NAME

gitattributes − defining attributes per path

SYNOPSIS

$GIT_DIR/info/attributes, .gitattributes

DESCRIPTION

A gitattributes file is a simple text file that gives attributes to pathnames.

Each line in gitattributes file is of form:

```
pattern attr1 attr2 ...
```

That is, a pattern followed by an attributes list, separated by whitespaces. When the pattern matches the path in question, the attributes listed on the line are given to the path.

Each attribute can be in one of these states for a given path:

Set
: The path has the attribute with special value "true"; this is specified by listing only the name of the attribute in the attribute list.

Unset
: The path has the attribute with special value "false"; this is specified by listing the name of the attribute prefixed with a dash − in the attribute list.

Set to a value
: The path has the attribute with specified string value; this is specified by listing the name of the attribute followed by an equal sign = and its value in the attribute list.

Unspecified
: No pattern matches the path, and nothing says if the path has or does not have the attribute, the attribute for the path is said to be Unspecified.

When more than one pattern matches the path, a later line overrides an earlier line. This overriding is done per attribute. The rules how the pattern matches paths are the same as in .gitignore files; see **gitignore**(5).

When deciding what attributes are assigned to a path, git consults $GIT_DIR/info/attributes file (which has the highest precedence), .gitattributes file in the same directory as the path in question, and its parent directories up to the toplevel of the work tree (the further the directory that contains .gitattributes is from the path in question, the lower its precedence).

If you wish to affect only a single repository (i.e., to assign attributes to files that are particular to one user's workflow), then attributes should be placed in the $GIT_DIR/info/attributes file. Attributes which should be version−controlled and distributed to other repositories (i.e., attributes of interest to all users) should go into .gitattributes files.

Sometimes you would need to override an setting of an attribute for a path to unspecified state. This can be done by listing the name of the attribute prefixed with an exclamation point !.

EFFECTS

Certain operations by git can be influenced by assigning particular attributes to a path. Currently, the following operations are attributes−aware.

Checking−out and checking−in

These attributes affect how the contents stored in the repository are copied to the working tree files when commands such as *git checkout* and *git merge* run. They also affect how git stores the contents you prepare in the working tree in the repository upon *git add* and *git commit*.

crlf

This attribute controls the line−ending convention.

Set
: Setting the crlf attribute on a path is meant to mark the path as a "text" file. *core.autocrlf* conversion takes place without guessing the content type by inspection.

Unset
: Unsetting the crlf attribute on a path tells git not to attempt any end−of−line conversion upon checkin or checkout.

Unspecified
: Unspecified crlf attribute tells git to apply the core.autocrlf conversion when the file content looks like text.

Set to string value "input"
: This is similar to setting the attribute to true, but also forces git to act as if core.autocrlf is set to input for the path.

Any other value set to crlf attribute is ignored and git acts as if the attribute is left unspecified.

The core.autocrlf conversion

If the configuration variable core.autocrlf is false, no conversion is done.

When core.autocrlf is true, it means that the platform wants CRLF line endings for files in the working tree, and you want to convert them back to the normal LF line endings when checking in to the repository.

When core.autocrlf is set to "input", line endings are converted to LF upon checkin, but there is no conversion done upon checkout.

If core.safecrlf is set to "true" or "warn", git verifies if the conversion is reversible for the current setting of core.autocrlf. For "true", git rejects irreversible conversions; for "warn", git only prints a warning but accepts an irreversible conversion. The safety triggers to prevent such a conversion done to the files in the work tree, but there are a few exceptions. Even though...

- *git add* itself does not touch the files in the work tree, the next checkout would, so the safety triggers;
- *git apply* to update a text file with a patch does touch the files in the work tree, but the operation is about text files and CRLF conversion is about fixing the line ending inconsistencies, so the safety does not trigger;
- *git diff* itself does not touch the files in the work tree, it is often run to inspect the changes you intend to next *git add*. To catch potential problems early, safety triggers.

ident

When the attribute ident is set for a path, git replaces $Id$ in the blob object with $Id:, followed by the 40−character hexadecimal blob object name, followed by a dollar sign $ upon checkout. Any byte sequence that begins with $Id: and ends with $ in the worktree file is replaced with $Id$ upon check−in.

filter

A filter attribute can be set to a string value that names a filter driver specified in the configuration.

