Expansion Pak No. 1.

The Tiger's Vest (with a Basic Introduction to Irb)

there once was a tiger who wore a vest.
but that wasn't his biggest problem.
Because the earth was going to crash into the sun!!!
But he didn't know that because he wasn't up on science.
some of us don't have time for it.
He also really liked this girl robot whose name was: THE ECHELON 3500
He basically loved her, that's it. And he would have married her if she hadn't had to fight in the Great Robot Wars of their time. She died honorably for her fellow bots.
NOW WE'LL GET BACK TO THE STORY OF
the Tiger's Vest
in a moment. first, let's install Ruby...
BUT DON'T FORGET THE EARTH IS SECONDS AWAY FROM A SUN CRASH.

Let's install the very latest Ruby on your computer so you can follow all the examples in the (Poignant) Guide and actually do things right now! (Yes, things!)

Sidebar!

About Ruby Versions

1.9.3 is the latest **stable** version of

- If you are using **Microsoft Windows**, begin by downloading the Ruby Installer for Windows. Running this “one-click” installer will setup Ruby for you, as well as a tidy pack of useful software, such as a small text editor and some additional libraries.

- If you are on Apple’s **Mac OS X**, then Ruby is already installed. However, it is not the latest version available, so I encourage you to upgrade it by first installing Homebrew package manager, then in the Terminal run:

  ```
  brew install ruby
  ```

- On **Debian**, use `apt-get install ruby`.

- On **Gentoo**, `emerge ruby`.

To test if Ruby is installed, open a command shell and run: `ruby -v`. If Ruby is installed properly, you’ll see a bit of version info.

```
ruby 1.9.3p194 (2012-04-20) [x86_64-darwin12.0.0]
```

- To open a command shell in **Microsoft Windows**, go to the Start Menu and select `Run...`. Type in: `cmd`. Press OK. A command shell window will appear.

- To open a command shell on **Mac OS X**, run the **Terminal** application from **Spotlight**.

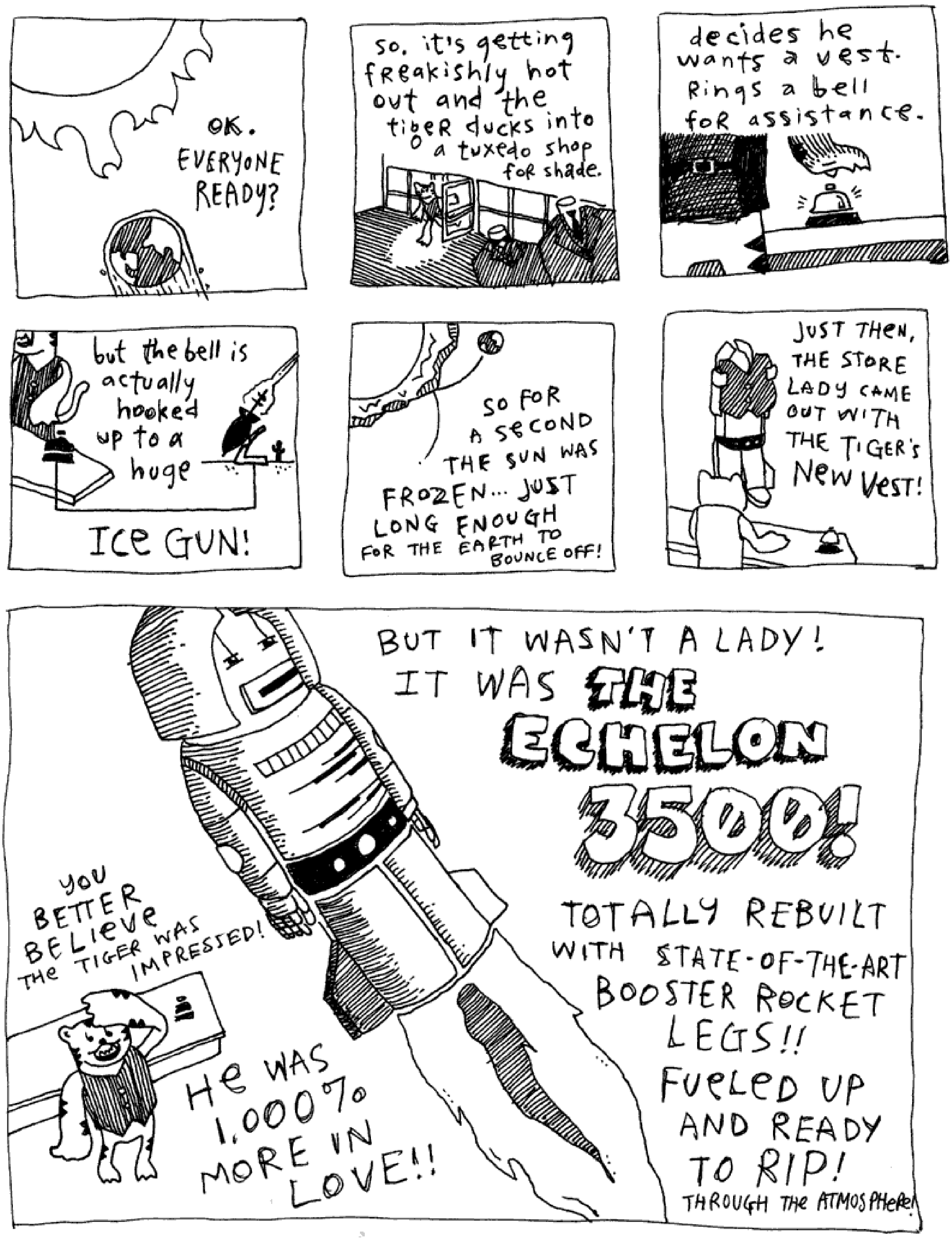
Okay, keep that command shell open cause we’ll need it if the Earth gets rescued from its plummet toward the sun.

