git-config(1) Manual Page

NAME

git-config - Get and set repository or global options

SYNOPSIS

git config [<file-option>] [type] [-z|--null] name [value [value_regex]]
git config [<file-option>] [type] --add name value
git config [<file-option>] [type] --replace-all name value [value_regex]
git config [<file-option>] [type] [-z|--null] --get name [value_regex]
git config [<file-option>] [type] [-z|--null] --get-all name [value_regex]
git config [<file-option>] [type] [-z|--null] --get-regexp name_regex [value_regex]
git config [<file-option>] --unset name [value_regex]
git config [<file-option>] --unset-all name [value_regex]
git config [<file-option>] --rename-section old_name new_name
git config [<file-option>] --remove-section name
git config [<file-option>] [-z|--null] -l | --list
git config [<file-option>] --get-color name [default]
git config [<file-option>] --get-colorbool name [stdout-is-tty]
git config [<file-option>] -e | --edit

DESCRIPTION

You can query/set/replace/unset options with this command. The name is actually the section and the key separated by a dot, and the value will be escaped.

Multiple lines can be added to an option by using the *--add* option. If you want to update or unset an option which can occur on multiple lines, a POSIX regexp `value_regex` needs to be given. Only the existing values that match the regexp are updated or unset. If you want to handle the lines that do **not** match the regex, just prepend a single exclamation mark in front (see also [EXAMPLES]).

The type specifier can be either *--int* or *--bool*, to make *git config* ensure that the variable(s) are of the given type and convert the value to the canonical form (simple decimal number for int, a "true" or "false" string for bool), or *--path*, which does some path expansion (see *--path* below). If no type specifier is passed, no checks or transformations are performed on the value.

When reading, the values are read from the system, global and repository local configuration files by default, and options *--system*, *--global*, *--local* and *--file <filename>* can be used to tell the command to read from only that location (see [FILES]).

When writing, the new value is written to the repository local configuration file by default, and options *--system*, *--global*, *--file <filename>* can be used to tell the command to write to

that location (you can say *--local* but that is the default).

This command will fail with non-zero status upon error. Some exit codes are:

1. The config file is invalid (ret=3),
2. can not write to the config file (ret=4),
3. no section or name was provided (ret=2),
4. the section or key is invalid (ret=1),
5. you try to unset an option which does not exist (ret=5),
6. you try to unset/set an option for which multiple lines match (ret=5), or
7. you try to use an invalid regexp (ret=6).

On success, the command returns the exit code 0.

OPTIONS

--replace-all
: Default behavior is to replace at most one line. This replaces all lines matching the key (and optionally the value_regex).

--add
: Adds a new line to the option without altering any existing values. This is the same as providing ^$ as the value_regex in `--replace-all`.

--get
: Get the value for a given key (optionally filtered by a regex matching the value). Returns error code 1 if the key was not found and error code 2 if multiple key values were found.

--get-all
: Like get, but does not fail if the number of values for the key is not exactly one.

--get-regexp
: Like --get-all, but interprets the name as a regular expression and writes out the key names. Regular expression matching is currently case-sensitive and done against a canonicalized version of the key in which section and variable names are lowercased, but subsection names are not.

--global
: For writing options: write to global /.gitconfig file rather than the repository .git/config, write to $XDG_CONFIG_HOME/git/config file if this file exists and the /.gitconfig file doesn't.

  For reading options: read only from global ~/.gitconfig and from $XDG_CONFIG_HOME/git/config rather than from all available files.

  See also [FILES].

--system
: For writing options: write to system-wide $(prefix)/etc/gitconfig rather than the repository .git/config.

  For reading options: read only from system-wide $(prefix)/etc/gitconfig rather than from all available files.

  See also [FILES].

-f config-file

--file config-file
: Use the given config file instead of the one specified by GIT_CONFIG.

--remove-section
: Remove the given section from the configuration file.

--rename-section
: Rename the given section to a new name.

--unset
: Remove the line matching the key from config file.

--unset-all
: Remove all lines matching the key from config file.

-l

--list
: List all variables set in config file.

--bool
: *git config* will ensure that the output is "true" or "false"

--int
: *git config* will ensure that the output is a simple decimal number. An optional value suffix of *k*, *m*, or *g* in the config file will cause the value to be multiplied by 1024, 1048576, or 1073741824 prior to output.

--bool-or-int
: *git config* will ensure that the output matches the format of either --bool or --int, as described above.

--path
: *git-config* will expand leading ~ to the value of *$HOME*, and *~user* to the home directory for the specified user. This option has no effect when setting the value (but you can use *git config bla ~/* from the command line to let your shell do the expansion).

-z

--null
: For all options that output values and/or keys, always end values with the null character (instead of a newline). Use newline instead as a delimiter between key and value. This allows for secure parsing of the output without getting confused e.g. by values that contain line breaks.

--get-colorbool name [stdout-is-tty]
: Find the color setting for `name` (e.g. `color.diff`) and output "true" or "false". `stdout-is-tty` should be either "true" or "false", and is taken into account when configuration says "auto". If `stdout-is-tty` is missing, then checks the standard output of the command itself, and exits with status 0 if color is to be used, or exits with status 1 otherwise. When the color setting for `name` is undefined, the command uses `color.ui` as fallback.

--get-color name [default]
: Find the color configured for `name` (e.g. `color.diff.new`) and output it as the ANSI color escape sequence to the standard output. The optional `default` parameter is used instead, if there is no color configured for `name`.

-e

--edit

Opens an editor to modify the specified config file; either *--system*, *--global*, or repository (default).

--includes

--no-includes

Respect `include.*` directives in config files when looking up values. Defaults to on.

FILES

If not set explicitly with *--file*, there are four files where *git config* will search for configuration options:

$GIT_DIR/config

Repository specific configuration file.

~/.gitconfig

User-specific configuration file. Also called "global" configuration file.

$XDG_CONFIG_HOME/git/config

Second user-specific configuration file. If $XDG_CONFIG_HOME is not set or empty, $HOME/.config/git/config will be used. Any single-valued variable set in this file will be overwritten by whatever is in ~/.gitconfig. It is a good idea not to create this file if you sometimes use older versions of Git, as support for this file was added fairly recently.

$(prefix)/etc/gitconfig

System-wide configuration file.

If no further options are given, all reading options will read all of these files that are available. If the global or the system-wide configuration file are not available they will be ignored. If the repository configuration file is not available or readable, *git config* will exit with a non-zero error code. However, in neither case will an error message be issued.

All writing options will per default write to the repository specific configuration file. Note that this also affects options like *--replace-all* and *--unset*. ***git config* will only ever change one file at a time**.

You can override these rules either by command line options or by environment variables. The *--global* and the *--system* options will limit the file used to the global or system-wide file respectively. The GIT_CONFIG environment variable has a similar effect, but you can specify any filename you want.

ENVIRONMENT

GIT_CONFIG

Take the configuration from the given file instead of .git/config. Using the "--global" option forces this to ~/.gitconfig. Using the "--system" option forces this to $(prefix)/etc/gitconfig.

GIT_CONFIG_NOSYSTEM

Whether to skip reading settings from the system-wide $(prefix)/etc/gitconfig file. See git(1) for details.

See also [FILES].

EXAMPLES

Given a .git/config like this:

```
#
# This is the config file, and
# a '#' or ';' character indicates
# a comment
#

; core variables
[core]
        ; Don't trust file modes
        filemode = false

; Our diff algorithm
[diff]
        external = /usr/local/bin/diff-wrapper
        renames = true

; Proxy settings
[core]
        gitproxy=proxy-command for kernel.org
        gitproxy=default-proxy ; for all the rest
```

you can set the filemode to true with

```
% git config core.filemode true
```

The hypothetical proxy command entries actually have a postfix to discern what URL they apply to. Here is how to change the entry for kernel.org to "ssh".

```
% git config core.gitproxy '"ssh" for kernel.org' 'for kernel.org$'
```

This makes sure that only the key/value pair for kernel.org is replaced.

To delete the entry for renames, do

```
% git config --unset diff.renames
```

If you want to delete an entry for a multivar (like core.gitproxy above), you have to provide a regex matching the value of exactly one line.

To query the value for a given key, do

```
% git config --get core.filemode
```

or

```
% git config core.filemode
```

or, to query a multivar:

```
% git config --get core.gitproxy "for kernel.org$"
```

If you want to know all the values for a multivar, do:

```
% git config --get-all core.gitproxy
```

If you like to live dangerously, you can replace **all** core.gitproxy by a new one with

```
% git config --replace-all core.gitproxy ssh
```

However, if you really only want to replace the line for the default proxy, i.e. the one without a "for …" postfix, do something like this:

```
% git config core.gitproxy ssh '! for '
```

To actually match only values with an exclamation mark, you have to

```
% git config section.key value '[!]'
```

To add a new proxy, without altering any of the existing ones, use

```
% git config --add core.gitproxy '"proxy-command" for example.com'
```

An example to use customized color from the configuration in your script:

```
#!/bin/sh
WS=$(git config --get-color color.diff.whitespace "blue reverse")
RESET=$(git config --get-color "" "reset")
echo "${WS}your whitespace color or blue reverse${RESET}"
```

CONFIGURATION FILE

The git configuration file contains a number of variables that affect the git command's behavior. The `.git/config` file in each repository is used to store the configuration for that repository, and `$HOME/.gitconfig` is used to store a per-user configuration as fallback values for the `.git/config` file. The file `/etc/gitconfig` can be used to store a system-wide default configuration.

The configuration variables are used by both the git plumbing and the porcelains. The variables are divided into sections, wherein the fully qualified variable name of the variable itself is the last dot-separated segment and the section name is everything before the last dot. The variable names are case-insensitive, allow only alphanumeric characters and `-`, and must start with an alphabetic character. Some variables may appear multiple times.

Syntax

The syntax is fairly flexible and permissive; whitespaces are mostly ignored. The # and ; characters begin comments to the end of line, blank lines are ignored.

The file consists of sections and variables. A section begins with the name of the section in square brackets and continues until the next section begins. Section names are not case sensitive. Only alphanumeric characters, `-` and `.` are allowed in section names. Each variable must belong to some section, which means that there must be a section header before the first setting of a variable.

Sections can be further divided into subsections. To begin a subsection put its name in double quotes, separated by space from the section name, in the section header, like in the example below:

```
        [section "subsection"]
```

Subsection names are case sensitive and can contain any characters except newline (doublequote `"` and backslash have to be escaped as `\"` and `\\`, respectively). Section headers cannot span multiple lines. Variables may belong directly to a section or to a given subsection. You can have `[section]` if you have `[section "subsection"]`, but you don't need to.

There is also a deprecated `[section.subsection]` syntax. With this syntax, the subsection name is converted to lower-case and is also compared case sensitively. These subsection names follow the same restrictions as section names.

