# Configuring Git to handle line endings

**In this article**



To avoid problems in your diffs, you can configure Git to properly handle line endings.

Mac Windows Linux

## About line endings

Every time you press `return` on your keyboard you insert an invisible character called a line ending. Different operating systems handle line endings differently.

When you're collaborating on projects with Git and GitHub Enterprise Cloud, Git might produce unexpected results if, for example, you're working on a Windows machine, and your collaborator has made a change in macOS.

You can configure Git to handle line endings automatically so you can collaborate effectively with people who use different operating systems.

## Global settings for line endings

The `git config core.autocrlf` command is used to change how Git handles line endings. It takes a single argument.

On macOS, you simply pass `input` to the configuration. For example:

```
$ git config --global core.autocrlf input
# Configure Git to ensure line endings in files you checkout are correct for
macOS
```

On Windows, you simply pass `true` to the configuration. For example:

```
$ git config --global core.autocrlf true
# Configure Git to ensure line endings in files you checkout are correct for
Windows.
# For compatibility, line endings are converted to Unix style when you commit
files.
```

On Linux, you simply pass `input` to the configuration. For example:

```
$ git config --global core.autocrlf input
# Configure Git to ensure line endings in files you checkout are correct for
```

```
Linux
```

## Per-repository settings

Optionally, you can configure a `.gitattributes` file to manage how Git reads line endings in a specific repository. When you commit this file to a repository, it overrides the `core.autocrlf` setting for all repository contributors. This ensures consistent behavior for all users, regardless of their Git settings and environment.

The `.gitattributes` file must be created in the root of the repository and committed like any other file.

A `.gitattributes` file looks like a table with two columns:

- On the left is the file name for Git to match.
- On the right is the line ending configuration that Git should use for those files.

### Example

Here's an example `.gitattributes` file. You can use it as a template for your repositories:

```
# Set the default behavior, in case people don't have core.autocrlf set.
* text=auto

# Explicitly declare text files you want to always be normalized and converted
# to native line endings on checkout.
*.c text
*.h text

# Declare files that will always have CRLF line endings on checkout.
*.sln text eol=crlf

# Denote all files that are truly binary and should not be modified.
*.png binary
*.jpg binary
```

You'll notice that files are matched—`*.c`, `*.sln`, `*.png`—, separated by a space, then given a setting—`text`, `text eol=crlf`, `binary`. We'll go over some possible settings below.

- `text=auto` Git will handle the files in whatever way it thinks is best. This is a good default option.
- `text eol=crlf` Git will always convert line endings to `CRLF` on checkout. You should use this for files that must keep `CRLF` endings, even on OSX or Linux.
- `text eol=lf` Git will always convert line endings to `LF` on checkout. You should use this for files that must keep LF endings, even on Windows.
- `binary` Git will understand that the files specified are not text, and it should not try to change them. The `binary` setting is also an alias for `-text -diff`.

## Refreshing a repository after changing line endings

After you set the `core.autocrlf` option or commit a `.gitattributes` file, Git automatically changes line endings to match your new configuration. You may find that Git reports changes to files that you have not modified.

To ensure that all the line endings in your repository match your new configuration, back up your files with Git, then remove and restore all of the files to normalize the line endings.

1. Before adding or committing any changes, verify that Git has applied the configuration correctly. For example, Git automatically determines whether files in a repository are text or binary files. To avoid corruption of binary files in your repository, we recommend that you explicitly mark files as binary in `.gitattributes`. For more information, see gitattributes - Defining attributes per path in the Git documentation.

2. To avoid losing any local changes to files in the repository, add and commit any outstanding changes by running the following commands.

   Shell
   ```
   git add . -u
   git commit -m "Saving files before refreshing line endings"
   ```

3. To update all files on the current branch to reflect the new configuration, run the following commands.

   Shell
   ```
   git rm -rf --cached .
   git reset --hard HEAD
   ```

4. To display the rewritten, normalized files, run the following command.

   Shell
   ```
   git status
   ```

5. Optionally, to commit any outstanding changes in your repository, run the following command.

   Shell
   ```
   git commit -m "Normalize all the line endings"
   ```

## Further reading

- Customizing Git - Git Attributes in the Pro Git book
- git-config in the man pages for Git
- Getting Started - First-Time Git Setup in the Pro Git book
- Mind the End of Your Line by Tim Clem