Ruby. Stable versions offer the latest features which have been widely tested and are recommended for general use. When new stable versions come out, you can safely upgrade to these new versions.

Now, you’ll notice that the version number is comprised of three individual numbers, which are: **the major version**, **the minor version**, and **the teeny version**. The major version counts each complete rewrite of Ruby. The minor version is changed based on sweeping changes to Ruby. The teeny version is incremented every few months as smaller changes and bug fixes are brought in.

Oh, and if I could give you a taste of how smoothly Ruby moves along! I can’t count how often a fresh idea lands on the Ruby-Core mailing list, where the Emperors of Ruby converse, and **that same idea** gets worked into Ruby within a few days. It is a lovely machine says I.

Sidebar!


OK. EVERYONE READY?
so, it's getting freakishly hot out and the tiger ducks into a tuxedo shop for shade.
decides he wants a vest. Rings a bell for assistance.
but the bell is actually hooked up to a huge
ICE GUN!
SO FOR A SECOND THE SUN WAS FROZEN... JUST LONG ENOUGH FOR THE EARTH TO BOUNCE OFF!
JUST THEN, THE STORE LADY CAME OUT WITH THE TIGER'S NEW VEST!
BUT IT WASN'T A LADY! IT WAS THE ECHELON 3500!
TOTALLY REBUILT WITH STATE-OF-THE-ART BOOSTER ROCKET LEGS!! FUELED UP AND READY TO RIP! THROUGH THE ATMOSPHERE!
YOU BETTER BELIEVE THE TIGER WAS IMPRESSED!
HE WAS 1,000% MORE IN LOVE!!

Ruby comes with a very, very, very extremely helpful tool called **Irb**. Interactive Ruby. In your command shell, type: `irb`.

```
irb(main):001:0>
```

You should see the prompt above. This Irb prompt will allow you to enter Ruby code and, upon pressing *Enter*, the code will run.

So, at the Irb prompt, do a: `3000 + 500`.

```
irb(main):001:0> 3000 + 500
=> 3500
irb(main):002:0>
```

The example `3000 + 500` is legitimate code. We're just not assigning the answer to a variable. Which is okay in Irb, because Irb prints the answer given back by code you run.

Irb is a great calculator.

```
irb(main):002:0> ( ( 220.00 + 34.15 ) * 1.08 ) / 12
=> 22.8735
irb(main):003:0> "1011010".to_i( 2 )
=> 90
irb(main):004:0> Time.now - Time.local( 2003, "Jul", 31, 8, 10, 0 )
=> 31119052.510118
```

The first example demonstrates a bit of math and is read as: *220.00 plus 34.15 times 1.08 divided by 12*. The second example takes a binary string and converts it to a decimal number. The third example computes the number of seconds between `now` and July 31, 2003 at 8:10 AM. The answers to all of these are printed back to us by Irb with a little ASCII arrow pointing.

