**Mind the End of Your Line**

March, 2012

One of the most frequent questions I answer about Git is why dealing with line endings is so difficult. This is an attempt to answer that question and explain the myriad of options and settings that control line endings in Git.

Git has gone through two systems for dealing with line endings in repositories. The root of the problem being that Unix, Linux and OS X use LF and Windows uses CRLF to denote the end of a line. Previous to OS X, Mac actually used CR, but for the most part we can ignore that.

None of this would be a problem if we each lived in our own little worlds and never shared code between operating systems. And by share I mean everything from working on a cross platform project to copy-pasting code out of a browser. In fact, anytime you download a sample project in a zip file, copy code out of a gist, copy code from someones blog or use code out of a file that you keep in Dropbox - you are sharing text and you need to deal with these invisible line-ending characters. All of these activities potentially introduce a different set of line endings into your code base which is going to make diffs messy and Git generally unhappy.

Git’s primary solution to all this is to specify that LF is the best way to store line endings for *text files* in a Git repository’s object database. It doesn’t force this on you but most developers using Git and GitHub have adopted this as a convention and even our own [help](http://help.github.com/line-endings/) recommends setting up your config to do this.

**Background**

Before describing the settings that control line endings in Git, there are a couple of things you need to know about: core.eol and what it means to write something to the object database.

**End of line**

core.eol

The first setting you need to know about is core.eol. In all but the rarest of cases you should never have to change this setting from its default. This setting doesn’t do much on its own, but as soon as we start telling Git to change our line endings for us we need to know the value of core.eol. This setting is used by all the other things we are going to talk about below, so it’s good to know that it exists and good to know that you probably don’t want to change it.

* core.eol = native The default. When Git needs to change line endings to write a file in your working directory it will change them to whatever is the default line ending on your platform. For Windows this will be CRLF, for Unix/Linux/OS X this will be LF.
* core.eol = crlf When Git needs to change line endings to write a file in your working directory it will always use CRLF to denote end of line.
* core.eol = lf When Git needs to change line endings to write a file in your working directory it will always use LF to denote end of line.

You can run git config --global core.eol to see what this value is set to on your system. If nothing comes back that means you are on the using the default which is native.

**In and out of the object database**

What is the object database? I’m going to talk a lot about two different operations: writing to the object database and writing out to the working directory. It helps to understand these concepts a bit before moving on.

You may already know that Git has its own database in that .git folder which Scott does a great job of explaining in [Chapter 9](http://progit.org/book/ch9-0.html) of his [Pro Git book](http://progit.org/book/). All you need to know is that when you do something like git commit you are writing objects into the database. This involves taking the files that you are committing, calculating their shas and writing them into the object database as blobs. This is what I mean when I say *writing to the object database* and this is when Git has a chance to run filters and do things like converting line endings.

The other place that Git has a chance to run filters is when it reads out of the object database and writes files into your working directory. This is what I mean when I say *writing out into the working directory*. Many commands in Git do this, but git checkout is the most obvious and easy to understand. This also happens when you do a git clone or run a command like git reset that changes your working directory.

**The Old System**

First, let’s talk about the old system. This is the original set of features in Git designed to solve this particular problem of line endings. There is a good chance you are still using this system and don’t even know. Here is how it works: Git has a configuration setting called core.autocrlf which is specifically designed to make sure that when a *text* file is written to the repository’s object database that all line endings in that text file are normalized to LF. Here are the different options for core.autocrlf and what they mean:

* core.autocrlf = false This is the default, but most people are [encouraged](http://help.github.com/line-endings/) to change this immediately. The result of using false is that Git doesn’t ever mess with line endings on your file. You can check in files with LF or CRLF or CR or some random mix of those three and Git does not care. This can make diffs harder to read and merges more difficult. Most people working in a Unix/Linux world use this value because they don’t have CRLF problems and they don’t need Git to be doing extra work whenever files are written to the object database or written out into the working directory.
* core.autocrlf = true This means that Git will process all *text* files and make sure that CRLF is replaced with LF when writing that file to the object database and turn all LF back into CRLF when writing out into the working directory. This is the recommended setting on Windows because it ensures that your repository can be used on other platforms while retaining CRLF in your working directory.
* core.autocrlf = input This means that Git will process all *text* files and make sure that CRLF is replaced with LF when writing that file to the object database. It will **not**, however, do the reverse. When you read files back out of the object database and write them into the working directory they will still have LFs to denote the end of line. This setting is generally used on Unix/Linux/OS X to prevent CRLFs from getting written into the repository. The idea being that if you pasted code from a web browser and accidentally got CRLFs into one of your files, Git would make sure they were replaced with LFs when you wrote to the object database.

