Rocky Enterprise Linux 9.2 Manual Pages on command 'git-config.1'

C:\>man git-config.1

GIT-CONFIG(1) Git Manual GIT-CONFIG(1)

NAME

git-config - Get and set repository or global options

SYNOPSIS

git config [<file-option>] [--type=<type>] [--show-origin] [-z|--null] name [value [value_regex]]

git config [<file-option>] [--type=<type>] --add name value

git config [<file-option>] [--type=<type>] --replace-all name value [value_regex]

git config [<file-option>] [--type=<type>] [--show-origin] [-z|--null] --get name [value_regex]

git config [<file-option>] [--type=<type>] [--show-origin] [-z|--null] --get-all name [value_regex]

git config [<file-option>] [--type=<type>] [--show-origin] [-z|--null] [--name-only] --get-regexp name_regex [value_regex]

git config [<file-option>] [--type=<type>] [-z|--null] --get-urlmatch name URL

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>] [--show-origin] [-z|--null] [--name-only] -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 the section called ?EXAMPLES?).

The --type=<type> option instructs git config to ensure that incoming and outgoing values are canonicalize-able under the given <type>. If no --type=<type> is given, no canonicalization will be performed. Callers may unset an existing --type specifier with --no-type.

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

When writing, the new value is written to the repository local configuration file by default, and options --system, --global, --worktree, --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:

? The section or key is invalid (ret=1),

? no section or name was provided (ret=2),

? the config file is invalid (ret=3),

? the config file cannot be written (ret=4),

? you try to unset an option which does not exist (ret=5),

? you try to unset/set an option for which multiple lines match (ret=5), or

? 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 the last value if multiple key values were found.

--get-all

Like get, but returns all values for a multi-valued key.

--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.

--get-urlmatch name URL

When given a two-part name section.key, the value for section.<url>.key whose <url> part matches the best to the given URL is returned (if no such key exists, the value for section.key is used as a fallback). When given just the section as name, do so for all the keys in the section and list them. Returns error code 1 if no value is found.

--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 the section called ?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 the section called ?FILES?.

--local

For writing options: write to the repository .git/config file. This is the

default behavior.

For reading options: read only from the repository .git/config rather than from all available files.

See also the section called ?FILES?.

--worktree

Similar to --local except that .git/config.worktree is read from or written to if extensions.worktreeConfig is present. If not it?s the same as --local.

-f config-file, --file config-file

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

--blob blob

Similar to --file but use the given blob instead of a file. E.g. you can use master:.gitmodules to read values from the file .gitmodules in the master branch. See "SPECIFYING REVISIONS" section in gitrevisions(7) for a more complete list of ways to spell blob names.

--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, along with their values.

--type <type>

git config will ensure that any input or output is valid under the given type constraint(s), and will canonicalize outgoing values in <type>'s canonical form.

Valid <type>'s include:

? bool: canonicalize values as either "true" or "false".

? int: canonicalize values as simple decimal numbers. An optional suffix of k, m, or g will cause the value to be multiplied by 1024, 1048576, or 1073741824 upon input.

? bool-or-int: canonicalize according to either bool or int, as described above.

? path: canonicalize by adding a leading ~ to the value of $HOME and ~user to the home directory for the specified user. This specifier has no effect when setting the value (but you can use git config section.variable ~/ from the command line to let your shell do the expansion.)

? expiry-date: canonicalize by converting from a fixed or relative date-string to a timestamp. This specifier has no effect when setting the value.

? color: When getting a value, canonicalize by converting to an ANSI color escape sequence. When setting a value, a sanity-check is performed to ensure that the given value is canonicalize-able as an ANSI color, but it is written as-is.

--bool, --int, --bool-or-int, --path, --expiry-date

Historical options for selecting a type specifier. Prefer instead --type (see above).

--no-type

Un-sets the previously set type specifier (if one was previously set). This option requests that git config not canonicalize the retrieved variable. --no-type has no effect without --type=<type> or --<type>.

-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.

--name-only

Output only the names of config variables for --list or --get-regexp.

