Technical Explorations

by Keith R. Bennett

Clipboard Text Processing with Ruby

Apr 8, 2018

This article will explain how to integrate text processing scripts in Ruby (or any other language, scripting or otherwise) with *any* editor. First, some basics…

On the Mac, `pbcopy` and `pbpaste` copy to, and paste from, respectively, the system clipboard. (Unix has `xclip` and `xsel`, both of which do both copying and pasting). These utilities enable you manipulate the system clipboard on the command line. This can come in really handy. First, I'll show you how they work:

```
echo 'hello, world' | pbcopy
```

`pbcopy` takes its standard input and puts it in the system clipboard. Then, you can paste it into any application using `Cmd-V`. Or, you can use `pbpaste` to output the content of the system clipboard to stdout:

```
> pbpaste
hello, world
```

This bridge between text and graphical mode boundaries opens a whole world of possibilities for processing text in graphical applications.

Removing Formatting from Text

For me, the most common use case for using `pbcopy/pbpaste` is stripping away fancy formatting from text from graphical apps such as web browsers. Some editors, such as the GMail message editor, do not have a "Paste and Match Style" option. So if you want to paste text from a web page (for example) into your email message, it will probably be inserted with its original typeface, size, and color. Sometimes this is ok, but usually it will just be a glaring annoyance. Combining `pbpaste` and `pbcopy` as follows will remove all text formatting:

```
pbpaste | pbcopy
```

Extending Your Text Editor with pbcopy/pbpaste

Things got more interesting when I realized that these utilities open any GUI application up to the possibility of using scripts and command line utilities for processing their text. Admittedly, this is not a new idea; Vim and Textmate, for example, have this feature built into their software; but being able to use the clipboard enables this for *all* text editors.

I recently moved this blog from WordPress to Jekyll and Github Pages. I used the Wordpress to Jekyll Exporter plugin which, among other things, converted the HTML text to markdown format. The conversion was imperfect and I had to do some cleanup. There were 134 <pre> blocks like this:

```
<pre class="brush: ruby; title: ; notranslate" title="">def stringified_key_hash(numbers)
  numbers.each_with_object({}) do |n, hsh|
    hsh[n] = n.to_s
  end
end
</pre>
```

I needed to remove the `pre` tags and replace them with triple backticks, including the appropriate language when necessary (Ruby, Clojure, and Java in my case):

~~~
```ruby
def stringified_key_hash(numbers)
  numbers.each_with_object({}) do |n, hsh|
    hsh[n] = n.to_s
  end
end
```
~~~

Who knows if I really saved time by automating this, but I certainly aided my sanity, and got more practice in automating stuff as well.

On the highest level, the transformation works like this:

```
pbpaste | transform | pbcopy
```

I wanted to put the `pbpaste/pbcopy` handling in the script to simplify calling it, and that made the script a bit more complex. Here it is, with some nonessential code added for clearer and more informative output:

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

require 'nokogiri'
require 'trick_bag'


LANGUAGE = begin
  if ARGV[0].nil?
    nil
  else
    case ARGV[0][0].downcase
      when 'r'
        'ruby'
      when 'j'
        'java'
      when 'c'
        'clojure'
      else
        ''
    end
  end
end


def transform(s)
  text = Nokogiri::HTML(s).xpath('html/body/pre').text
  "```#{LANGUAGE}\n" + CGI.unescapeHTML(text) + "```\n"
end


def output_results(input, output)

  separator_line = "#{'-' * 79}\n"

  sandwich = ->(s) do
    '' << separator_line << s.chomp << "\n" << separator_line
````

```
  end

  puts separator_line
  puts "Input:\n#{sandwich.(input)}"
  puts "Output:\n#{sandwich.(output)}"
end


def copy_result_to_clipboard(result)
  TrickBag::Io::TempFiles.file_containing(result) do |temp_filespec|
    `cat #{temp_filespec} | pbcopy`
  end
end


input = `pbpaste`
output = transform(input)
output_results(input, output)
copy_result_to_clipboard(output)

```

Incidentally, notice the use of the `sandwich` lambda in the `output_results` method. This is one of my favorite use cases for lambdas - to encapsulate formatting done multiple times in the same method. By using a lambda for this we:

- eliminate duplication
- separate the lower level formatting code from the higher level code that calls it
- name the formatting behavior
- limit its scope to the method in which it is used, rather than polluting the broader class scope as an instance method

Here is an example of the output of a run using `strip-pre r` for Ruby highlighted code:

~~~
-------------------------------------------------------------------------------
Input:
-------------------------------------------------------------------------------
<pre class="brush: ruby; title: ; notranslate" title="">def stringified_key_hash(numbers)
  numbers.each_with_object({}) do |n, hsh|
    hsh[n] = n.to_s
  end
end
</pre>
-------------------------------------------------------------------------------
Output:
-------------------------------------------------------------------------------
```ruby
def stringified_key_hash(numbers)
  numbers.each_with_object({}) do |n, hsh|
    hsh[n] = n.to_s
  end
end
```
-------------------------------------------------------------------------------
~~~

The `transform` method is where all the text processing happens. Everything else supports or supplements that method.

The script is pretty standard Ruby. In the `copy_result_to_clipboard` method, I've used the `TrickBag::Io::TempFiles.file_containing` method to simplify creating a temp file with content, using it, then deleting it when done. This and other convenience methods can be found in my `trick_bag` gem on

Github here.

On first glance, that might seem like overkill, and it might seem sufficient to just `echo #{content} | pbcopy`, but that could be problematic. The string might be too large for a shell command. In addition, we would have to escape any characters modified or handled by the shell, such as multi-space strings, dollar signs, and backslashes. This could be done using `Shellwords.escape`, but since the TrickBag method was handy and solved both problems, I used that.

Other Uses for pbcopy/pbpaste

- Miguel Paraz says: "I `pbpaste | jq .` a lot, for grabbing from the browser's debugger and passing into back-ends under development."
- Eumir Gaspar says: "I needed a bunch of bank statements merged into one pdf. I added the filenames into a text file, did some minor vim magicks to escape the parentheses, and `cat statements.txt | pbcopy` so i can: `pdftk <cmd+v> cat output statements.pdf`.
- Chris Sexton uses `uuidgen | pbcopy` to generate UUID's and copy them to the clipboard.
- Marian Lucius Pop uses `cat ~/.ssh/id_rsa.pub | pbcopy` for copying his public key to the clipboard. Several other people mentioned using pbcopy with ssh keys.
- Hunter Madison suggests using it to add syntax highlighting to code, e.g. `cat <file> | highlight -O rtf | pbcopy`. (`highlight` is a syntax highlighting command line tool installable via `brew`.) He then pastes the highlighted code into his Keynote presentation.
- Björn Lindström - writes Jira comments in Vim, then "`:%!xclip -f` (X11 equivalent of pbcopy and pbpaste) and paste it into the browser".
- Ross Bragg says, when working with ssl, he uses `cat cert/key/intermediate | pbcopy` and pastes it into the console.

Conclusion

I hope I've illustrated the usefulness of using these utilities to cross the graphical/text mode boundary. If you find other interesting use cases, feel free to contact me on Twitter or Gmail as *keithrbennett*.

Published with GitHub Pages