All the other lines (and the remainder of the line after the section header) are recognized as setting variables, in the form *name = value*. If there is no equal sign on the line, the entire line is taken as *name* and the variable is recognized as boolean "true". The variable names are case-insensitive, allow only alphanumeric characters and `-`, and must start with an alphabetic character. There can be more than one value for a given variable; we say then that the variable is multivalued.

Leading and trailing whitespace in a variable value is discarded. Internal whitespace within a variable value is retained verbatim.

The values following the equals sign in variable assign are all either a string, an integer, or a boolean. Boolean values may be given as yes/no, 1/0, true/false or on/off. Case is not significant in boolean values, when converting value to the canonical form using *--bool* type specifier; *git config* will ensure that the output is "true" or "false".

String values may be entirely or partially enclosed in double quotes. You need to enclose variable values in double quotes if you want to preserve leading or trailing whitespace, or if the variable value contains comment characters (i.e. it contains # or ;). Double quote `"` and backslash `\` characters in variable values must be escaped: use `\"` for `"` and `\\` for `\`.

The following escape sequences (beside `\"` and `\\`) are recognized: `\n` for newline character (NL), `\t` for horizontal tabulation (HT, TAB) and `\b` for backspace (BS). No other char escape sequence, nor octal char sequences are valid.

Variable values ending in a `\` are continued on the next line in the customary UNIX fashion.

Some variables may require a special value format.

Includes

You can include one config file from another by setting the special `include.path` variable to the name of the file to be included. The included file is expanded immediately, as if its contents had been found at the location of the include directive. If the value of the

`include.path` variable is a relative path, the path is considered to be relative to the configuration file in which the include directive was found. The value of `include.path` is subject to tilde expansion: `~/` is expanded to the value of `$HOME`, and `~user/` to the specified user's home directory. See below for examples.

Example

```
# Core variables
[core]
        ; Don't trust file modes
        filemode = false

# Our diff algorithm
[diff]
        external = /usr/local/bin/diff-wrapper
        renames = true

[branch "devel"]
        remote = origin
        merge = refs/heads/devel

# Proxy settings
[core]
        gitProxy="ssh" for "kernel.org"
        gitProxy=default-proxy ; for the rest

[include]
        path = /path/to/foo.inc ; include by absolute path
        path = foo ; expand "foo" relative to the current file
        path = ~/foo ; expand "foo" in your $HOME directory
```

Variables

Note that this list is non-comprehensive and not necessarily complete. For command-specific variables, you will find a more detailed description in the appropriate manual page. You will find a description of non-core porcelain configuration variables in the respective porcelain documentation.

advice.*

These variables control various optional help messages designed to aid new users. All *advice.** variables default to *true*, and you can tell Git that you do not need help by setting these to *false*:

pushNonFastForward

Set this variable to *false* if you want to disable *pushNonFFCurrent*, *pushNonFFDefault*, and *pushNonFFMatching* simultaneously.

pushNonFFCurrent

Advice shown when git-push(1) fails due to a non-fast-forward update to the current branch.

pushNonFFDefault

Advice to set *push.default* to *upstream* or *current* when you ran git-push(1) and pushed *matching refs* by default (i.e. you did not provide an explicit refspec, and no *push.default* configuration was set) and it resulted in a non-fast-forward

error.

pushNonFFMatching
: Advice shown when you ran git-push(1) and pushed *matching refs* explicitly (i.e. you used *:*, or specified a refspec that isn't your current branch) and it resulted in a non-fast-forward error.

statusHints
: Show directions on how to proceed from the current state in the output of git-status(1), in the template shown when writing commit messages in git-commit(1), and in the help message shown by git-checkout(1) when switching branch.

commitBeforeMerge
: Advice shown when git-merge(1) refuses to merge to avoid overwriting local changes.

resolveConflict
: Advices shown by various commands when conflicts prevent the operation from being performed.

implicitIdentity
: Advice on how to set your identity configuration when your information is guessed from the system username and domain name.

detachedHead
: Advice shown when you used git-checkout(1) to move to the detach HEAD state, to instruct how to create a local branch after the fact.

amWorkDir
: Advice that shows the location of the patch file when git-am(1) fails to apply it.

core.fileMode
: If false, the executable bit differences between the index and the working tree are ignored; useful on broken filesystems like FAT. See git-update-index(1).

  The default is true, except git-clone(1) or git-init(1) will probe and set core.fileMode false if appropriate when the repository is created.

core.ignoreCygwinFSTricks
: This option is only used by Cygwin implementation of Git. If false, the Cygwin stat() and lstat() functions are used. This may be useful if your repository consists of a few separate directories joined in one hierarchy using Cygwin mount. If true, Git uses native Win32 API whenever it is possible and falls back to Cygwin functions only to handle symbol links. The native mode is more than twice faster than normal Cygwin l/stat() functions. True by default, unless core.filemode is true, in which case ignoreCygwinFSTricks is ignored as Cygwin's POSIX emulation is required to support core.filemode.

core.ignorecase
: If true, this option enables various workarounds to enable git to work better on filesystems that are not case sensitive, like FAT. For example, if a directory listing finds "makefile" when git expects "Makefile", git will assume it is really the same file, and continue to remember it as "Makefile".

  The default is false, except git-clone(1) or git-init(1) will probe and set core.ignorecase true if appropriate when the repository is created.

core.precomposeunicode
: This option is only used by Mac OS implementation of git. When core.precomposeunicode=true, git reverts the unicode decomposition of filenames

done by Mac OS. This is useful when sharing a repository between Mac OS and Linux or Windows. (Git for Windows 1.7.10 or higher is needed, or git under cygwin 1.7). When false, file names are handled fully transparent by git, which is backward compatible with older versions of git.

core.trustctime

If false, the ctime differences between the index and the working tree are ignored; useful when the inode change time is regularly modified by something outside Git (file system crawlers and some backup systems). See git-update-index(1). True by default.

core.quotepath

The commands that output paths (e.g. *ls-files*, *diff*), when not given the `-z` option, will quote "unusual" characters in the pathname by enclosing the pathname in a double-quote pair and with backslashes the same way strings in C source code are quoted. If this variable is set to false, the bytes higher than 0x80 are not quoted but output as verbatim. Note that double quote, backslash and control characters are always quoted without `-z` regardless of the setting of this variable.

core.eol

Sets the line ending type to use in the working directory for files that have the `text` property set. Alternatives are *lf*, *crlf* and *native*, which uses the platform's native line ending. The default value is `native`. See gitattributes(5) for more information on end-of-line conversion.

core.safecrlf

If true, makes git check if converting `CRLF` is reversible when end-of-line conversion is active. Git will verify if a command modifies a file in the work tree either directly or indirectly. For example, committing a file followed by checking out the same file should yield the original file in the work tree. If this is not the case for the current setting of `core.autocrlf`, git will reject the file. The variable can be set to "warn", in which case git will only warn about an irreversible conversion but continue the operation.

CRLF conversion bears a slight chance of corrupting data. When it is enabled, git will convert CRLF to LF during commit and LF to CRLF during checkout. A file that contains a mixture of LF and CRLF before the commit cannot be recreated by git. For text files this is the right thing to do: it corrects line endings such that we have only LF line endings in the repository. But for binary files that are accidentally classified as text the conversion can corrupt data.

If you recognize such corruption early you can easily fix it by setting the conversion type explicitly in .gitattributes. Right after committing you still have the original file in your work tree and this file is not yet corrupted. You can explicitly tell git that this file is binary and git will handle the file appropriately.

Unfortunately, the desired effect of cleaning up text files with mixed line endings and the undesired effect of corrupting binary files cannot be distinguished. In both cases CRLFs are removed in an irreversible way. For text files this is the right thing to do because CRLFs are line endings, while for binary files converting CRLFs corrupts data.

Note, this safety check does not mean that a checkout will generate a file identical to the original file for a different setting of `core.eol` and `core.autocrlf`, but only for the current one. For example, a text file with `LF` would be accepted with `core.eol=lf` and could later be checked out with `core.eol=crlf`, in which case the resulting file would contain `CRLF`, although the original file contained `LF`. However, in both work trees the line endings would be consistent, that is either all `LF` or all `CRLF`, but never mixed. A file with mixed line endings would be reported by the `core.safecrlf` mechanism.

core.autocrlf

Setting this variable to "true" is almost the same as setting the `text` attribute to "auto" on all files except that text files are not guaranteed to be normalized: files that contain `CRLF` in the repository will not be touched. Use this setting if you want to have `CRLF` line endings in your working directory even though the repository does not have normalized line endings. This variable can be set to *input*, in which case no output conversion is performed.

core.symlinks

If false, symbolic links are checked out as small plain files that contain the link text. git-update-index(1) and git-add(1) will not change the recorded type to regular file. Useful on filesystems like FAT that do not support symbolic links.

The default is true, except git-clone(1) or git-init(1) will probe and set core.symlinks false if appropriate when the repository is created.

core.gitProxy

A "proxy command" to execute (as *command host port*) instead of establishing direct connection to the remote server when using the git protocol for fetching. If the variable value is in the "COMMAND for DOMAIN" format, the command is applied only on hostnames ending with the specified domain string. This variable may be set multiple times and is matched in the given order; the first match wins.

Can be overridden by the *GIT_PROXY_COMMAND* environment variable (which always applies universally, without the special "for" handling).

The special string `none` can be used as the proxy command to specify that no proxy be used for a given domain pattern. This is useful for excluding servers inside a firewall from proxy use, while defaulting to a common proxy for external domains.

core.ignoreStat

If true, commands which modify both the working tree and the index will mark the updated paths with the "assume unchanged" bit in the index. These marked files are then assumed to stay unchanged in the working tree, until you mark them otherwise manually - Git will not detect the file changes by lstat() calls. This is useful on systems where those are very slow, such as Microsoft Windows. See git-update-index(1). False by default.

core.preferSymlinkRefs

Instead of the default "symref" format for HEAD and other symbolic reference files, use symbolic links. This is sometimes needed to work with old scripts that expect HEAD to be a symbolic link.

core.bare

If true this repository is assumed to be *bare* and has no working directory associated with it. If this is the case a number of commands that require a working directory will be disabled, such as git-add(1) or git-merge(1).

This setting is automatically guessed by git-clone(1) or git-init(1) when the repository was created. By default a repository that ends in "/.git" is assumed to be not bare (bare = false), while all other repositories are assumed to be bare (bare = true).

core.worktree

Set the path to the root of the working tree. This can be overridden by the GIT_WORK_TREE environment variable and the *--work-tree* command line option. The value can be an absolute path or relative to the path to the .git directory, which is either specified by --git-dir or GIT_DIR, or automatically discovered. If --git-dir or GIT_DIR is specified but none of --work-tree, GIT_WORK_TREE and core.worktree is specified, the current working directory is regarded as the top level of your working tree.

