby-doc.org/docs/ProgrammingRuby

- *Ruby Cookbook*, by Lucas Carlson and Leonard Richardson, is packed full of small but practical examples of using Ruby in a wide variety of settings.

- *Agile Web Development with Rails, 2nd edition*, by Dave Thomas et al. is commonly recommended if you're interested in learning about Ruby on Rails. It assumes knowledge of Ruby.

- The ruby-lang channel on irc.freenode.net is pretty good for live Q&A.

- My slides for the Ruby segment of CSc 372, Fall 2006 are still on the web.