--show-origin

Augment the output of all queried config options with the origin type (file, standard input, blob, command line) and the actual origin (config file path, ref, or blob id if applicable).

--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.

--type=color [--default=<default>] is preferred over --get-color (but note that --get-color will omit the trailing newline printed by --type=color).

-e, --edit

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

--[no-]includes

Respect include.* directives in config files when looking up values. Defaults to off when a specific file is given (e.g., using --file, --global, etc) and on when searching all config files.

--default <value>

When using --get, and the requested variable is not found, behave as if <value> were the value assigned to the that variable.

CONFIGURATION

pager.config is only respected when listing configuration, i.e., when using --list or any of the --get-* which may return multiple results. The default is to use a pager.

FILES

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

$(prefix)/etc/gitconfig

System-wide 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.

~/.gitconfig

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

$GIT_DIR/config

Repository specific configuration file.

$GIT_DIR/config.worktree

This is optional and is only searched when extensions.worktreeConfig is present in $GIT_DIR/config.

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.

The files are read in the order given above, with last value found taking precedence over values read earlier. When multiple values are taken then all values of a key from all files will be used.

You may override individual configuration parameters when running any git command by using the -c option. See git(1) for details.

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, --system and --worktree options will limit the file used to the global, system-wide or per-worktree 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 the section called ?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
; HTTP
[http]
        sslVerify
[http "https://weak.example.com"]
        sslVerify = false
        cookieFile = /tmp/cookie.txt
```

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}"
```

For URLs in https://weak.example.com, http.sslVerify is set to false, while it is set to true for all others:

```
% git config --type=bool --get-urlmatch http.sslverify https://good.example.com
true
```

% git config --type=bool --get-urlmatch http.sslverify https://weak.example.com

false

% git config --get-urlmatch http https://weak.example.com

http.cookieFile /tmp/cookie.txt

http.sslverify false

CONFIGURATION FILE

The Git configuration file contains a number of variables that affect the Git commands' behavior. The files .git/config and optionally config.worktree (see extensions.worktreeConfig below) in each repository are 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; we say then that the variable is multivalued.

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 case-insensitive. 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 and the null byte. Doublequote " and backslash can be included by escaping them as \" and \\, respectively. Backslashes preceding other characters are dropped when

reading; for example, \t is read as t and \0 is read as 0 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 (or just name, which is a short-hand to say that the variable is the boolean "true"). The variable names are case-insensitive, allow only alphanumeric characters and -, and must start with an alphabetic character.

A line that defines a value can be continued to the next line by ending it with a \; the backquote and the end-of-line are stripped. Leading whitespaces after name =, the remainder of the line after the first comment character # or ;, and trailing whitespaces of the line are discarded unless they are enclosed in double quotes. Internal whitespaces within the value are retained verbatim.

Inside double quotes, double quote " and backslash \ characters 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). Other char escape sequences (including octal escape sequences) are invalid.

Includes

The include and includeIf sections allow you to include config directives from another source. These sections behave identically to each other with the exception that includeIf sections may be ignored if their condition does not evaluate to true; see "Conditional includes" below.

You can include a config file from another by setting the special include.path (or includeIf.*.path) variable to the name of the file to be included. The variable takes a pathname as its value, and is subject to tilde expansion. These variables can be given multiple times.

The contents of the included file are inserted immediately, as if they had been found at the location of the include directive. If the value of the variable is a

relative path, the path is considered to be relative to the configuration file in which the include directive was found. See below for examples.

Conditional includes

You can include a config file from another conditionally by setting a includeIf.<condition>.path variable to the name of the file to be included.

The condition starts with a keyword followed by a colon and some data whose format and meaning depends on the keyword. Supported keywords are:

gitdir

The data that follows the keyword gitdir: is used as a glob pattern. If the location of the .git directory matches the pattern, the include condition is met.

The .git location may be auto-discovered, or come from $GIT_DIR environment variable. If the repository is auto discovered via a .git file (e.g. from submodules, or a linked worktree), the .git location would be the final location where the .git directory is, not where the .git file is.