Note that this variable is honored even when set in a configuration file in a ".git" subdirectory of a directory and its value differs from the latter directory (e.g. "/path /to/.git/config" has core.worktree set to "/different/path"), which is most likely a misconfiguration. Running git commands in the "/path/to" directory will still use "/different/path" as the root of the work tree and can cause confusion unless you know what you are doing (e.g. you are creating a read-only snapshot of the same index to a location different from the repository's usual working tree).

core.logAllRefUpdates

Enable the reflog. Updates to a ref <ref> is logged to the file "$GIT_DIR/logs/<ref>", by appending the new and old SHA1, the date/time and the reason of the update, but only when the file exists. If this configuration variable is set to true, missing "$GIT_DIR/logs/<ref>" file is automatically created for branch heads (i.e. under refs/heads/), remote refs (i.e. under refs/remotes/), note refs (i.e. under refs/notes/), and the symbolic ref HEAD.

This information can be used to determine what commit was the tip of a branch "2 days ago".

This value is true by default in a repository that has a working directory associated with it, and false by default in a bare repository.

core.repositoryFormatVersion

Internal variable identifying the repository format and layout version.

core.sharedRepository

When *group* (or *true*), the repository is made shareable between several users in a group (making sure all the files and objects are group-writable). When *all* (or *world* or *everybody*), the repository will be readable by all users, additionally to being group-shareable. When *umask* (or *false*), git will use permissions reported by umask(2). When *0xxx*, where *0xxx* is an octal number, files in the repository will have this mode value. *0xxx* will override user's umask value (whereas the other options will only override requested parts of the user's umask value). Examples: *0660* will make the repo read/write-able for the owner and group, but inaccessible to others (equivalent to *group* unless umask is e.g. *0022*). *0640* is a repository that is group-readable but not group-writable. See git-init(1). False by default.

core.warnAmbiguousRefs

If true, git will warn you if the ref name you passed it is ambiguous and might match multiple refs in the .git/refs/ tree. True by default.

core.compression

An integer -1..9, indicating a default compression level. -1 is the zlib default. 0 means no compression, and 1..9 are various speed/size tradeoffs, 9 being slowest. If set, this provides a default to other compression variables, such as *core.loosecompression* and *pack.compression*.

core.loosecompression

An integer -1..9, indicating the compression level for objects that are not in a pack file. -1 is the zlib default. 0 means no compression, and 1..9 are various speed/size tradeoffs, 9 being slowest. If not set, defaults to core.compression. If that is not set, defaults to 1 (best speed).

core.packedGitWindowSize

Number of bytes of a pack file to map into memory in a single mapping operation. Larger window sizes may allow your system to process a smaller number of large pack files more quickly. Smaller window sizes will negatively affect performance due to increased calls to the operating system's memory manager, but may improve performance when accessing a large number of large pack files.

Default is 1 MiB if NO_MMAP was set at compile time, otherwise 32 MiB on 32 bit platforms and 1 GiB on 64 bit platforms. This should be reasonable for all users/operating systems. You probably do not need to adjust this value.

Common unit suffixes of *k*, *m*, or *g* are supported.

core.packedGitLimit

Maximum number of bytes to map simultaneously into memory from pack files. If Git needs to access more than this many bytes at once to complete an operation it will unmap existing regions to reclaim virtual address space within the process.

Default is 256 MiB on 32 bit platforms and 8 GiB on 64 bit platforms. This should be reasonable for all users/operating systems, except on the largest projects. You probably do not need to adjust this value.

Common unit suffixes of *k*, *m*, or *g* are supported.

core.deltaBaseCacheLimit

Maximum number of bytes to reserve for caching base objects that may be referenced by multiple deltified objects. By storing the entire decompressed base objects in a cache Git is able to avoid unpacking and decompressing frequently used base objects multiple times.

Default is 16 MiB on all platforms. This should be reasonable for all users/operating systems, except on the largest projects. You probably do not need to adjust this value.

Common unit suffixes of *k*, *m*, or *g* are supported.

core.bigFileThreshold

Files larger than this size are stored deflated, without attempting delta compression. Storing large files without delta compression avoids excessive memory usage, at the slight expense of increased disk usage.

Default is 512 MiB on all platforms. This should be reasonable for most projects as source code and other text files can still be delta compressed, but larger binary media files won’t be.

Common unit suffixes of *k*, *m*, or *g* are supported.

core.excludesfile

In addition to *.gitignore* (per-directory) and *.git/info/exclude*, git looks into this file for patterns of files which are not meant to be tracked. "`~/`" is expanded to the value of `$HOME` and "`~user/`" to the specified user’s home directory. Its default value is $XDG_CONFIG_HOME/git/ignore. If $XDG_CONFIG_HOME is either not set or empty, $HOME/.config/git/ignore is used instead. See gitignore(5).

core.askpass

Some commands (e.g. svn and http interfaces) that interactively ask for a password can be told to use an external program given via the value of this variable. Can be overridden by the *GIT_ASKPASS* environment variable. If not set, fall back to the value of the *SSH_ASKPASS* environment variable or, failing that, a simple password prompt. The external program shall be given a suitable prompt as command line argument and write the password on its STDOUT.

core.attributesfile

In addition to *.gitattributes* (per-directory) and *.git/info/attributes*, git looks into this file for attributes (see gitattributes(5)). Path expansions are made the same way as for `core.excludesfile`. Its default value is $XDG_CONFIG_HOME/git/attributes. If $XDG_CONFIG_HOME is either not set or empty, $HOME/.config/git/attributes is used instead.

core.editor

Commands such as `commit` and `tag` that lets you edit messages by launching an editor uses the value of this variable when it is set, and the environment variable `GIT_EDITOR` is not set. See git-var(1).

sequence.editor
: Text editor used by `git rebase -i` for editing the rebase insn file. The value is meant to be interpreted by the shell when it is used. It can be overridden by the `GIT_SEQUENCE_EDITOR` environment variable. When not configured the default commit message editor is used instead.

core.pager
: The command that git will use to paginate output. Can be overridden with the `GIT_PAGER` environment variable. Note that git sets the `LESS` environment variable to `FRSX` if it is unset when it runs the pager. One can change these settings by setting the `LESS` variable to some other value. Alternately, these settings can be overridden on a project or global basis by setting the `core.pager` option. Setting `core.pager` has no effect on the `LESS` environment variable behaviour above, so if you want to override git's default settings this way, you need to be explicit. For example, to disable the S option in a backward compatible manner, set `core.pager` to `less -+S`. This will be passed to the shell by git, which will translate the final command to `LESS=FRSX less -+S`.

core.whitespace
: A comma separated list of common whitespace problems to notice. *git diff* will use `color.diff.whitespace` to highlight them, and *git apply --whitespace=error* will consider them as errors. You can prefix `-` to disable any of them (e.g. `-trailing-space`):

  - `blank-at-eol` treats trailing whitespaces at the end of the line as an error (enabled by default).
  - `space-before-tab` treats a space character that appears immediately before a tab character in the initial indent part of the line as an error (enabled by default).
  - `indent-with-non-tab` treats a line that is indented with space characters instead of the equivalent tabs as an error (not enabled by default).
  - `tab-in-indent` treats a tab character in the initial indent part of the line as an error (not enabled by default).
  - `blank-at-eof` treats blank lines added at the end of file as an error (enabled by default).
  - `trailing-space` is a short-hand to cover both `blank-at-eol` and `blank-at-eof`.
  - `cr-at-eol` treats a carriage-return at the end of line as part of the line terminator, i.e. with it, `trailing-space` does not trigger if the character before such a carriage-return is not a whitespace (not enabled by default).
  - `tabwidth=<n>` tells how many character positions a tab occupies; this is relevant for `indent-with-non-tab` and when git fixes `tab-in-indent` errors. The default tab width is 8. Allowed values are 1 to 63.

core.fsyncobjectfiles
: This boolean will enable *fsync()* when writing object files.

  This is a total waste of time and effort on a filesystem that orders data writes properly, but can be useful for filesystems that do not use journalling (traditional UNIX filesystems) or that only journal metadata and not file contents (OS X's HFS+, or Linux ext3 with "data=writeback").

core.preloadindex
: Enable parallel index preload for operations like *git diff*

  This can speed up operations like *git diff* and *git status* especially on filesystems like NFS that have weak caching semantics and thus relatively high IO latencies. With this set to *true*, git will do the index comparison to the filesystem data in parallel, allowing overlapping IO's.

core.createObject
: You can set this to *link*, in which case a hardlink followed by a delete of the source are used to make sure that object creation will not overwrite existing objects.

  On some file system/operating system combinations, this is unreliable. Set this config setting to *rename* there; However, This will remove the check that makes sure that existing object files will not get overwritten.

core.notesRef
: When showing commit messages, also show notes which are stored in the given ref. The ref must be fully qualified. If the given ref does not exist, it is not an error but means that no notes should be printed.

  This setting defaults to "refs/notes/commits", and it can be overridden by the *GIT_NOTES_REF* environment variable. See git-notes(1).

core.sparseCheckout
: Enable "sparse checkout" feature. See section "Sparse checkout" in git-read-tree(1) for more information.

core.abbrev
: Set the length object names are abbreviated to. If unspecified, many commands abbreviate to 7 hexdigits, which may not be enough for abbreviated object names to stay unique for sufficiently long time.

add.ignore-errors

add.ignoreErrors
: Tells *git add* to continue adding files when some files cannot be added due to indexing errors. Equivalent to the *--ignore-errors* option of git-add(1). Older versions of git accept only `add.ignore-errors`, which does not follow the usual naming convention for configuration variables. Newer versions of git honor `add.ignoreErrors` as well.

alias.*
: Command aliases for the git(1) command wrapper - e.g. after defining "alias.last = cat-file commit HEAD", the invocation "git last" is equivalent to "git cat-file commit HEAD". To avoid confusion and troubles with script usage, aliases that hide existing git commands are ignored. Arguments are split by spaces, the usual shell quoting and escaping is supported. quote pair and a backslash can be used to quote them.

