Ruby 1.9 Common Problems Pt. 1: Encoding

By Loren Segal on February 2nd, 2009 at 1:05 AM

If you're migrating from Ruby 1.8.x to 1.9 you probably have run into one of the following error messages:

```
invalid multibyte char (US-ASCII)
```

- OR -

```
CompatibilityError: incompatible encoding regexp match (Windows-31J regexp with UTF-8 string)
```

The errors themselves are relatively self-explanatory. Ruby 1.9 is far more Unicode aware than 1.8, and this error happens when have some Unicode (usually UTF-8) in one of your files. What you have to do to fix these, however, is not always as straightforward.

Well, after some time pulling my hair out, I've figure out that the solutions to these issues are actually quite simple.

Invalid multibyte char (encoding here)

If you get this issue, add the following to the top of each exploding file (below the shebang if there is one):

```
# encoding: utf-8
```

> **Note:** You can also use "coding:" or even the Emacs style `-*- encoding: utf-8 -*-` but I like the simple term, 'encoding'. Also note that you might need to replace 'utf-8′ wth your specific encoding if it's something else.

This should resolve the issue.

Basically, Ruby by default assumes that every file is encoded as US-ASCII, and so when it reads UTF-8 (or any Unicode) it freaks out because it is beyond the 7-bit encoding. You have to tell it that the file is encoded as utf-8 by listing it as we did at the top of the file. Yes, it's a little anal, but I'm sure we'll get used to it.

Incompatible encoding regexp match

This issue is a little bit hairier. I had this problem with the BlueCloth 1.0.0 gem (on line 972 of bluecloth.rb). It turns out that there are a few switches that turn on specific encodings, and for some reason BlueCloth turns on the `//s` switch which enables the SJIS encoding (maybe John Gruber wanted `//m` for multiline?). Ruby 1.8 didn't mind having this on, but 1.9 freaks out. Moral of the story, when you see this error, check your Regexp switches.