A filter driver consists of a clean command and a smudge command, either of which can be left unspecified. Upon checkout, when the smudge command is specified, the command is fed the blob object from its standard input, and its standard output is used to update the worktree file. Similarly, the clean command is used to convert the contents of worktree file upon checkin.

A missing filter driver definition in the config is not an error but makes the filter a no−op passthru.

The content filtering is done to massage the content into a shape that is more convenient for the platform, filesystem, and the user to use. The key phrase here is "more convenient" and not "turning something unusable into usable". In other words, the intent is that if someone unsets the filter driver definition, or does not have the appropriate filter program, the project should still be usable.

For example, in .gitattributes, you would assign the filter attribute for paths.

```
*.c     filter=indent
```

Then you would define a "filter.indent.clean" and "filter.indent.smudge" configuration in your .git/config to specify a pair of commands to modify the contents of C programs when the source files are checked in ("clean" is run) and checked out (no change is made because the command is "cat").

```
[filter "indent"]
        clean = indent
        smudge = cat
```

Interaction between checkin/checkout attributes

In the check−in codepath, the worktree file is first converted with filter driver (if specified and corresponding driver defined), then the result is processed with ident (if specified), and then finally with crlf (again, if specified and applicable).

In the check−out codepath, the blob content is first converted with crlf, and then ident and fed to filter.

Generating diff text

diff

The attribute diff affects how *git* generates diffs for particular files. It can tell git whether to generate a textual patch for the path or to treat the path as a binary file. It can also affect what line is shown on the hunk header @@ −k,l +n,m @@ line, tell git to use an external command to generate the diff, or ask git to convert binary files to a text format before generating the diff.

Set
: A path to which the diff attribute is set is treated as text, even when they contain byte values that normally never appear in text files, such as NUL.

Unset
: A path to which the diff attribute is unset will generate Binary files differ (or a binary patch, if binary patches are enabled).

Unspecified
: A path to which the diff attribute is unspecified first gets its contents inspected, and if it looks like text, it is treated as text. Otherwise it would generate Binary files differ.

String
: Diff is shown using the specified diff driver. Each driver may specify one or more options, as described in the following section. The options for the diff driver "foo" are defined by the configuration variables in the "diff.foo" section of the git config file.

Defining an external diff driver

The definition of a diff driver is done in gitconfig, not gitattributes file, so strictly speaking this manual page is a wrong place to talk about it. However...

To define an external diff driver jcdiff, add a section to your $GIT_DIR/config file (or

$HOME/.gitconfig file) like this:

```
[diff "jcdiff"]
        command = j−c−diff
```

When git needs to show you a diff for the path with diff attribute set to jcdiff, it calls the command you specified with the above configuration, i.e. j−c−diff, with 7 parameters, just like GIT_EXTERNAL_DIFF program is called. See **git**(1) for details.

Defining a custom hunk-header

Each group of changes (called a "hunk") in the textual diff output is prefixed with a line of the form:

```
@@ −k,l +n,m @@ TEXT
```

This is called a *hunk header*. The "TEXT" portion is by default a line that begins with an alphabet, an underscore or a dollar sign; this matches what GNU *diff −p* output uses. This default selection however is not suited for some contents, and you can use a customized pattern to make a selection.

First, in .gitattributes, you would assign the diff attribute for paths.

```
*.tex   diff=tex
```

Then, you would define a "diff.tex.xfuncname" configuration to specify a regular expression that matches a line that you would want to appear as the hunk header "TEXT". Add a section to your $GIT_DIR/config file (or $HOME/.gitconfig file) like this:

```
[diff "tex"]
        xfuncname = "^(\\\\(sub)*section\\{.*)$"
```

Note. A single level of backslashes are eaten by the configuration file parser, so you would need to double the backslashes; the pattern above picks a line that begins with a backslash, and zero or more occurrences of sub followed by section followed by open brace, to the end of line.

There are a few built−in patterns to make this easier, and tex is one of them, so you do not have to write the above in your configuration file (you still need to enable this with the attribute mechanism, via .gitattributes). The following built in patterns are available:

- bibtex suitable for files with BibTeX coded references.
- cpp suitable for source code in the C and C++ languages.
- html suitable for HTML/XHTML documents.
- java suitable for source code in the Java language.
- objc suitable for source code in the Objective−C language.
- pascal suitable for source code in the Pascal/Delphi language.
- php suitable for source code in the PHP language.
- python suitable for source code in the Python language.
- ruby suitable for source code in the Ruby language.
- tex suitable for source code for LaTeX documents.