  If the alias expansion is prefixed with an exclamation point, it will be treated as a shell command. For example, defining "alias.new = !gitk --all --not ORIG_HEAD", the invocation "git new" is equivalent to running the shell command "gitk --all --not ORIG_HEAD". Note that shell commands will be executed from the top-level directory of a repository, which may not necessarily be the current directory. *GIT_PREFIX* is set as returned by running *git rev-parse --show-prefix* from the original current directory. See git-rev-parse(1).

am.keepcr
: If true, git-am will call git-mailsplit for patches in mbox format with parameter *--keep-cr*. In this case git-mailsplit will not remove `\r` from lines ending with `\r\n`. Can be overridden by giving *--no-keep-cr* from the command line. See git-am(1), git-mailsplit(1).

apply.ignorewhitespace

When set to *change*, tells *git apply* to ignore changes in whitespace, in the same way as the *--ignore-space-change* option. When set to one of: no, none, never, false tells *git apply* to respect all whitespace differences. See git-apply(1).

apply.whitespace

Tells *git apply* how to handle whitespaces, in the same way as the *--whitespace* option. See git-apply(1).

branch.autosetupmerge

Tells *git branch* and *git checkout* to set up new branches so that git-pull(1) will appropriately merge from the starting point branch. Note that even if this option is not set, this behavior can be chosen per-branch using the `--track` and `--no-track` options. The valid settings are: `false` — no automatic setup is done; `true` — automatic setup is done when the starting point is a remote-tracking branch; `always` — automatic setup is done when the starting point is either a local branch or remote-tracking branch. This option defaults to true.

branch.autosetuprebase

When a new branch is created with *git branch* or *git checkout* that tracks another branch, this variable tells git to set up pull to rebase instead of merge (see "branch.<name>.rebase"). When `never`, rebase is never automatically set to true. When `local`, rebase is set to true for tracked branches of other local branches. When `remote`, rebase is set to true for tracked branches of remote-tracking branches. When `always`, rebase will be set to true for all tracking branches. See "branch.autosetupmerge" for details on how to set up a branch to track another branch. This option defaults to never.

branch.<name>.remote

When in branch <name>, it tells *git fetch* and *git push* which remote to fetch from/push to. It defaults to `origin` if no remote is configured. `origin` is also used if you are not on any branch.

branch.<name>.merge

Defines, together with branch.<name>.remote, the upstream branch for the given branch. It tells *git fetch*/*git pull*/*git rebase* which branch to merge and can also affect *git push* (see push.default). When in branch <name>, it tells *git fetch* the default refspec to be marked for merging in FETCH_HEAD. The value is handled like the remote part of a refspec, and must match a ref which is fetched from the remote given by "branch.<name>.remote". The merge information is used by *git pull* (which at first calls *git fetch*) to lookup the default branch for merging. Without this option, *git pull* defaults to merge the first refspec fetched. Specify multiple values to get an octopus merge. If you wish to setup *git pull* so that it merges into <name> from another branch in the local repository, you can point branch.<name>.merge to the desired branch, and use the special setting `.` (a period) for branch.<name>.remote.

branch.<name>.mergeoptions

Sets default options for merging into branch <name>. The syntax and supported options are the same as those of git-merge(1), but option values containing whitespace characters are currently not supported.

branch.<name>.rebase

When true, rebase the branch <name> on top of the fetched branch, instead of merging the default branch from the default remote when "git pull" is run. See "pull.rebase" for doing this in a non branch-specific manner.

NOTE: this is a possibly dangerous operation; do **not** use it unless you understand the implications (see git-rebase(1) for details).

browser.<tool>.cmd
: Specify the command to invoke the specified browser. The specified command is evaluated in shell with the URLs passed as arguments. (See git-web--browse(1).)

browser.<tool>.path
: Override the path for the given tool that may be used to browse HTML help (see *-w* option in git-help(1)) or a working repository in gitweb (see git-instaweb(1)).

clean.requireForce
: A boolean to make git-clean do nothing unless given -f or -n. Defaults to true.

color.branch
: A boolean to enable/disable color in the output of git-branch(1). May be set to `always`, `false` (or `never`) or `auto` (or `true`), in which case colors are used only when the output is to a terminal. Defaults to false.

color.branch.<slot>
: Use customized color for branch coloration. `<slot>` is one of `current` (the current branch), `local` (a local branch), `remote` (a remote-tracking branch in refs/remotes/), `plain` (other refs).

  The value for these configuration variables is a list of colors (at most two) and attributes (at most one), separated by spaces. The colors accepted are `normal`, `black`, `red`, `green`, `yellow`, `blue`, `magenta`, `cyan` and `white`; the attributes are `bold`, `dim`, `ul`, `blink` and `reverse`. The first color given is the foreground; the second is the background. The position of the attribute, if any, doesn't matter.

color.diff
: Whether to use ANSI escape sequences to add color to patches. If this is set to `always`, git-diff(1), git-log(1), and git-show(1) will use color for all patches. If it is set to `true` or `auto`, those commands will only use color when output is to the terminal. Defaults to false.

  This does not affect git-format-patch(1) nor the *git-diff-** plumbing commands. Can be overridden on the command line with the `--color[=<when>]` option.

color.diff.<slot>
: Use customized color for diff colorization. `<slot>` specifies which part of the patch to use the specified color, and is one of `plain` (context text), `meta` (metainformation), `frag` (hunk header), *func* (function in hunk header), `old` (removed lines), `new` (added lines), `commit` (commit headers), or `whitespace` (highlighting whitespace errors). The values of these variables may be specified as in color.branch.<slot>.

color.decorate.<slot>
: Use customized color for *git log --decorate* output. `<slot>` is one of `branch`, `remoteBranch`, `tag`, `stash` or `HEAD` for local branches, remote-tracking branches, tags, stash and HEAD, respectively.

color.grep
: When set to `always`, always highlight matches. When `false` (or `never`), never. When set to `true` or `auto`, use color only when the output is written to the terminal. Defaults to `false`.

color.grep.<slot>
: Use customized color for grep colorization. `<slot>` specifies which part of the line to use the specified color, and is one of

  `context`
  : non-matching text in context lines (when using `-A`, `-B`, or `-C`)

  `filename`

filename prefix (when not using `-h`)

`function`
: function name lines (when using `-p`)

`linenumber`
: line number prefix (when using `-n`)

`match`
: matching text

`selected`
: non-matching text in selected lines

`separator`
: separators between fields on a line (`:`, `-`, and `=`) and between hunks (`--`)

The values of these variables may be specified as in color.branch.<slot>.

color.interactive
: When set to `always`, always use colors for interactive prompts and displays (such as those used by "git-add --interactive"). When false (or `never`), never. When set to `true` or `auto`, use colors only when the output is to the terminal. Defaults to false.

color.interactive.<slot>
: Use customized color for *git add --interactive* output. `<slot>` may be `prompt`, `header`, `help` or `error`, for four distinct types of normal output from interactive commands. The values of these variables may be specified as in color.branch.<slot>.

color.pager
: A boolean to enable/disable colored output when the pager is in use (default is true).

color.showbranch
: A boolean to enable/disable color in the output of git-show-branch(1). May be set to `always`, `false` (or `never`) or `auto` (or `true`), in which case colors are used only when the output is to a terminal. Defaults to false.

color.status
: A boolean to enable/disable color in the output of git-status(1). May be set to `always`, `false` (or `never`) or `auto` (or `true`), in which case colors are used only when the output is to a terminal. Defaults to false.

color.status.<slot>
: Use customized color for status colorization. `<slot>` is one of `header` (the header text of the status message), `added` or `updated` (files which are added but not committed), `changed` (files which are changed but not added in the index), `untracked` (files which are not tracked by git), `branch` (the current branch), or `nobranch` (the color the *no branch* warning is shown in, defaulting to red). The values of these variables may be specified as in color.branch.<slot>.

color.ui
: This variable determines the default value for variables such as `color.diff` and `color.grep` that control the use of color per command family. Its scope will expand as more commands learn configuration to set a default for the `--color` option. Set it to `always` if you want all output not intended for machine consumption to use color, to `true` or `auto` if you want such output to use color when written to the terminal, or to `false` or `never` if you prefer git commands not to use color unless enabled explicitly with some other configuration or the `--color` option.

column.ui
: Specify whether supported commands should output in columns. This variable consists

of a list of tokens separated by spaces or commas:

`always`
: always show in columns

`never`
: never show in columns

`auto`
: show in columns if the output is to the terminal

`column`
: fill columns before rows (default)

`row`
: fill rows before columns

`plain`
: show in one column

`dense`
: make unequal size columns to utilize more space

`nodense`
: make equal size columns

This option defaults to *never*.

column.branch
: Specify whether to output branch listing in `git branch` in columns. See `column.ui` for details.

column.status
: Specify whether to output untracked files in `git status` in columns. See `column.ui` for details.

column.tag
: Specify whether to output tag listing in `git tag` in columns. See `column.ui` for details.

commit.status
: A boolean to enable/disable inclusion of status information in the commit message template when using an editor to prepare the commit message. Defaults to true.

commit.template
: Specify a file to use as the template for new commit messages. "`~/`" is expanded to the value of `$HOME` and "`~user/`" to the specified user's home directory.

credential.helper
: Specify an external helper to be called when a username or password credential is needed; the helper may consult external storage to avoid prompting the user for the credentials. See gitcredentials(7) for details.

credential.useHttpPath
: When acquiring credentials, consider the "path" component of an http or https URL to be important. Defaults to false. See gitcredentials(7) for more information.

credential.username
: If no username is set for a network authentication, use this username by default. See credential.<context>.* below, and gitcredentials(7).

credential.<url>.*
: Any of the credential.* options above can be applied selectively to some credentials.

For example "credential.https://example.com.username" would set the default username only for https connections to example.com. See gitcredentials(7) for details on how URLs are matched.

diff.autorefreshindex

When using *git diff* to compare with work tree files, do not consider stat-only change as changed. Instead, silently run `git update-index --refresh` to update the cached stat information for paths whose contents in the work tree match the contents in the index. This option defaults to true. Note that this affects only *git diff* Porcelain, and not lower level *diff* commands such as *git diff-files*.

diff.dirstat

A comma separated list of `--dirstat` parameters specifying the default behavior of the `--dirstat` option to git-diff(1)` and friends. The defaults can be overridden on the command line (using `--dirstat=<param1,param2,...>`). The fallback defaults (when not changed by `diff.dirstat`) are `changes,noncumulative,3`. The following parameters are available:

`changes`

Compute the dirstat numbers by counting the lines that have been removed from the source, or added to the destination. This ignores the amount of pure code movements within a file. In other words, rearranging lines in a file is not counted as much as other changes. This is the default behavior when no parameter is given.

`lines`

Compute the dirstat numbers by doing the regular line-based diff analysis, and summing the removed/added line counts. (For binary files, count 64-byte chunks instead, since binary files have no natural concept of lines). This is a more expensive `--dirstat` behavior than the `changes` behavior, but it does count rearranged lines within a file as much as other changes. The resulting output is consistent with what you get from the other `--*stat` options.

`files`

Compute the dirstat numbers by counting the number of files changed. Each changed file counts equally in the dirstat analysis. This is the computationally cheapest `--dirstat` behavior, since it does not have to look at the file contents at all.

`cumulative`

Count changes in a child directory for the parent directory as well. Note that when using `cumulative`, the sum of the percentages reported may exceed 100%. The default (non-cumulative) behavior can be specified with the `noncumulative` parameter.

<limit>

An integer parameter specifies a cut-off percent (3% by default). Directories contributing less than this percentage of the changes are not shown in the output.