The pattern can contain standard globbing wildcards and two additional ones, **/ and /**, that can match multiple path components. Please refer to gitignore(5) for details. For convenience:

- If the pattern starts with ~/, ~ will be substituted with the content of the environment variable HOME.
- If the pattern starts with ./, it is replaced with the directory containing the current config file.
- If the pattern does not start with either ~/, ./ or /, **/ will be automatically prepended. For example, the pattern foo/bar becomes **/foo/bar and would match /any/path/to/foo/bar.
- If the pattern ends with /, ** will be automatically added. For example, the pattern foo/ becomes foo/**. In other words, it matches "foo" and everything inside, recursively.

gitdir/i

This is the same as gitdir except that matching is done case-insensitively (e.g. on case-insensitive file systems)

onbranch

The data that follows the keyword onbranch: is taken to be a pattern with

standard globbing wildcards and two additional ones, **/ and /**, that can match multiple path components. If we are in a worktree where the name of the branch that is currently checked out matches the pattern, the include condition is met.

If the pattern ends with /, ** will be automatically added. For example, the pattern foo/ becomes foo/**. In other words, it matches all branches that begin with foo/. This is useful if your branches are organized hierarchically and you would like to apply a configuration to all the branches in that hierarchy.

A few more notes on matching via gitdir and gitdir/i:

? Symlinks in $GIT_DIR are not resolved before matching.

? Both the symlink & realpath versions of paths will be matched outside of $GIT_DIR. E.g. if ~/git is a symlink to /mnt/storage/git, both gitdir:~/git and gitdir:/mnt/storage/git will match.

This was not the case in the initial release of this feature in v2.13.0, which only matched the realpath version. Configuration that wants to be compatible with the initial release of this feature needs to either specify only the realpath version, or both versions.

? Note that "../" is not special and will match literally, which is unlikely what you want.