Customizing word diff

You can customize the rules that git diff −−color−words uses to split words in a line, by specifying an appropriate regular expression in the "diff.*.wordRegex" configuration variable. For example, in TeX a backslash followed by a sequence of letters forms a command, but several such commands can be run together without intervening whitespace. To separate them, use a regular expression in your $GIT_DIR/config file (or $HOME/.gitconfig file) like this:

```
[diff "tex"]
        wordRegex = "\\\\[a−zA−Z]+|[{}]|\\\\.|[ˆ\\{}[:space:]]+"
```

A built−in pattern is provided for all languages listed in the previous section.

Performing text diffs of binary files

Sometimes it is desirable to see the diff of a text−converted version of some binary files. For example, a word processor document can be converted to an ASCII text representation, and the diff of the text shown. Even though this conversion loses some information, the resulting diff is useful for human viewing (but cannot be applied directly).

The textconv config option is used to define a program for performing such a conversion. The program should take a single argument, the name of a file to convert, and produce the resulting text on stdout.

For example, to show the diff of the exif information of a file instead of the binary information (assuming you have the exif tool installed), add the following section to your $GIT_DIR/config file (or $HOME/.gitconfig file):

```
[diff "jpg"]
        textconv = exif
```

Note

The text conversion is generally a one−way conversion; in this example, we lose the actual image contents and focus just on the text data. This means that diffs generated by textconv are *not* suitable for applying. For this reason, only git diff and the git log family of commands (i.e., log, whatchanged, show) will perform text conversion. git format−patch will never generate this output. If you want to send somebody a text−converted diff of a binary file (e.g., because it quickly conveys the changes you have made), you should generate it separately and send it as a comment *in addition to* the usual binary diff that you might send.

Performing a three−way merge

merge

The attribute merge affects how three versions of a file is merged when a file−level merge is necessary during git merge, and other commands such as git revert and git cherry−pick.

Set

Built−in 3−way merge driver is used to merge the contents in a way similar to *merge* command of RCS suite. This is suitable for ordinary text files.

Unset

Take the version from the current branch as the tentative merge result, and declare that the merge has conflicts. This is suitable for binary files that does not have a well−defined merge semantics.

Unspecified

By default, this uses the same built−in 3−way merge driver as is the case the merge attribute is set. However, merge.default configuration variable can name different merge driver to be used for paths to which the merge attribute is unspecified.

String

3−way merge is performed using the specified custom merge driver. The built−in 3−way merge driver can be explicitly specified by asking for "text" driver; the built−in "take the current branch" driver can be requested with "binary".

Built-in merge drivers

There are a few built−in low−level merge drivers defined that can be asked for via the merge attribute.

text
: Usual 3−way file level merge for text files. Conflicted regions are marked with conflict markers <<<<<<<, ======= and >>>>>>>. The version from your branch appears before the ======= marker, and the version from the merged branch appears after the ======= marker.

binary
: Keep the version from your branch in the work tree, but leave the path in the conflicted state for the user to sort out.

union
: Run 3−way file level merge for text files, but take lines from both versions, instead of leaving conflict markers. This tends to leave the added lines in the resulting file in random order and the user should verify the result. Do not use this if you do not understand the implications.

Defining a custom merge driver

The definition of a merge driver is done in the .git/config file, not in the gitattributes file, so strictly speaking this manual page is a wrong place to talk about it. However...

To define a custom merge driver filfre, add a section to your $GIT_DIR/config file (or $HOME/.gitconfig file) like this:

```
[merge "filfre"]
        name = feel−free merge driver
        driver = filfre %O %A %B
        recursive = binary
```

The merge.*.name variable gives the driver a human−readable name.

The ‘merge.*.driver‘ variable’s value is used to construct a command to run to merge ancestor’s version (%O), current version (%A) and the other branches’ version (%B). These three tokens are replaced with the names of temporary files that hold the contents of these versions when the command line is built. Additionally, %L will be replaced with the conflict marker size (see below).