Example: The following will count changed files, while ignoring directories with less than 10% of the total amount of changed files, and accumulating child directory counts in the parent directories: `files,10,cumulative`.

diff.statGraphWidth

Limit the width of the graph part in --stat output. If set, applies to all commands generating --stat output except format-patch.

diff.context

Generate diffs with <n> lines of context instead of the default of 3. This value is overridden by the -U option.

diff.external

If this config variable is set, diff generation is not performed using the internal diff machinery, but using the given command. Can be overridden with the 'GIT_EXTERNAL_DIFF' environment variable. The command is called with parameters as described under "git Diffs" in git(1). Note: if you want to use an external diff program only on a subset of your files, you might want to use gitattributes(5) instead.

diff.ignoreSubmodules

Sets the default value of --ignore-submodules. Note that this affects only *git diff* Porcelain, and not lower level *diff* commands such as *git diff-files*. *git checkout* also honors this setting when reporting uncommitted changes.

diff.mnemonicprefix

If set, *git diff* uses a prefix pair that is different from the standard "a/" and "b/" depending on what is being compared. When this configuration is in effect, reverse diff output also swaps the order of the prefixes:

`git diff`
: compares the (i)ndex and the (w)ork tree;

`git diff HEAD`
: compares a (c)ommit and the (w)ork tree;

`git diff --cached`
: compares a (c)ommit and the (i)ndex;

`git diff HEAD:file1 file2`
: compares an (o)bject and a (w)ork tree entity;

`git diff --no-index a b`
: compares two non-git things (1) and (2).

diff.noprefix

If set, *git diff* does not show any source or destination prefix.

diff.renameLimit

The number of files to consider when performing the copy/rename detection; equivalent to the *git diff* option *-l*.

diff.renames

Tells git to detect renames. If set to any boolean value, it will enable basic rename detection. If set to "copies" or "copy", it will detect copies, as well.

diff.suppressBlankEmpty

A boolean to inhibit the standard behavior of printing a space before each empty output line. Defaults to false.

diff.submodule

Specify the format in which differences in submodules are shown. The "log" format lists the commits in the range like git-submodule(1) `summary` does. The "short" format format just shows the names of the commits at the beginning and end of the range. Defaults to short.

diff.wordRegex

A POSIX Extended Regular Expression used to determine what is a "word" when performing word-by-word difference calculations. Character sequences that match the regular expression are "words", all other characters are **ignorable** whitespace.

diff.<driver>.command
: The custom diff driver command. See gitattributes(5) for details.

diff.<driver>.xfuncname
: The regular expression that the diff driver should use to recognize the hunk header. A built-in pattern may also be used. See gitattributes(5) for details.

diff.<driver>.binary
: Set this option to true to make the diff driver treat files as binary. See gitattributes(5) for details.

diff.<driver>.textconv
: The command that the diff driver should call to generate the text-converted version of a file. The result of the conversion is used to generate a human-readable diff. See gitattributes(5) for details.

diff.<driver>.wordregex
: The regular expression that the diff driver should use to split words in a line. See gitattributes(5) for details.

diff.<driver>.cachetextconv
: Set this option to true to make the diff driver cache the text conversion outputs. See gitattributes(5) for details.

diff.tool
: The diff tool to be used by git-difftool(1). This option overrides `merge.tool`, and has the same valid built-in values as `merge.tool` minus "tortoisemerge" and plus "kompare". Any other value is treated as a custom diff tool, and there must be a corresponding `difftool.<tool>.cmd` option.

difftool.<tool>.path
: Override the path for the given tool. This is useful in case your tool is not in the PATH.

difftool.<tool>.cmd
: Specify the command to invoke the specified diff tool. The specified command is evaluated in shell with the following variables available: *LOCAL* is set to the name of the temporary file containing the contents of the diff pre-image and *REMOTE* is set to the name of the temporary file containing the contents of the diff post-image.

difftool.prompt
: Prompt before each invocation of the diff tool.

fetch.recurseSubmodules
: This option can be either set to a boolean value or to *on-demand*. Setting it to a boolean changes the behavior of fetch and pull to unconditionally recurse into submodules when set to true or to not recurse at all when set to false. When set to *on-demand* (the default value), fetch and pull will only recurse into a populated submodule when its superproject retrieves a commit that updates the submodule’s reference.

fetch.fsckObjects
: If it is set to true, git-fetch-pack will check all fetched objects. It will abort in the case of a malformed object or a broken link. The result of an abort are only dangling objects. Defaults to false. If not set, the value of `transfer.fsckObjects` is used instead.

fetch.unpackLimit
: If the number of objects fetched over the git native transfer is below this limit, then the objects will be unpacked into loose object files. However if the number of received objects equals or exceeds this limit then the received pack will be stored as a pack,

after adding any missing delta bases. Storing the pack from a push can make the push operation complete faster, especially on slow filesystems. If not set, the value of `transfer.unpackLimit` is used instead.

format.attach
: Enable multipart/mixed attachments as the default for *format-patch*. The value can also be a double quoted string which will enable attachments as the default and set the value as the boundary. See the --attach option in git-format-patch(1).

format.numbered
: A boolean which can enable or disable sequence numbers in patch subjects. It defaults to "auto" which enables it only if there is more than one patch. It can be enabled or disabled for all messages by setting it to "true" or "false". See --numbered option in git-format-patch(1).

format.headers
: Additional email headers to include in a patch to be submitted by mail. See git-format-patch(1).

format.to

format.cc
: Additional recipients to include in a patch to be submitted by mail. See the --to and --cc options in git-format-patch(1).

format.subjectprefix
: The default for format-patch is to output files with the *[PATCH]* subject prefix. Use this variable to change that prefix.

format.signature
: The default for format-patch is to output a signature containing the git version number. Use this variable to change that default. Set this variable to the empty string ("") to suppress signature generation.

format.suffix
: The default for format-patch is to output files with the suffix `.patch`. Use this variable to change that suffix (make sure to include the dot if you want it).

format.pretty
: The default pretty format for log/show/whatchanged command, See git-log(1), git-show(1), git-whatchanged(1).

format.thread
: The default threading style for *git format-patch*. Can be a boolean value, or `shallow` or `deep`. `shallow` threading makes every mail a reply to the head of the series, where the head is chosen from the cover letter, the `--in-reply-to`, and the first patch mail, in this order. `deep` threading makes every mail a reply to the previous one. A true boolean value is the same as `shallow`, and a false value disables threading.

format.signoff
: A boolean value which lets you enable the `-s/--signoff` option of format-patch by default. **Note:** Adding the Signed-off-by: line to a patch should be a conscious act and means that you certify you have the rights to submit this work under the same open source license. Please see the *SubmittingPatches* document for further discussion.

filter.<driver>.clean
: The command which is used to convert the content of a worktree file to a blob upon checkin. See gitattributes(5) for details.

filter.<driver>.smudge
: The command which is used to convert the content of a blob object to a worktree file

upon checkout. See gitattributes(5) for details.

gc.aggressiveWindow
: The window size parameter used in the delta compression algorithm used by *git gc --aggressive*. This defaults to 250.

gc.auto
: When there are approximately more than this many loose objects in the repository, `git gc --auto` will pack them. Some Porcelain commands use this command to perform a light-weight garbage collection from time to time. The default value is 6700. Setting this to 0 disables it.

gc.autopacklimit
: When there are more than this many packs that are not marked with `*.keep` file in the repository, `git gc --auto` consolidates them into one larger pack. The default value is 50. Setting this to 0 disables it.

gc.packrefs
: Running `git pack-refs` in a repository renders it unclonable by Git versions prior to 1.5.1.2 over dumb transports such as HTTP. This variable determines whether *git gc* runs `git pack-refs`. This can be set to `notbare` to enable it within all non-bare repos or it can be set to a boolean value. The default is `true`.

gc.pruneexpire
: When *git gc* is run, it will call *prune --expire 2.weeks.ago*. Override the grace period with this config variable. The value "now" may be used to disable this grace period and always prune unreachable objects immediately.

gc.reflogexpire

gc.<pattern>.reflogexpire
: *git reflog expire* removes reflog entries older than this time; defaults to 90 days. With "<pattern>" (e.g. "refs/stash") in the middle the setting applies only to the refs that match the <pattern>.

gc.reflogexpireunreachable

gc.<ref>.reflogexpireunreachable
: *git reflog expire* removes reflog entries older than this time and are not reachable from the current tip; defaults to 30 days. With "<pattern>" (e.g. "refs/stash") in the middle, the setting applies only to the refs that match the <pattern>.

gc.rerereresolved
: Records of conflicted merge you resolved earlier are kept for this many days when *git rerere gc* is run. The default is 60 days. See git-rerere(1).

gc.rerereunresolved
: Records of conflicted merge you have not resolved are kept for this many days when *git rerere gc* is run. The default is 15 days. See git-rerere(1).

gitcvs.commitmsgannotation
: Append this string to each commit message. Set to empty string to disable this feature. Defaults to "via git-CVS emulator".

gitcvs.enabled
: Whether the CVS server interface is enabled for this repository. See git-cvsserver(1).

gitcvs.logfile
: Path to a log file where the CVS server interface well… logs various stuff. See git-cvsserver(1).

gitcvs.usecrlfattr

If true, the server will look up the end-of-line conversion attributes for files to determine the *-k* modes to use. If the attributes force git to treat a file as text, the *-k* mode will be left blank so CVS clients will treat it as text. If they suppress text conversion, the file will be set with *-kb* mode, which suppresses any newline munging the client might otherwise do. If the attributes do not allow the file type to be determined, then *gitcvs.allbinary* is used. See gitattributes(5).

gitcvs.allbinary
: This is used if *gitcvs.usecrlfattr* does not resolve the correct *-kb* mode to use. If true, all unresolved files are sent to the client in mode *-kb*. This causes the client to treat them as binary files, which suppresses any newline munging it otherwise might do. Alternatively, if it is set to "guess", then the contents of the file are examined to decide if it is binary, similar to *core.autocrlf*.

gitcvs.dbname
: Database used by git-cvsserver to cache revision information derived from the git repository. The exact meaning depends on the used database driver, for SQLite (which is the default driver) this is a filename. Supports variable substitution (see git-cvsserver(1) for details). May not contain semicolons (`;`). Default: *%Ggitcvs.%m.sqlite*

gitcvs.dbdriver
: Used Perl DBI driver. You can specify any available driver for this here, but it might not work. git-cvsserver is tested with *DBD::SQLite*, reported to work with *DBD::Pg*, and reported **not** to work with *DBD::mysql*. Experimental feature. May not contain double colons (`:`). Default: *SQLite*. See git-cvsserver(1).

gitcvs.dbuser, gitcvs.dbpass
: Database user and password. Only useful if setting *gitcvs.dbdriver*, since SQLite has no concept of database users and/or passwords. *gitcvs.dbuser* supports variable substitution (see git-cvsserver(1) for details).

gitcvs.dbTableNamePrefix
: Database table name prefix. Prepended to the names of any database tables used, allowing a single database to be used for several repositories. Supports variable substitution (see git-cvsserver(1) for details). Any non-alphabetic characters will be replaced with underscores.