Reading the Prompt

I know the prompt surely looks bewildering. Well, let's not delay in dissecting it. It's very simple. The prompt has three parts, each separated by **colons**.

The first section, which reads `irb(main)`, shows **the name of the program** we are running. The second section shows a line number, **a count of how many lines of Ruby we've typed**. The third section is **a depth level**. Whenever you open a statement which needs closing, the depth level will increase by one. And whenever Irb senses that your code is unfinished, **the end of the prompt will become an asterisk.**

```
irb(main):001:0> bell = :pressed
=> :pressed
irb(main):002:0> ice_gun =
irb(main):003:0*   if bell == :pressed
irb(main):004:1>     :on
irb(main):005:1>   else
irb(main):006:1*     :off
irb(main):007:1>   end
=> :on
```

Notice how the depth level increased to 1 when I opened the `if` statement. And how the asterisk indicates the continuation of a line.

Tweaking the Prompt

You don't have to like the prompt's appearance, though. I'm not forcing you to do anything and, if you want to hack it up, I'm right here beside you.

Irb has a few other included prompts which may better appease your senses. Try `irb --prompt simple`. This time Irb will treat you to a very basic set of arrows, allowing you to enter your code without the whole status report.

```
>> %w(my best friend's arm)
=> ["my", "best", "friend's", "arm"]
>>
```

Ruby comes with a few prompts. The **simple** prompt seen above. The **xmp** prompt which has no prompt at all and indents the answer arrow. (It's supposed to look nice for printing.) Also, the **null** prompt, which eliminates prompts altogether. Set the prompt by just supplying the name with the `--prompt` option. (So, `irb --prompt null`.)

Making your own prompt is okay as well. Irb is completely customizable from inside of itself. The `conf` object contains all of Irb's configuration settings. Some of these configuration settings are for controlling the prompt.

```
>> conf.methods.grep /prompt/
=> ["prompting?", "prompt_s", "prompt_s=", "prompt_c", "prompt_c=",
    "prompt_i", "prompt_mode", "prompt_i=", "prompt_mode="]
```

Let's setup our prompt to display line numbers with just a bit of decor.

```
>> conf.prompt_i = "%3n :> "          # the normal prompt
>> conf.prompt_s = "%3n .%l "         # the string continuation prompt
>> conf.prompt_c = "%3n .^ "          # the code continuation prompt
>> conf.return_format = "    => %s\n" # the answer arrow
```

Above are the four parts to an Irb prompt. The string continuation prompt is displayed when a string is still open when you hit *Enter*. The `%3n` describes that Irb should reserve three characters for the line number. The `%l` saves a place for displaying the type of string being continued. (If you're continuing a double-quoted string, it shows a double quote. If you're continuing a regular expression, it shows a slash.)

The rest are little symbols to decorate the prompt. So, in the case of a continuing code line, I show a caret which points up to the line where that line of code started.

You can read more about customizing Irb and saving your configuration to a file in the complete guide to Irb, available in the free-for-your-wandering-Internet-eyes *Programming Ruby*.

Tab Completion

One feature of Irb which is rarely mentioned is its beneficial tab completion.

```
irb --readline -r irb/completion
```

Basically, when you hit *Tab*, Irb will take a guess at what you're trying to type. Try typing: `[].col` and hit *Tab*. Irb will finish it. `[].collect` and it leaves your cursor at the end so you can add further.

If there are several matches, just hitting *Tab* will do nothing. But hit it twice and Ruby will give you a complete list of possible matches.

This is great if you just want to see all the methods for a certain object. Type in any number, a dot, and use *Tab*.

```
>> 42.
              42.floor                    42.next                     42.step
42.__id__     42.freeze                   42.nil?                     42.succ
42.__send__   42.frozen?                  42.nonzero?                 42.taint
42.abs        42.hash                     42.object_id                42.tainted?
42.between?   42.id                       42.prec                     42.times
42.ceil       42.id2name                  42.prec_f                   42.to_f
42.chr        42.inspect                  42.prec_i                   42.to_i
42.class      42.instance_eval            42.private_methods          42.to_int
42.clone      42.instance_of?             42.protected_methods        42.to_s
42.coerce     42.instance_variable_get    42.public_methods           42.to_sym
42.display    42.instance_variable_set    42.quo                      42.truncate
42.div        42.instance_variables       42.remainder                42.type
42.divmod     42.integer?                 42.respond_to?              42.untaint
42.downto     42.is_a?                    42.round                    42.upto
42.dup        42.kind_of?                 42.send                     42.zero?
42.eql?       42.method                   42.singleton_method_added
42.equal?     42.methods                  42.singleton_methods
42.extend     42.modulo                   42.size
```

Now, trying typing `Kernel::` and then *Tab*. All the core methods. Don't ever forget this and use it all the time.

unfortunately, the tiger didn't chain the Echelon 3500 down, so she flew somewhere off somewhere else.