The merge driver is expected to leave the result of the merge in the file named with %A by overwriting it, and exit with zero status if it managed to merge them cleanly, or non−zero if there were conflicts.

The merge.*.recursive variable specifies what other merge driver to use when the merge driver is called for an internal merge between common ancestors, when there are more than one. When left unspecified, the driver itself is used for both internal merge and the final merge.

conflict-marker-size

This attribute controls the length of conflict markers left in the work tree file during a conflicted merge. Only setting to the value to a positive integer has any meaningful effect.

For example, this line in .gitattributes can be used to tell the merge machinery to leave much longer (instead of the usual 7−character−long) conflict markers when merging the file Documentation/git−merge.txt results in a conflict.

```
Documentation/git−merge.txt     conflict−marker−size=32
```

Checking whitespace errors

whitespace

The core.whitespace configuration variable allows you to define what *diff* and *apply* should consider whitespace errors for all paths in the project (See **git-config**(1)). This attribute gives you finer control per path.

Set
: Notice all types of potential whitespace errors known to git.

Unset
: Do not notice anything as error.

Unspecified
: Use the value of core.whitespace configuration variable to decide what to notice as error.

String
: Specify a comma separate list of common whitespace problems to notice in the same format as core.whitespace configuration variable.

Creating an archive

export-ignore

Files and directories with the attribute export−ignore won’t be added to archive files.

export-subst

If the attribute export−subst is set for a file then git will expand several placeholders when adding this file to an archive. The expansion depends on the availability of a commit ID, i.e., if **git-archive**(1) has been given a tree instead of a commit or a tag then no replacement will be done. The placeholders are the same as those for the option −−pretty=format: of **git-log**(1), except that they need to be wrapped like this: $Format:PLACEHOLDERS$ in the file. E.g. the string $Format:%H$ will be replaced by the commit hash.

Packing objects

delta

Delta compression will not be attempted for blobs for paths with the attribute delta set to false.

Viewing files in GUI tools

encoding

The value of this attribute specifies the character encoding that should be used by GUI tools (e.g. **gitk**(1) and **git-gui**(1)) to display the contents of the relevant file. Note that due to performance considerations **gitk**(1) does not use this attribute unless you manually enable per−file encodings in its options.

If this attribute is not set or has an invalid value, the value of the gui.encoding configuration variable is used instead (See **git-config**(1)).

USING ATTRIBUTE MACROS

You do not want any end−of−line conversions applied to, nor textual diffs produced for, any binary file you track. You would need to specify e.g.

```
*.jpg −crlf −diff
```

but that may become cumbersome, when you have many attributes. Using attribute macros, you can specify groups of attributes set or unset at the same time. The system knows a built−in attribute macro, binary:

```
*.jpg binary
```

which is equivalent to the above. Note that the attribute macros can only be "Set" (see the above example that sets "binary" macro as if it were an ordinary attribute −−− setting it in turn unsets "crlf" and "diff").

DEFINING ATTRIBUTE MACROS

Custom attribute macros can be defined only in the .gitattributes file at the toplevel (i.e. not in any subdirectory). The built−in attribute macro "binary" is equivalent to:

```
[attr]binary −diff −crlf
```

EXAMPLE

If you have these three gitattributes file:

```
(in $GIT_DIR/info/attributes)

a*      foo !bar −baz

(in .gitattributes)
abc     foo bar baz

(in t/.gitattributes)
ab*     merge=filfre
abc     −foo −bar
*.c     frotz
```

the attributes given to path t/abc are computed as follows:

1. By examining t/.gitattributes (which is in the same directory as the path in question), git finds that the first line matches. merge attribute is set. It also finds that the second line matches, and attributes foo and bar are unset.
2. Then it examines .gitattributes (which is in the parent directory), and finds that the first line matches, but t/.gitattributes file already decided how merge, foo and bar attributes should be given to this path, so it leaves foo and bar unset. Attribute baz is set.
3. Finally it examines $GIT_DIR/info/attributes. This file is used to override the in−tree settings. The first line is a match, and foo is set, bar is reverted to unspecified state, and baz is unset.

As the result, the attributes assignment to t/abc becomes:

```
foo     set to true
bar     unspecified
baz     set to false
merge   set to string value "filfre"
frotz   unspecified
```

GIT

Part of the **git**(1) suite