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.inc ; find "foo.inc" relative to the current file
        path = ~/foo.inc ; find "foo.inc" in your `$HOME` directory
    ; include if $GIT_DIR is /path/to/foo/.git
    [includeIf "gitdir:/path/to/foo/.git"]
        path = /path/to/foo.inc
    ; include for all repositories inside /path/to/group
    [includeIf "gitdir:/path/to/group/"]
        path = /path/to/foo.inc
    ; include for all repositories inside $HOME/to/group
    [includeIf "gitdir:~/to/group/"]
        path = /path/to/foo.inc
    ; relative paths are always relative to the including
    ; file (if the condition is true); their location is not
    ; affected by the condition
    [includeIf "gitdir:/path/to/group/"]
        path = foo.inc
    ; include only if we are in a worktree where foo-branch is
    ; currently checked out
    [includeIf "onbranch:foo-branch"]
        path = foo.inc
```

Values

Values of many variables are treated as a simple string, but there are variables that take values of specific types and there are rules as to how to spell them.

boolean

When a variable is said to take a boolean value, many synonyms are accepted for true and false; these are all case-insensitive.

true

Boolean true literals are yes, on, true, and 1. Also, a variable defined without = <value> is taken as true.

false

Boolean false literals are no, off, false, 0 and the empty string.

When converting a value to its canonical form using the --type=bool type specifier, git config will ensure that the output is "true" or "false" (spelled in lowercase).

integer

The value for many variables that specify various sizes can be suffixed with k, M,... to mean "scale the number by 1024", "by 1024x1024", etc.

color

The value for a variable that takes a color is a list of colors (at most two, one for foreground and one for background) and attributes (as many as you want), separated by spaces.

The basic colors accepted are normal, black, red, green, yellow, blue, magenta, cyan and white. The first color given is the foreground; the second is the background.

Colors may also be given as numbers between 0 and 255; these use ANSI 256-color mode (but note that not all terminals may support this). If your terminal supports it, you may also specify 24-bit RGB values as hex, like #ff0ab3.

The accepted attributes are bold, dim, ul, blink, reverse, italic, and strike (for crossed-out or "strikethrough" letters). The position of any attributes with respect to the colors (before, after, or in between), doesn?t matter.

Specific attributes may be turned off by prefixing them with no or no- (e.g., noreverse, no-ul, etc).

An empty color string produces no color effect at all. This can be used to avoid coloring specific elements without disabling color entirely.

For git?s pre-defined color slots, the attributes are meant to be reset at the beginning of each item in the colored output. So setting color.decorate.branch to black will paint that branch name in a plain black, even if the previous thing on the same output line (e.g. opening parenthesis before the list of branch names in log --decorate output) is set to be painted with bold or some other attribute. However, custom log formats may do more complicated and layered coloring, and the negated forms may be useful there.

pathname

A variable that takes a pathname value can be given a string that begins with "~/" or "~user/", and the usual tilde expansion happens to such a string: ~/ is expanded to the value of $HOME, and ~user/ to the specified user?s 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.

Other git-related tools may and do use their own variables. When inventing new variables for use in your own tool, make sure their names do not conflict with those that are used by Git itself and other popular tools, and describe them in your 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:

fetchShowForcedUpdates

Advice shown when git-fetch(1) takes a long time to calculate forced updates after ref updates, or to warn that the check is disabled.

pushUpdateRejected

Set this variable to false if you want to disable pushNonFFCurrent, pushNonFFMatching, pushAlreadyExists, pushFetchFirst, and pushNeedsForce simultaneously.

pushNonFFCurrent

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

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.

pushAlreadyExists

Shown when git-push(1) rejects an update that does not qualify for fast-forwarding (e.g., a tag.)

pushFetchFirst

Shown when git-push(1) rejects an update that tries to overwrite a remote ref that points at an object we do not have.

pushNeedsForce

Shown when git-push(1) rejects an update that tries to overwrite a remote ref that points at an object that is not a commit-ish, or make the remote ref point at an object that is not a commit-ish.

pushUnqualifiedRefname

Shown when git-push(1) gives up trying to guess based on the source and destination refs what remote ref namespace the source belongs in, but where we can still suggest that the user push to either refs/heads/* or refs/tags/* based on the type of the source object.

statusAheadBehind

Shown when git-status(1) computes the ahead/behind counts for a local ref compared to its remote tracking ref, and that calculation takes longer than expected. Will not appear if status.aheadBehind is false or the option --no-ahead-behind is given.

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-switch(1) or git-checkout(1) when switching branch.

statusUoption

Advise to consider using the -u option to git-status(1) when the command takes more than 2 seconds to enumerate untracked files.

commitBeforeMerge

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

resetQuiet