You can run git config --global core.autocrlf to see what this value is set to on your system. If nothing comes back that means you are on the using the default which is false.

How does Git know that a file is *text*? Good question. Git has an internal method for heuristically checking if a file is binary or not. A file is deemed *text* if it is not binary. Git can sometimes be wrong and this is the basis for our next setting.

The next setting that was introduced is core.safecrlf which is designed to protect against these cases where Git might change line endings on a file that really should just be left alone.

* core.safecrlf = true - When getting ready to run this operation of replacing CRLF with LF before writing to the object database, Git will make sure that it can actually successfully back out of the operation. It will verify that the reverse can happen (LF to CRLF) and if not the operation will be aborted.
* core.safecrlf = warn - Same as above, but instead of aborting the operation, Git will just warn you that something bad might happen.

One final layer on all this is that you can create a file called .gitattributes in the root of your repository and add rules for specific files. These rules allow you to control things like autocrlf on a per file basis. So you could, for instance, put this in that file to tell Git to always replace CRLF with LF in txt files:

\*.txt crlf

Or you could do this to tell Git to never replace CRLF with LF for txt files like this:

\*.txt -crlf

Or you could do this to tell Git to only replace CRLF with LF when writing, but to read back LF when writing the working directory for txt files like this:

\*.txt crlf=input

OK. Got all that? See all the problems and the mess we’ve made? It gets worse when you start working on projects that push you towards different global settings. Enter the new system which is available in Git 1.7.2 and above.

**The New System**

The new system moves to defining all of this in the .gitattributes file that you keep with your repository. This means that line endings can be encapsulated entirely within a repository and don’t depend on everyone having the proper global settings.

In the new system you are in charge of telling git which files you would like CRLF to LF replacement to be done on. This is done with a text attribute in your repository’s .gitattributes file. In this case the [man page](http://schacon.github.com/git/gitattributes.html) is actually quite helpful. Here are some examples of using the text attribute:

* \*.txt text Set all files matching the filter \*.txt to be text. This means that Git will run CRLF to LF replacement on these files every time they are written to the object database and the reverse replacement will be run when writing out to the working directory.
* \*.txt -text Unset all files matching the filter. These files will never run through the CRLF to LF replacement.
* \*.txt text=auto Set all files matching the filter to be converted (CRLF to LF) **if** those files are determined by Git to be text and not binary. This relies on Git’s built in binary detection heuristics.

If a file is unspecified then Git falls back to the core.autocrlf setting and you are back in the old system. This is how backwards compatibility is maintained, but I would recommend (especially for Windows developers) that you explicitly create a .gitattributes file.

Here is an example you might use for a C# project:

# These files are text and should be normalized (convert crlf =&gt; lf)

\*.cs text diff=csharp

\*.xaml text

\*.csproj text

\*.sln text

\*.tt text

\*.ps1 text

\*.cmd text

\*.msbuild text

\*.md text

# Images should be treated as binary

# (binary is a macro for -text -diff)

\*.png binary

\*.jepg binary

\*.sdf binary

One final note that the [man page for gitattributes](http://schacon.github.com/git/gitattributes.html) mentions is that you can tell git to detect *all* text files and automatically normalize them (convert CRLF to LF):

\* text=auto

This is certainly better than requiring everyone to be on the same global setting for core.autocrlf, but it means that you really trust Git to do binary detection properly. In my opinion it is better to explicitly specify your text files that you want normalized. Don’t forget if you are going to use this setting that it should be the first line in your .gitattributes file so that subsequent lines can override that setting.

Building [GitHub](https://github.com) since 2011, programming language connoisseur, San Francisco resident, aspiring surfer, father of two, life partner to [@ktkates](https://github.com/ktkates)—all words by me, [Tim Clem](https://adaptivepatchwork.com/about/)

**https://adaptivepatchwork.com/2012/03/01/mind-the-end-of-your-line/**