All gitcvs variables except for *gitcvs.usecrlfattr* and *gitcvs.allbinary* can also be specified as *gitcvs.<access_method>.<varname>* (where *access_method* is one of "ext" and "pserver") to make them apply only for the given access method.

gitweb.category

gitweb.description

gitweb.owner

gitweb.url
: See gitweb(1) for description.

gitweb.avatar

gitweb.blame

gitweb.grep

gitweb.highlight

gitweb.patches

gitweb.pickaxe

gitweb.remote_heads

gitweb.showsizes

gitweb.snapshot
: See gitweb.conf(5) for description.

grep.lineNumber
: If set to true, enable *-n* option by default.

grep.patternType
: Set the default matching behavior. Using a value of *basic*, *extended*, *fixed*, or *perl* will enable the *--basic-regexp*, *--extended-regexp*, *--fixed-strings*, or *--perl-regexp* option accordingly, while the value *default* will return to the default matching behavior.

grep.extendedRegexp
: If set to true, enable *--extended-regexp* option by default. This option is ignored when the *grep.patternType* option is set to a value other than *default*.

gpg.program
: Use this custom program instead of "gpg" found on $PATH when making or verifying a PGP signature. The program must support the same command line interface as GPG, namely, to verify a detached signature, "gpg --verify $file - <$signature" is run, and the program is expected to signal a good signature by exiting with code 0, and to generate an ascii-armored detached signature, the standard input of "gpg -bsau $key" is fed with the contents to be signed, and the program is expected to send the result to its standard output.

gui.commitmsgwidth
: Defines how wide the commit message window is in the git-gui(1). "75" is the default.

gui.diffcontext
: Specifies how many context lines should be used in calls to diff made by the git-gui(1). The default is "5".

gui.encoding
: Specifies the default encoding to use for displaying of file contents in git-gui(1) and gitk(1). It can be overridden by setting the *encoding* attribute for relevant files (see gitattributes(5)). If this option is not set, the tools default to the locale encoding.

gui.matchtrackingbranch
: Determines if new branches created with git-gui(1) should default to tracking remote branches with matching names or not. Default: "false".

gui.newbranchtemplate
: Is used as suggested name when creating new branches using the git-gui(1).

gui.pruneduringfetch
: "true" if git-gui(1) should prune remote-tracking branches when performing a fetch. The default value is "false".

gui.trustmtime
: Determines if git-gui(1) should trust the file modification timestamp or not. By default the timestamps are not trusted.

gui.spellingdictionary
: Specifies the dictionary used for spell checking commit messages in the git-gui(1). When set to "none" spell checking is turned off.

gui.fastcopyblame
: If true, *git gui blame* uses `-C` instead of `-C -C` for original location detection. It makes blame significantly faster on huge repositories at the expense of less thorough copy

detection.

gui.copyblamethreshold
: Specifies the threshold to use in *git gui blame* original location detection, measured in alphanumeric characters. See the git-blame(1) manual for more information on copy detection.

gui.blamehistoryctx
: Specifies the radius of history context in days to show in gitk(1) for the selected commit, when the `Show History Context` menu item is invoked from *git gui blame*. If this variable is set to zero, the whole history is shown.

guitool.<name>.cmd
: Specifies the shell command line to execute when the corresponding item of the git-gui(1) `Tools` menu is invoked. This option is mandatory for every tool. The command is executed from the root of the working directory, and in the environment it receives the name of the tool as *GIT_GUITOOL*, the name of the currently selected file as *FILENAME*, and the name of the current branch as *CUR_BRANCH* (if the head is detached, *CUR_BRANCH* is empty).

guitool.<name>.needsfile
: Run the tool only if a diff is selected in the GUI. It guarantees that *FILENAME* is not empty.

guitool.<name>.noconsole
: Run the command silently, without creating a window to display its output.

guitool.<name>.norescan
: Don’t rescan the working directory for changes after the tool finishes execution.

guitool.<name>.confirm
: Show a confirmation dialog before actually running the tool.

guitool.<name>.argprompt
: Request a string argument from the user, and pass it to the tool through the *ARGS* environment variable. Since requesting an argument implies confirmation, the *confirm* option has no effect if this is enabled. If the option is set to *true*, *yes*, or *1*, the dialog uses a built-in generic prompt; otherwise the exact value of the variable is used.

guitool.<name>.revprompt
: Request a single valid revision from the user, and set the *REVISION* environment variable. In other aspects this option is similar to *argprompt*, and can be used together with it.

guitool.<name>.revunmerged
: Show only unmerged branches in the *revprompt* subdialog. This is useful for tools similar to merge or rebase, but not for things like checkout or reset.

guitool.<name>.title
: Specifies the title to use for the prompt dialog. The default is the tool name.

guitool.<name>.prompt
: Specifies the general prompt string to display at the top of the dialog, before subsections for *argprompt* and *revprompt*. The default value includes the actual command.

help.browser
: Specify the browser that will be used to display help in the *web* format. See git-help(1).

help.format
: Override the default help format used by git-help(1). Values *man*, *info*, *web* and *html*

are supported. *man* is the default. *web* and *html* are the same.

help.autocorrect
: Automatically correct and execute mistyped commands after waiting for the given number of deciseconds (0.1 sec). If more than one command can be deduced from the entered text, nothing will be executed. If the value of this option is negative, the corrected command will be executed immediately. If the value is 0 - the command will be just shown but not executed. This is the default.

help.htmlpath
: Specify the path where the HTML documentation resides. File system paths and URLs are supported. HTML pages will be prefixed with this path when help is displayed in the *web* format. This defaults to the documentation path of your Git installation.

http.proxy
: Override the HTTP proxy, normally configured using the *http_proxy*, *https_proxy*, and *all_proxy* environment variables (see `curl(1)`). This can be overridden on a per-remote basis; see remote.<name>.proxy

http.cookiefile
: File containing previously stored cookie lines which should be used in the git http session, if they match the server. The file format of the file to read cookies from should be plain HTTP headers or the Netscape/Mozilla cookie file format (see curl(1)). NOTE that the file specified with http.cookiefile is only used as input. No cookies will be stored in the file.

http.sslVerify
: Whether to verify the SSL certificate when fetching or pushing over HTTPS. Can be overridden by the *GIT_SSL_NO_VERIFY* environment variable.

http.sslCert
: File containing the SSL certificate when fetching or pushing over HTTPS. Can be overridden by the *GIT_SSL_CERT* environment variable.

http.sslKey
: File containing the SSL private key when fetching or pushing over HTTPS. Can be overridden by the *GIT_SSL_KEY* environment variable.

http.sslCertPasswordProtected
: Enable git's password prompt for the SSL certificate. Otherwise OpenSSL will prompt the user, possibly many times, if the certificate or private key is encrypted. Can be overridden by the *GIT_SSL_CERT_PASSWORD_PROTECTED* environment variable.

http.sslCAInfo
: File containing the certificates to verify the peer with when fetching or pushing over HTTPS. Can be overridden by the *GIT_SSL_CAINFO* environment variable.

http.sslCAPath
: Path containing files with the CA certificates to verify the peer with when fetching or pushing over HTTPS. Can be overridden by the *GIT_SSL_CAPATH* environment variable.

http.maxRequests
: How many HTTP requests to launch in parallel. Can be overridden by the *GIT_HTTP_MAX_REQUESTS* environment variable. Default is 5.

http.minSessions
: The number of curl sessions (counted across slots) to be kept across requests. They will not be ended with curl_easy_cleanup() until http_cleanup() is invoked. If USE_CURL_MULTI is not defined, this value will be capped at 1. Defaults to 1.

http.postBuffer
: Maximum size in bytes of the buffer used by smart HTTP transports when POSTing data to the remote system. For requests larger than this buffer size, HTTP/1.1 and Transfer-Encoding: chunked is used to avoid creating a massive pack file locally. Default is 1 MiB, which is sufficient for most requests.

http.lowSpeedLimit, http.lowSpeedTime
: If the HTTP transfer speed is less than *http.lowSpeedLimit* for longer than *http.lowSpeedTime* seconds, the transfer is aborted. Can be overridden by the *GIT_HTTP_LOW_SPEED_LIMIT* and *GIT_HTTP_LOW_SPEED_TIME* environment variables.

http.noEPSV
: A boolean which disables using of EPSV ftp command by curl. This can helpful with some "poor" ftp servers which don’t support EPSV mode. Can be overridden by the *GIT_CURL_FTP_NO_EPSV* environment variable. Default is false (curl will use EPSV).

http.useragent
: The HTTP USER_AGENT string presented to an HTTP server. The default value represents the version of the client git such as git/1.7.1. This option allows you to override this value to a more common value such as Mozilla/4.0. This may be necessary, for instance, if connecting through a firewall that restricts HTTP connections to a set of common USER_AGENT strings (but not including those like git/1.7.1). Can be overridden by the *GIT_HTTP_USER_AGENT* environment variable.

i18n.commitEncoding
: Character encoding the commit messages are stored in; git itself does not care per se, but this information is necessary e.g. when importing commits from emails or in the gitk graphical history browser (and possibly at other places in the future or in other porcelains). See e.g. git-mailinfo(1). Defaults to *utf-8*.

i18n.logOutputEncoding
: Character encoding the commit messages are converted to when running *git log* and friends.

imap
: The configuration variables in the *imap* section are described in git-imap-send(1).

init.templatedir
: Specify the directory from which templates will be copied. (See the "TEMPLATE DIRECTORY" section of git-init(1).)

instaweb.browser
: Specify the program that will be used to browse your working repository in gitweb. See git-instaweb(1).

instaweb.httpd
: The HTTP daemon command-line to start gitweb on your working repository. See git-instaweb(1).

instaweb.local
: If true the web server started by git-instaweb(1) will be bound to the local IP (127.0.0.1).

instaweb.modulepath
: The default module path for git-instaweb(1) to use instead of /usr/lib/apache2 /modules. Only used if httpd is Apache.

instaweb.port

The port number to bind the gitweb httpd to. See git-instaweb(1).

interactive.singlekey

In interactive commands, allow the user to provide one-letter input with a single key (i.e., without hitting enter). Currently this is used by the `--patch` mode of git-add(1), git-checkout(1), git-commit(1), git-reset(1), and git-stash(1). Note that this setting is silently ignored if portable keystroke input is not available.

log.abbrevCommit

If true, makes git-log(1), git-show(1), and git-whatchanged(1) assume `--abbrev-commit`. You may override this option with `--no-abbrev-commit`.

log.date

Set the default date-time mode for the *log* command. Setting a value for log.date is similar to using *git log*'s `--date` option. Possible values are `relative`, `local`, `default`, `iso`, `rfc`, and `short`; see git-log(1) for details.

log.decorate

Print out the ref names of any commits that are shown by the log command. If *short* is specified, the ref name prefixes *refs/heads/*, *refs/tags/* and *refs/remotes/* will not be printed. If *full* is specified, the full ref name (including prefix) will be printed. This is the same as the log commands *--decorate* option.