Advice to consider using the --quiet option to git-reset(1) when the command takes more than 2 seconds to enumerate unstaged changes after reset.

resolveConflict

Advice shown by various commands when conflicts prevent the operation from being performed.

sequencerInUse

Advice shown when a sequencer command is already in progress.

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-switch(1) or git-checkout(1) to move to the detach HEAD state, to instruct how to create a local branch after the fact.

checkoutAmbiguousRemoteBranchName

Advice shown when the argument to git-checkout(1) and git-switch(1) ambiguously resolves to a remote tracking branch on more than one remote in situations where an unambiguous argument would have otherwise caused a remote-tracking branch to be checked out. See the checkout.defaultRemote configuration variable for how to set a given remote to used by default in some situations where this advice would be printed.

amWorkDir

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

rmHints

In case of failure in the output of git-rm(1), show directions on how to proceed from the current state.

addEmbeddedRepo

Advice on what to do when you?ve accidentally added one git repo inside of another.

ignoredHook

Advice shown if a hook is ignored because the hook is not set as executable.

waitingForEditor

Print a message to the terminal whenever Git is waiting for editor input from the user.

nestedTag

Advice shown if a user attempts to recursively tag a tag object.

submoduleAlternateErrorStrategyDie

Advice shown when a submodule.alternateErrorStrategy option configured to "die" causes a fatal error.

core.fileMode

Tells Git if the executable bit of files in the working tree is to be honored.

Some filesystems lose the executable bit when a file that is marked as executable is checked out, or checks out a non-executable file with executable bit on. git-clone(1) or git-init(1) probe the filesystem to see if it handles the executable bit correctly and this variable is automatically set as necessary.

A repository, however, may be on a filesystem that handles the filemode correctly, and this variable is set to true when created, but later may be made accessible from another environment that loses the filemode (e.g. exporting ext4 via CIFS mount, visiting a Cygwin created repository with Git for Windows or Eclipse). In such a case it may be necessary to set this variable to false. See git-update-index(1).

The default is true (when core.filemode is not specified in the config file).

core.hideDotFiles

(Windows-only) If true, mark newly-created directories and files whose name starts with a dot as hidden. If dotGitOnly, only the .git/ directory is hidden, but no other files starting with a dot. The default mode is dotGitOnly.

core.ignoreCase

Internal variable which enables various workarounds to enable Git to work better on filesystems that are not case sensitive, like APFS, HFS+, FAT, NTFS, etc. 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.

Git relies on the proper configuration of this variable for your operating and file system. Modifying this value may result in unexpected behavior.

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.protectHFS

If set to true, do not allow checkout of paths that would be considered equivalent to .git on an HFS+ filesystem. Defaults to true on Mac OS, and false elsewhere.

core.protectNTFS

If set to true, do not allow checkout of paths that would cause problems with the NTFS filesystem, e.g. conflict with 8.3 "short" names. Defaults to true on Windows, and false elsewhere.

core.fsmonitor

If set, the value of this variable is used as a command which will identify all files that may have changed since the requested date/time. This information is used to speed up git by avoiding unnecessary processing of files that have not changed. See the "fsmonitor-watchman" section of githooks(5).

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.splitIndex

If true, the split-index feature of the index will be used. See git-update-index(1). False by default.

core.untrackedCache

Determines what to do about the untracked cache feature of the index. It will be kept, if this variable is unset or set to keep. It will automatically be added if set to true. And it will automatically be removed, if set to false. Before setting it to true, you should check that mtime is working properly on your system. See git-update-index(1). keep by default, unless

feature.manyFiles is enabled which sets this setting to true by default.

core.checkStat

When missing or is set to default, many fields in the stat structure are checked to detect if a file has been modified since Git looked at it. When this configuration variable is set to minimal, sub-second part of mtime and ctime, the uid and gid of the owner of the file, the inode number (and the device number, if Git was compiled to use it), are excluded from the check among these fields, leaving only the whole-second part of mtime (and ctime, if core.trustCtime is set) and the filesize to be checked.

There are implementations of Git that do not leave usable values in some fields (e.g. JGit); by excluding these fields from the comparison, the minimal mode may help interoperability when the same repository is used by these other systems at the same time.

core.quotePath

Commands that output paths (e.g. ls-files, diff), will quote "unusual" characters in the pathname by enclosing the pathname in double-quotes and escaping those characters with backslashes in the same way C escapes control characters (e.g. \t for TAB, \n for LF, \\ for backslash) or bytes with values larger than 0x80 (e.g. octal \302\265 for "micro" in UTF-8). If this variable is set to false, bytes higher than 0x80 are not considered "unusual" any more. Double-quotes, backslash and control characters are always escaped regardless of the setting of this variable. A simple space character is not considered "unusual". Many commands can output pathnames completely verbatim using the -z option. The default value is true.