The Ice Gun got left on, so everybody had to leave that city.
by the way, this whole story takes place in Detroit. But not now that this happened.

Okay, one last thing and then I'll quit bugging you with all this great technology. But I have to say it loud, so take cover! I'm across the world here, folks, but the volume comes down from the sky—a bold, red crescendo of—

((ri))

(Ruby’s Own 411 or 555-1212 or *Yes, Operator, Get Belgrade on the Line*—I'll Be Right Here—Just Plain Hammering The Pound Key Until Someone Picks Up...)

And Ri picks up the line. “This is Ri. Class and colon, please.”

You rush in, “This is an instance method, Operator. Enumerable#zip.”

```
ri Enumerable#zip
```

Without delay, right up on your teletype display (so swiftly that even the cat perched atop cranes his neck around, gapes and hands it the royal cup *Most Blatantly Great Thing Since Michael Dorn*):

```
---------------------------------------------------------- Enumerable#zip
     enum.zip(arg, ...)                   => array
     enum.zip(arg, ...) {|arr| block }    => nil
------------------------------------------------------------------------
     Converts any arguments to arrays, then merges elements of _enum_
     with corresponding elements from each argument. This generates a
     sequence of +enum#size+ _n_-element arrays, where _n_ is one more
     that the count of arguments. If the size of any argument is less
     than +enum#size+, +nil+ values are supplied. If a block given, it
     is invoked for each output array, otherwise an array of arrays is
     returned.

        a = [ 4, 5, 6 ]
        b = [ 7, 8, 9 ]

        (1..3).zip(a, b)      #=> [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
        "cat\ndog".zip([1])   #=> [["cat\n", 1], ["dog", nil]]
        (1..3).zip            #=> [[1], [2], [3]]
```

It’s an unabridged Ruby dictionary servo—the Power of Just Asking is at your fingertips—*don’t tell me you’ve never heard of this no-money-down lifetime-supply-of-proper-explanations!*

To get an explanation of any class, along with a complete directory to all of its methods, all in a very soothing voice fit for calming any of you panicking cosmonauts out there fighting the pull of some zero-tolerance tractor beam, just use at your command shell: `ri Class`.

But for help on class methods, you cuff on: `ri Class::method`.

Though instance methods use a pound key rather than a dot. (Since the dot can mean class **or** instance methods.) I mean: `ri Class#method`.

The full spread of classes, a full list from the Very Top down to the Earth’s core, can be achieved with `ri -l`.

And beyond text, you can make HTML:

```
ri -Tf html String#gsub > gsub.html
```

Or show colory ANSI:

```
ri -Tf ansi String#gsub
```

And this is last and best.

Into the Ri Switchboard

Behind Ri sings a chorus of human voices, primarily Dave Thomas, an author of *Programming Ruby*, and absolutely the American foster parent of Ruby. Many of these elaborate spiels from Ri are straight from the references of *Programming Ruby*. Don't forget to thank Dave periodically.

Ri culls its lush information set from the very code that Ruby is built from. In each of the files of code back in the Ruby Headquarter's file cabinet, detailed comments describe everything in sight.

In Ruby's `date` class, here we have such commented methods:

```
# Get the time of this date as [hours, minutes, seconds,
# fraction_of_a_second]
def time() self.class.day_fraction_to_time(day_fraction) end

# Get the hour of this date.
def hour() time[0] end

# Get the minute of this date.
def min() time[1] end
```

The comments show up in Ri. We type: `ri Date#time`.

```
------------------------------------------------------------- Date#time
     time()
------------------------------------------------------------------------
     Get the time of this date as [hours, minutes, seconds,
     fraction_of_a_second]
```

Ri figures out much of how a method works, although it expects coders to write a brief description in the comments just before a method or class definition. I would suggest that whenever you write a method, add a brief description in the comments before that method. In time, you can generate Ri documentation for that method.