log.showroot

If true, the initial commit will be shown as a big creation event. This is equivalent to a diff against an empty tree. Tools like git-log(1) or git-whatchanged(1), which normally hide the root commit will now show it. True by default.

mailmap.file

The location of an augmenting mailmap file. The default mailmap, located in the root of the repository, is loaded first, then the mailmap file pointed to by this variable. The location of the mailmap file may be in a repository subdirectory, or somewhere outside of the repository itself. See git-shortlog(1) and git-blame(1).

man.viewer

Specify the programs that may be used to display help in the *man* format. See git-help(1).

man.<tool>.cmd

Specify the command to invoke the specified man viewer. The specified command is evaluated in shell with the man page passed as argument. (See git-help(1).)

man.<tool>.path

Override the path for the given tool that may be used to display help in the *man* format. See git-help(1).

merge.conflictstyle

Specify the style in which conflicted hunks are written out to working tree files upon merge. The default is "merge", which shows a `<<<<<<<` conflict marker, changes made by one side, a `=======` marker, changes made by the other side, and then a `>>>>>>>` marker. An alternate style, "diff3", adds a `|||||||` marker and the original text before the `=======` marker.

merge.defaultToUpstream

If merge is called without any commit argument, merge the upstream branches configured for the current branch by using their last observed values stored in their remote-tracking branches. The values of the `branch.<current branch>.merge` that name the branches at the remote named by `branch.<current branch>.remote` are consulted, and then they are mapped via `remote.<remote>.fetch` to their corresponding remote-tracking branches, and the tips of these tracking branches are

merged.

merge.ff

By default, git does not create an extra merge commit when merging a commit that is a descendant of the current commit. Instead, the tip of the current branch is fast-forwarded. When set to `false`, this variable tells git to create an extra merge commit in such a case (equivalent to giving the `--no-ff` option from the command line). When set to `only`, only such fast-forward merges are allowed (equivalent to giving the `--ff-only` option from the command line).

merge.log

In addition to branch names, populate the log message with at most the specified number of one-line descriptions from the actual commits that are being merged. Defaults to false, and true is a synonym for 20.

merge.renameLimit

The number of files to consider when performing rename detection during a merge; if not specified, defaults to the value of diff.renameLimit.

merge.renormalize

Tell git that canonical representation of files in the repository has changed over time (e.g. earlier commits record text files with CRLF line endings, but recent ones use LF line endings). In such a repository, git can convert the data recorded in commits to a canonical form before performing a merge to reduce unnecessary conflicts. For more information, see section "Merging branches with differing checkin/checkout attributes" in gitattributes(5).

merge.stat

Whether to print the diffstat between ORIG_HEAD and the merge result at the end of the merge. True by default.

merge.tool

Controls which merge resolution program is used by git-mergetool(1). Valid built-in values are: "araxis", "bc3", "diffuse", "ecmerge", "emerge", "gvimdiff", "kdiff3", "meld", "opendiff", "p4merge", "tkdiff", "tortoisemerge", "vimdiff" and "xxdiff". Any other value is treated is custom merge tool and there must be a corresponding mergetool.<tool>.cmd option.

merge.verbosity

Controls the amount of output shown by the recursive merge strategy. Level 0 outputs nothing except a final error message if conflicts were detected. Level 1 outputs only conflicts, 2 outputs conflicts and file changes. Level 5 and above outputs debugging information. The default is level 2. Can be overridden by the *GIT_MERGE_VERBOSITY* environment variable.

merge.<driver>.name

Defines a human-readable name for a custom low-level merge driver. See gitattributes(5) for details.

merge.<driver>.driver

Defines the command that implements a custom low-level merge driver. See gitattributes(5) for details.

merge.<driver>.recursive

Names a low-level merge driver to be used when performing an internal merge between common ancestors. See gitattributes(5) for details.

mergetool.<tool>.path

Override the path for the given tool. This is useful in case your tool is not in the PATH.

mergetool.<tool>.cmd

Specify the command to invoke the specified merge tool. The specified command is evaluated in shell with the following variables available: *BASE* is the name of a temporary file containing the common base of the files to be merged, if available; *LOCAL* is the name of a temporary file containing the contents of the file on the current branch; *REMOTE* is the name of a temporary file containing the contents of the file from the branch being merged; *MERGED* contains the name of the file to which the merge tool should write the results of a successful merge.

mergetool.<tool>.trustExitCode

For a custom merge command, specify whether the exit code of the merge command can be used to determine whether the merge was successful. If this is not set to true then the merge target file timestamp is checked and the merge assumed to have been successful if the file has been updated, otherwise the user is prompted to indicate the success of the merge.

mergetool.keepBackup

After performing a merge, the original file with conflict markers can be saved as a file with a `.orig` extension. If this variable is set to `false` then this file is not preserved. Defaults to `true` (i.e. keep the backup files).

mergetool.keepTemporaries

When invoking a custom merge tool, git uses a set of temporary files to pass to the tool. If the tool returns an error and this variable is set to `true`, then these temporary files will be preserved, otherwise they will be removed after the tool has exited. Defaults to `false`.

mergetool.prompt

Prompt before each invocation of the merge resolution program.

notes.displayRef

The (fully qualified) refname from which to show notes when showing commit messages. The value of this variable can be set to a glob, in which case notes from all matching refs will be shown. You may also specify this configuration variable several times. A warning will be issued for refs that do not exist, but a glob that does not match any refs is silently ignored.

This setting can be overridden with the `GIT_NOTES_DISPLAY_REF` environment variable, which must be a colon separated list of refs or globs.

The effective value of "core.notesRef" (possibly overridden by GIT_NOTES_REF) is also implicitly added to the list of refs to be displayed.

notes.rewrite.<command>

When rewriting commits with <command> (currently `amend` or `rebase`) and this variable is set to `true`, git automatically copies your notes from the original to the rewritten commit. Defaults to `true`, but see "notes.rewriteRef" below.

notes.rewriteMode

When copying notes during a rewrite (see the "notes.rewrite.<command>" option), determines what to do if the target commit already has a note. Must be one of `overwrite`, `concatenate`, or `ignore`. Defaults to `concatenate`.

This setting can be overridden with the `GIT_NOTES_REWRITE_MODE` environment variable.

notes.rewriteRef

When copying notes during a rewrite, specifies the (fully qualified) ref whose notes should be copied. The ref may be a glob, in which case notes in all matching refs will be copied. You may also specify this configuration several times.

Does not have a default value; you must configure this variable to enable note rewriting. Set it to `refs/notes/commits` to enable rewriting for the default commit notes.

This setting can be overridden with the `GIT_NOTES_REWRITE_REF` environment variable, which must be a colon separated list of refs or globs.

pack.window

The size of the window used by git-pack-objects(1) when no window size is given on the command line. Defaults to 10.

pack.depth

The maximum delta depth used by git-pack-objects(1) when no maximum depth is given on the command line. Defaults to 50.

pack.windowMemory

The window memory size limit used by git-pack-objects(1) when no limit is given on the command line. The value can be suffixed with "k", "m", or "g". Defaults to 0, meaning no limit.

pack.compression

An integer -1..9, indicating the compression level for objects in a pack file. -1 is the zlib default. 0 means no compression, and 1..9 are various speed/size tradeoffs, 9 being slowest. If not set, defaults to core.compression. If that is not set, defaults to -1, the zlib default, which is "a default compromise between speed and compression (currently equivalent to level 6)."

Note that changing the compression level will not automatically recompress all existing objects. You can force recompression by passing the -F option to git-repack(1).

pack.deltaCacheSize

The maximum memory in bytes used for caching deltas in git-pack-objects(1) before writing them out to a pack. This cache is used to speed up the writing object phase by not having to recompute the final delta result once the best match for all objects is found. Repacking large repositories on machines which are tight with memory might be badly impacted by this though, especially if this cache pushes the system into swapping. A value of 0 means no limit. The smallest size of 1 byte may be used to virtually disable this cache. Defaults to 256 MiB.

pack.deltaCacheLimit

The maximum size of a delta, that is cached in git-pack-objects(1). This cache is used to speed up the writing object phase by not having to recompute the final delta result once the best match for all objects is found. Defaults to 1000.

pack.threads

Specifies the number of threads to spawn when searching for best delta matches. This requires that git-pack-objects(1) be compiled with pthreads otherwise this option is ignored with a warning. This is meant to reduce packing time on multiprocessor machines. The required amount of memory for the delta search window is however multiplied by the number of threads. Specifying 0 will cause git to auto-detect the number of CPU’s and set the number of threads accordingly.

pack.indexVersion

Specify the default pack index version. Valid values are 1 for legacy pack index used by Git versions prior to 1.5.2, and 2 for the new pack index with capabilities for packs larger than 4 GB as well as proper protection against the repacking of corrupted packs. Version 2 is the default. Note that version 2 is enforced and this config option ignored whenever the corresponding pack is larger than 2 GB.

If you have an old git that does not understand the version 2 `*.idx` file, cloning or

fetching over a non native protocol (e.g. "http" and "rsync") that will copy both `*.pack` file and corresponding `*.idx` file from the other side may give you a repository that cannot be accessed with your older version of git. If the `*.pack` file is smaller than 2 GB, however, you can use git-index-pack(1) on the *.pack file to regenerate the `*.idx` file.

pack.packSizeLimit

The maximum size of a pack. This setting only affects packing to a file when repacking, i.e. the git:// protocol is unaffected. It can be overridden by the `--max-pack-size` option of git-repack(1). The minimum size allowed is limited to 1 MiB. The default is unlimited. Common unit suffixes of *k*, *m*, or *g* are supported.

pager.<cmd>

If the value is boolean, turns on or off pagination of the output of a particular git subcommand when writing to a tty. Otherwise, turns on pagination for the subcommand using the pager specified by the value of `pager.<cmd>`. If `--paginate` or `--no-pager` is specified on the command line, it takes precedence over this option. To disable pagination for all commands, set `core.pager` or `GIT_PAGER` to `cat`.

pretty.<name>

Alias for a --pretty= format string, as specified in git-log(1). Any aliases defined here can be used just as the built-in pretty formats could. For example, running `git config pretty.changelog "format:* %H %s"` would cause the invocation `git log --pretty=changelog` to be equivalent to running `git log "--pretty=format:* %H %s"`. Note that an alias with the same name as a built-in format will be silently ignored.

pull.rebase

When true, rebase branches on top of the fetched branch, instead of merging the default branch from the default remote when "git pull" is run. See "branch.<name>.rebase" for setting this on a per-branch basis.