core.eol

Sets the line ending type to use in the working directory for files that are marked as text (either by having the text attribute set, or by having text=auto and Git auto-detecting the contents as text). 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.

Note that this value is ignored if core.autocrlf is set to true or input.

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 the same as setting the text attribute to

"auto" on all files and core.eol to "crlf". Set to true if you want to have CRLF line endings in your working directory and the repository has LF line endings. This variable can be set to input, in which case no output conversion is performed.

core.checkRoundtripEncoding

A comma and/or whitespace separated list of encodings that Git performs UTF-8 round trip checks on if they are used in an working-tree-encoding attribute (see gitattributes(5)). The default value is SHIFT-JIS.

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.sshCommand

If this variable is set, git fetch and git push will use the specified command instead of ssh when they need to connect to a remote system. The command is in the same form as the GIT_SSH_COMMAND environment variable and is overridden when the environment variable is set.

core.ignoreStat

If true, Git will avoid using lstat() calls to detect if files have changed by setting the "assume-unchanged" bit for those tracked files which it has updated identically in both the index and working tree.

When files are modified outside of Git, the user will need to stage the modified files explicitly (e.g. see Examples section in git-update-index(1)). Git will not normally detect changes to those files.

This is useful on systems where lstat() calls are very slow, such as CIFS/Microsoft Windows.

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.alternateRefsCommand

When advertising tips of available history from an alternate, use the shell to execute the specified command instead of git-for-each-ref(1). The first argument is the absolute path of the alternate. Output must contain one hex object id per line (i.e., the same as produced by git for-each-ref --format='%(objectname)').

Note that you cannot generally put git for-each-ref directly into the config value, as it does not take a repository path as an argument (but you can wrap the command above in a shell script).

core.alternateRefsPrefixes

When listing references from an alternate, list only references that begin with the given prefix. Prefixes match as if they were given as arguments to git-for-each-ref(1). To list multiple prefixes, separate them with whitespace. If core.alternateRefsCommand is set, setting core.alternateRefsPrefixes has no effect.

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. If GIT_COMMON_DIR environment variable is set, core.worktree is ignored and not used for determining the root of 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 "$GIT_DIR/logs/<ref>", by appending the new and old SHA-1, the date/time and the reason of the update, but only when the file exists. If this configuration variable is set to true, missing "$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. If it is set to always, then a missing reflog is automatically created for any ref under refs/.

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 repository. 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 32 TiB (effectively unlimited) 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 96 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. Additionally files larger than this size are always treated as binary.

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

Specifies the pathname to the file that contains patterns to describe paths that are not meant to be tracked, in addition to .gitignore (per-directory) and .git/info/exclude. Defaults to $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.hooksPath

By default Git will look for your hooks in the $GIT_DIR/hooks directory. Set this to different path, e.g. /etc/git/hooks, and Git will try to find your hooks in that directory, e.g. /etc/git/hooks/pre-receive instead of in $GIT_DIR/hooks/pre-receive.

The path can be either absolute or relative. A relative path is taken as relative to the directory where the hooks are run (see the "DESCRIPTION" section of githooks(5)).

This configuration variable is useful in cases where you?d like to centrally configure your Git hooks instead of configuring them on a per-repository basis, or as a more flexible and centralized alternative to having an init.templateDir where you?ve changed default hooks.

core.editor

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

core.commentChar

Commands such as commit and tag that let you edit messages consider a line that begins with this character commented, and removes them after the editor returns (default #).

If set to "auto", git-commit would select a character that is not the beginning character of any line in existing commit messages.

core.filesRefLockTimeout

The length of time, in milliseconds, to retry when trying to lock an individual reference. Value 0 means not to retry at all; -1 means to try indefinitely.

Default is 100 (i.e., retry for 100ms).

core.packedRefsTimeout

The length of time, in milliseconds, to retry when trying to lock the packed-refs file. Value 0 means not to retry at all; -1 means to try indefinitely. Default is 1000 (i.e., retry for 1 second).