You can also use a few special characters to enhance your description. For example, if you indent a paragraph and use an asterisk `*` or a dash `-` just before the letters of the first sentence, the paragraph will be recognized as a list item. Then, if your description needs to be converted to HTML, you'll see the list item appear as an HTML unordered list.

```
# Get the time of this date as an Array of:
# * hours
# * minutes
# * seconds
# * fraction_of_a_second
def time() self.class.day_fraction_to_time(day_fraction) end
```

Other rules as well: Lists which start with digits followed by periods are recognized as numbered lists. Emphasized words are surrounded by underscores, bold words by asterisks, code words by plus signs. Examples are simply blocks of text indented a few spaces. All of these rules together are called RDoc.

Here's a bit of RDoc from the `initialize` method in one of my projects called RedCloth. Notice the indented example as well as the Ruby class and method names flanked with plusses.

```
#
# Returns a new RedCloth object, based on +String+ and
# enforcing all the included +restrictions+.
#
#   r = RedCloth.new( "h1. A <b>bold</b> man", [:filter_html] )
#   r.to_html
#     #=>"<h1>A &amp;lt;b&amp;gt;bold&amp;lt;/b&amp;gt; man</h1>"
#
def initialize( string, restrictions = [] )
    @lite = false
    restrictions.each { |r| method( "#{ r }=" ).call( true ) }
    super( string )
end
```

For the full set of RDoc rules see the **Markup** section of the README.

Pushing Out Your Own Ri

Ri doesn't automatically read your files for you, though. You've got to push it along, show it the way. Change to the directory of the code you'd like to scan. Then, use the RDoc tool to make it happen.

```
cd ~/cvs/your-code
rdoc --ri-site
```

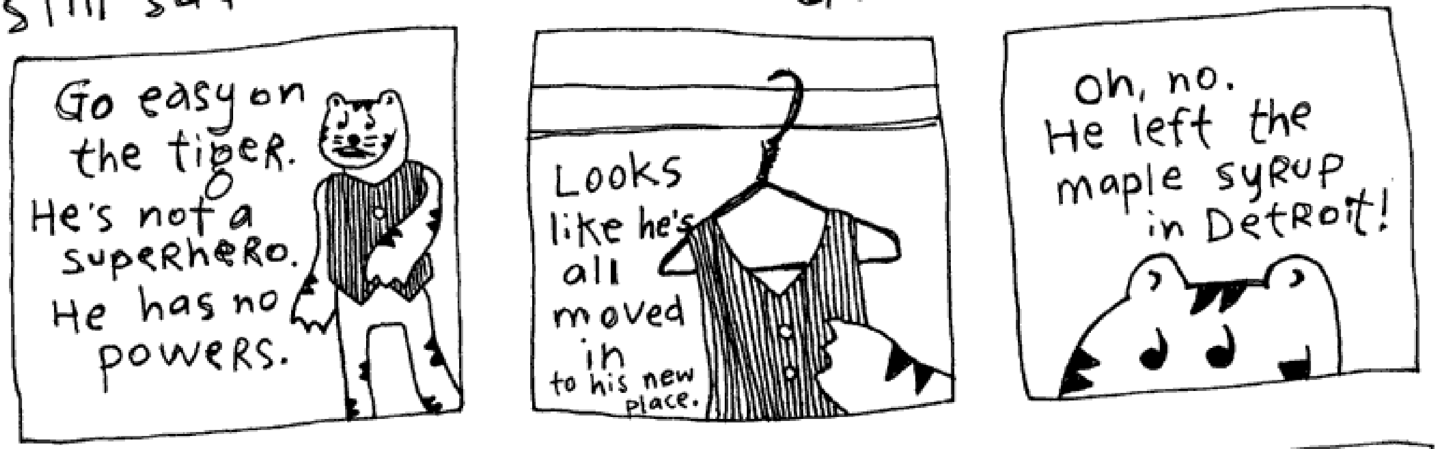
Now, try using `ri YourClass` to ensure all your descriptions are showing up properly. If you want to make HTML documentation, try this:

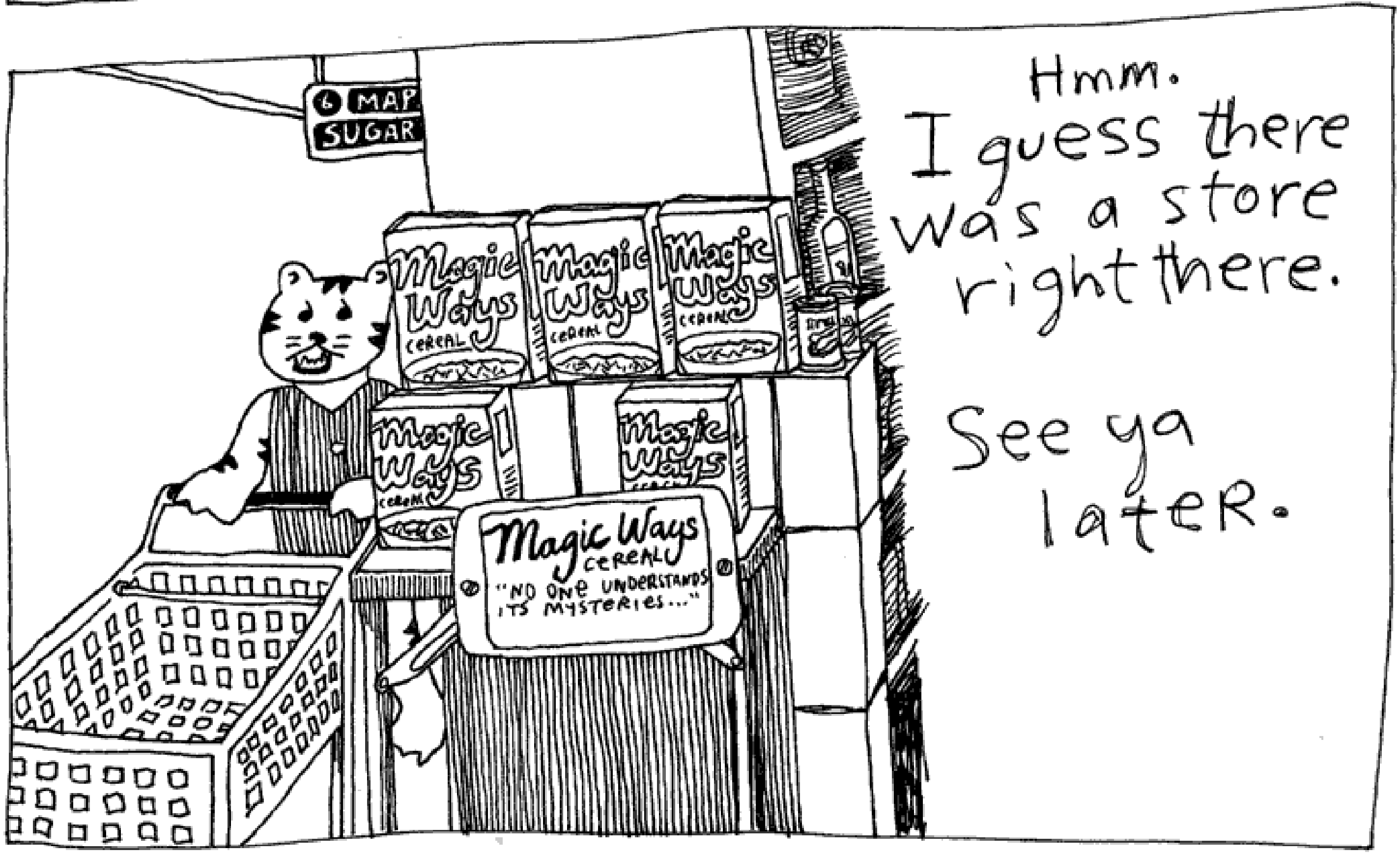
```
cd ~/cvs/your-code
rdoc
```

The `~/cvs/your-code` directory should now also have a fresh `doc` directory containing HTML documentation for all of your classes. View `index.html` for the pleasant news.

Well then. Your hands are in it all now. Welcome to Ruby.

Ok. Are we all still sad about the Ice Gun?


Go easy on the tiger. He's not a superhero. He has no powers.
Looks like he's all moved in to his new place.
Oh, no. He left the maple syrup in Detroit!


6 MAP
SUGAR
Magic Ways CEREAL
Magic Ways CEREAL "NO ONE UNDERSTANDS ITS MYSTERIES..."
Hmm. I guess there was a store right there.
See ya later.

Turn page.