NOTE: this is a possibly dangerous operation; do **not** use it unless you understand the implications (see git-rebase(1) for details).

pull.octopus

The default merge strategy to use when pulling multiple branches at once.

pull.twohead

The default merge strategy to use when pulling a single branch.

push.default

Defines the action git push should take if no refspec is given on the command line, no refspec is configured in the remote, and no refspec is implied by any of the options given on the command line. Possible values are:

- `nothing` - do not push anything.
- `matching` - push all branches having the same name in both ends. This is for those who prepare all the branches into a publishable shape and then push them out with a single command. It is not appropriate for pushing into a repository shared by multiple users, since locally stalled branches will attempt a non-fast forward push if other users updated the branch.
  This is currently the default, but Git 2.0 will change the default to `simple`.
- `upstream` - push the current branch to its upstream branch. With this, `git push` will update the same remote ref as the one which is merged by `git pull`, making `push` and `pull` symmetrical. See "branch.<name>.merge" for how to configure the upstream branch.
- `simple` - like `upstream`, but refuses to push if the upstream branch's name is different from the local one. This is the safest option and is well-suited for

beginners. It will become the default in Git 2.0.

- `current` - push the current branch to a branch of the same name.

The `simple`, `current` and `upstream` modes are for those who want to push out a single branch after finishing work, even when the other branches are not yet ready to be pushed out. If you are working with other people to push into the same shared repository, you would want to use one of these.

rebase.stat
: Whether to show a diffstat of what changed upstream since the last rebase. False by default.

rebase.autosquash
: If set to true enable *--autosquash* option by default.

receive.autogc
: By default, git-receive-pack will run "git-gc --auto" after receiving data from git-push and updating refs. You can stop it by setting this variable to false.

receive.fsckObjects
: If it is set to true, git-receive-pack will check all received objects. It will abort in the case of a malformed object or a broken link. The result of an abort are only dangling objects. Defaults to false. If not set, the value of `transfer.fsckObjects` is used instead.

receive.unpackLimit
: If the number of objects received in a push is below this limit then the objects will be unpacked into loose object files. However if the number of received objects equals or exceeds this limit then the received pack will be stored as a pack, after adding any missing delta bases. Storing the pack from a push can make the push operation complete faster, especially on slow filesystems. If not set, the value of `transfer.unpackLimit` is used instead.

receive.denyDeletes
: If set to true, git-receive-pack will deny a ref update that deletes the ref. Use this to prevent such a ref deletion via a push.

receive.denyDeleteCurrent
: If set to true, git-receive-pack will deny a ref update that deletes the currently checked out branch of a non-bare repository.

receive.denyCurrentBranch
: If set to true or "refuse", git-receive-pack will deny a ref update to the currently checked out branch of a non-bare repository. Such a push is potentially dangerous because it brings the HEAD out of sync with the index and working tree. If set to "warn", print a warning of such a push to stderr, but allow the push to proceed. If set to false or "ignore", allow such pushes with no message. Defaults to "refuse".

receive.denyNonFastForwards
: If set to true, git-receive-pack will deny a ref update which is not a fast-forward. Use this to prevent such an update via a push, even if that push is forced. This configuration variable is set when initializing a shared repository.

receive.updateserverinfo
: If set to true, git-receive-pack will run git-update-server-info after receiving data from git-push and updating refs.

remote.<name>.url
: The URL of a remote repository. See git-fetch(1) or git-push(1).

remote.<name>.pushurl

The push URL of a remote repository. See git-push(1).

remote.<name>.proxy
: For remotes that require curl (http, https and ftp), the URL to the proxy to use for that remote. Set to the empty string to disable proxying for that remote.

remote.<name>.fetch
: The default set of "refspec" for git-fetch(1). See git-fetch(1).

remote.<name>.push
: The default set of "refspec" for git-push(1). See git-push(1).

remote.<name>.mirror
: If true, pushing to this remote will automatically behave as if the `--mirror` option was given on the command line.

remote.<name>.skipDefaultUpdate
: If true, this remote will be skipped by default when updating using git-fetch(1) or the `update` subcommand of git-remote(1).

remote.<name>.skipFetchAll
: If true, this remote will be skipped by default when updating using git-fetch(1) or the `update` subcommand of git-remote(1).

remote.<name>.receivepack
: The default program to execute on the remote side when pushing. See option --receive-pack of git-push(1).

remote.<name>.uploadpack
: The default program to execute on the remote side when fetching. See option --upload-pack of git-fetch-pack(1).

remote.<name>.tagopt
: Setting this value to --no-tags disables automatic tag following when fetching from remote <name>. Setting it to --tags will fetch every tag from remote <name>, even if they are not reachable from remote branch heads. Passing these flags directly to git-fetch(1) can override this setting. See options --tags and --no-tags of git-fetch(1).

remote.<name>.vcs
: Setting this to a value <vcs> will cause git to interact with the remote with the git-remote-<vcs> helper.

remotes.<group>
: The list of remotes which are fetched by "git remote update <group>". See git-remote(1).

repack.usedeltabaseoffset
: By default, git-repack(1) creates packs that use delta-base offset. If you need to share your repository with git older than version 1.4.4, either directly or via a dumb protocol such as http, then you need to set this option to "false" and repack. Access from old git versions over the native protocol are unaffected by this option.

rerere.autoupdate
: When set to true, `git-rerere` updates the index with the resulting contents after it cleanly resolves conflicts using previously recorded resolution. Defaults to false.

rerere.enabled
: Activate recording of resolved conflicts, so that identical conflict hunks can be resolved automatically, should they be encountered again. By default, git-rerere(1) is enabled if there is an `rr-cache` directory under the `$GIT_DIR`, e.g. if "rerere" was previously used in the repository.

sendemail.identity
: A configuration identity. When given, causes values in the *sendemail.<identity>* subsection to take precedence over values in the *sendemail* section. The default identity is the value of *sendemail.identity*.

sendemail.smtpencryption
: See git-send-email(1) for description. Note that this setting is not subject to the *identity* mechanism.

sendemail.smtpssl
: Deprecated alias for *sendemail.smtpencryption = ssl*.

sendemail.<identity>.*
: Identity-specific versions of the *sendemail.** parameters found below, taking precedence over those when the this identity is selected, through command-line or *sendemail.identity*.

sendemail.aliasesfile

sendemail.aliasfiletype

sendemail.bcc

sendemail.cc

sendemail.cccmd

sendemail.chainreplyto

sendemail.confirm

sendemail.envelopesender

sendemail.from

sendemail.multiedit

sendemail.signedoffbycc

sendemail.smtppass

sendemail.suppresscc

sendemail.suppressfrom

sendemail.to

sendemail.smtpdomain

sendemail.smtpserver

sendemail.smtpserverport

sendemail.smtpserveroption

sendemail.smtpuser

sendemail.thread

sendemail.validate
: See git-send-email(1) for description.

sendemail.signedoffcc
: Deprecated alias for *sendemail.signedoffbycc*.

showbranch.default
: The default set of branches for git-show-branch(1). See git-show-branch(1).

status.relativePaths
: By default, git-status(1) shows paths relative to the current directory. Setting this

variable to `false` shows paths relative to the repository root (this was the default for git prior to v1.5.4).

status.showUntrackedFiles

By default, git-status(1) and git-commit(1) show files which are not currently tracked by Git. Directories which contain only untracked files, are shown with the directory name only. Showing untracked files means that Git needs to lstat() all all the files in the whole repository, which might be slow on some systems. So, this variable controls how the commands displays the untracked files. Possible values are:

- `no` - Show no untracked files.
- `normal` - Show untracked files and directories.
- `all` - Show also individual files in untracked directories.

If this variable is not specified, it defaults to *normal*. This variable can be overridden with the -u|--untracked-files option of git-status(1) and git-commit(1).

status.submodulesummary

Defaults to false. If this is set to a non zero number or true (identical to -1 or an unlimited number), the submodule summary will be enabled and a summary of commits for modified submodules will be shown (see --summary-limit option of git-submodule(1)).

submodule.<name>.path

submodule.<name>.url

submodule.<name>.update

The path within this project, URL, and the updating strategy for a submodule. These variables are initially populated by *git submodule init*; edit them to override the URL and other values found in the `.gitmodules` file. See git-submodule(1) and gitmodules(5) for details.

submodule.<name>.fetchRecurseSubmodules

This option can be used to control recursive fetching of this submodule. It can be overridden by using the --[no-]recurse-submodules command line option to "git fetch" and "git pull". This setting will override that from in the gitmodules(5) file.

submodule.<name>.ignore

Defines under what circumstances "git status" and the diff family show a submodule as modified. When set to "all", it will never be considered modified, "dirty" will ignore all changes to the submodules work tree and takes only differences between the HEAD of the submodule and the commit recorded in the superproject into account. "untracked" will additionally let submodules with modified tracked files in their work tree show up. Using "none" (the default when this option is not set) also shows submodules that have untracked files in their work tree as changed. This setting overrides any setting made in .gitmodules for this submodule, both settings can be overridden on the command line by using the "--ignore-submodules" option.

tar.umask

This variable can be used to restrict the permission bits of tar archive entries. The default is 0002, which turns off the world write bit. The special value "user" indicates that the archiving user's umask will be used instead. See umask(2) and git-archive(1).

transfer.fsckObjects

When `fetch.fsckObjects` or `receive.fsckObjects` are not set, the value of this variable is used instead. Defaults to false.

transfer.unpackLimit

When `fetch.unpackLimit` or `receive.unpackLimit` are not set, the value of this

variable is used instead. The default value is 100.

url.<base>.insteadOf
: Any URL that starts with this value will be rewritten to start, instead, with <base>. In cases where some site serves a large number of repositories, and serves them with multiple access methods, and some users need to use different access methods, this feature allows people to specify any of the equivalent URLs and have git automatically rewrite the URL to the best alternative for the particular user, even for a never-before-seen repository on the site. When more than one insteadOf strings match a given URL, the longest match is used.

url.<base>.pushInsteadOf
: Any URL that starts with this value will not be pushed to; instead, it will be rewritten to start with <base>, and the resulting URL will be pushed to. In cases where some site serves a large number of repositories, and serves them with multiple access methods, some of which do not allow push, this feature allows people to specify a pull-only URL and have git automatically use an appropriate URL to push, even for a never-before-seen repository on the site. When more than one pushInsteadOf strings match a given URL, the longest match is used. If a remote has an explicit pushurl, git will ignore this setting for that remote.

user.email
: Your email address to be recorded in any newly created commits. Can be overridden by the *GIT_AUTHOR_EMAIL*, *GIT_COMMITTER_EMAIL*, and *EMAIL* environment variables. See git-commit-tree(1).

user.name
: Your full name to be recorded in any newly created commits. Can be overridden by the *GIT_AUTHOR_NAME* and *GIT_COMMITTER_NAME* environment variables. See git-commit-tree(1).

user.signingkey
: If git-tag(1) is not selecting the key you want it to automatically when creating a signed tag, you can override the default selection with this variable. This option is passed unchanged to gpg's --local-user parameter, so you may specify a key using any method that gpg supports.

web.browser
: Specify a web browser that may be used by some commands. Currently only git-instaweb(1) and git-help(1) may use it.

GIT

Part of the git(1) suite

Last updated 2013-02-15 19:06:56 UTC