core.pager

Text viewer for use by Git commands (e.g., less). The value is meant to be interpreted by the shell. The order of preference is the $GIT_PAGER environment variable, then core.pager configuration, then $PAGER, and then the default chosen at compile time (usually less).

When the LESS environment variable is unset, Git sets it to FRX (if LESS environment variable is set, Git does not change it at all). If you want to selectively override Git?s default setting for LESS, you can set core.pager to e.g. less -S. This will be passed to the shell by Git, which will translate the final command to LESS=FRX less -S. The environment does not set the S option but the command line does, instructing less to truncate long lines. Similarly, setting core.pager to less -+F will deactivate the F option specified by the environment from the command-line, deactivating the "quit if one screen" behavior of less. One can specifically activate some flags for particular commands: for example, setting pager.blame to less -S enables line truncation only for git blame.

Likewise, when the LV environment variable is unset, Git sets it to -c. You can override this setting by exporting LV with another value or setting core.pager to lv +c.

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. When enabled, Git will do the index comparison to the filesystem data in parallel, allowing overlapping IO?s. Defaults to true.

core.unsetenvvars

Windows-only: comma-separated list of environment variables' names that need to be unset before spawning any other process. Defaults to PERL5LIB to account for the fact that Git for Windows insists on using its own Perl interpreter.

core.restrictinheritedhandles

Windows-only: override whether spawned processes inherit only standard file handles (stdin, stdout and stderr) or all handles. Can be auto, true or false. Defaults to auto, which means true on Windows 7 and later, and false on older Windows versions.

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.commitGraph

If true, then git will read the commit-graph file (if it exists) to parse the graph structure of commits. Defaults to true. See git-commit-graph(1) for more information.

core.useReplaceRefs

If set to false, behave as if the --no-replace-objects option was given on the

command line. See git(1) and git-replace(1) for more information.

core.multiPackIndex

Use the multi-pack-index file to track multiple packfiles using a single index. See the multi-pack-index design document[1].

core.sparseCheckout

Enable "sparse checkout" feature. See git-sparse-checkout(1) for more information.

core.sparseCheckoutCone

Enables the "cone mode" of the sparse checkout feature. When the sparse-checkout file contains a limited set of patterns, then this mode provides significant performance advantages. See git-sparse-checkout(1) for more information.

core.abbrev

Set the length object names are abbreviated to. If unspecified or set to "auto", an appropriate value is computed based on the approximate number of packed objects in your repository, which hopefully is enough for abbreviated object names to stay unique for some time. The minimum length is 4.

add.ignoreErrors, add.ignore-errors (deprecated)

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). add.ignore-errors is deprecated, as it does not follow the usual naming convention for configuration variables.

add.interactive.useBuiltin

[EXPERIMENTAL] Set to true to use the experimental built-in implementation of the interactive version of git-add(1) instead of the Perl script version. Is false by default.

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. A quote pair or a backslash can be used to quote them.

Note that the first word of an alias does not necessarily have to be a command. It can be a command-line option that will be passed into the invocation of git. In particular, this is useful when used with -c to pass in one-time configurations or -p to force pagination. For example, loud-rebase = -c commit.verbose=true rebase can be defined such that running git loud-rebase would be equivalent to git -c commit.verbose=true rebase. Also, ps = -p status would be a helpful alias since git ps would paginate the output of git status where the original command does not.

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).

am.threeWay

By default, git am will fail if the patch does not apply cleanly. When set to true, this setting tells git am to fall back on 3-way merge if the patch records the identity of blobs it is supposed to apply to and we have those blobs available locally (equivalent to giving the --3way option from the command line). Defaults to false. See git-am(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).

blame.blankBoundary

Show blank commit object name for boundary commits in git-blame(1). This option defaults to false.

blame.coloring

This determines the coloring scheme to be applied to blame output. It can be repeatedLines, highlightRecent, or none which is the default.

blame.date

Specifies the format used to output dates in git-blame(1). If unset the iso format is used. For supported values, see the discussion of the --date option at git-log(1).

blame.showEmail

Show the author email instead of author name in git-blame(1). This option defaults to false.

blame.showRoot

Do not treat root commits as boundaries in git-blame(1). This option defaults to false.

blame.ignoreRevsFile

Ignore revisions listed in the file, one unabbreviated object name per line, in git-blame(1). Whitespace and comments beginning with # are ignored. This option may be repeated multiple times. Empty file names will reset the list of ignored revisions. This option will be handled before the command line option --ignore-revs-file.

blame.markUnblamables

Mark lines that were changed by an ignored revision that we could not attribute to another commit with a * in the output of git-blame(1).

blame.markIgnoredLines

Mark lines that were changed by an ignored revision that we attributed to another commit with a ? in the output of git-blame(1).

branch.autoSetupMerge

Tells git branch, git switch 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, git switch 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.sort

This variable controls the sort ordering of branches when displayed by git-branch(1). Without the "--sort=<value>" option provided, the value of this variable will be used as the default. See git-for-each-ref(1) field names for valid values.

branch.<name>.remote

When on branch <name>, it tells git fetch and git push which remote to fetch from/push to. The remote to push to may be overridden with remote.pushDefault (for all branches). The remote to push to, for the current branch, may be further overridden by branch.<name>.pushRemote. If no remote is configured, or if you are not on any branch, it defaults to origin for fetching and remote.pushDefault for pushing. Additionally, . (a period) is the current local repository (a dot-repository), see branch.<name>.merge's final note below.

branch.<name>.pushRemote

When on branch <name>, it overrides branch.<name>.remote for pushing. It also overrides remote.pushDefault for pushing from branch <name>. When you pull from one place (e.g. your upstream) and push to another place (e.g. your own publishing repository), you would want to set remote.pushDefault to specify the

remote to push to for all branches, and use this option to override it for a specific 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 relative path 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.

When merges, pass the --rebase-merges option to git rebase so that the local merge commits are included in the rebase (see git-rebase(1) for details).

When preserve (deprecated in favor of merges), also pass --preserve-merges along to git rebase so that locally committed merge commits will not be flattened by running git pull.

When the value is interactive, the rebase is run in interactive mode.

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

branch.<name>.description

Branch description, can be edited with git branch --edit-description. Branch

description is automatically added in the format-patch cover letter or request-pull summary.

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)).

checkout.defaultRemote

When you run git checkout <something> or git switch <something> and only have one remote, it may implicitly fall back on checking out and tracking e.g. origin/<something>. This stops working as soon as you have more than one remote with a <something> reference. This setting allows for setting the name of a preferred remote that should always win when it comes to disambiguation. The typical use-case is to set this to origin.

Currently this is used by git-switch(1) and git-checkout(1) when git checkout <something> or git switch <something> will checkout the <something> branch on another remote, and by git-worktree(1) when git worktree add refers to a remote branch. This setting might be used for other checkout-like commands or functionality in the future.

clean.requireForce

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

color.advice

A boolean to enable/disable color in hints (e.g. when a push failed, see advice.* for a list). May be set to always, false (or never) or auto (or true), in which case colors are used only when the error output goes to a terminal. If unset, then the value of color.ui is used (auto by default).

color.advice.hint

Use customized color for hints.

color.blame.highlightRecent

This can be used to color the metadata of a blame line depending on age of the

line.

This setting should be set to a comma-separated list of color and date settings, starting and ending with a color, the dates should be set from oldest to newest. The metadata will be colored given the colors if the line was introduced before the given timestamp, overwriting older timestamped colors.

Instead of an absolute timestamp relative timestamps work as well, e.g. 2.weeks.ago is valid to address anything older than 2 weeks.

It defaults to blue,12 month ago,white,1 month ago,red, which colors everything older than one year blue, recent changes between one month and one year old are kept white, and lines introduced within the last month are colored red.

color.blame.repeatedLines

Use the customized color for the part of git-blame output that is repeated meta information per line (such as commit id, author name, date and timezone). Defaults to cyan.

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. If unset, then the value of color.ui is used (auto by default).

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/), upstream (upstream tracking branch), plain (other refs).

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. If unset, then the value of color.ui is used (auto by default).

This does not affect git-format-patch(1) or 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 context (context text - plain is a historical synonym), meta (metainformation), frag (hunk header), func (function in hunk header), old (removed lines), new (added lines), commit (commit headers), whitespace (highlighting whitespace errors), oldMoved (deleted lines), newMoved (added lines), oldMovedDimmed, oldMovedAlternative, oldMovedAlternativeDimmed, newMovedDimmed, newMovedAlternative newMovedAlternativeDimmed (See the <mode> setting of --color-moved in git-diff(1) for details), contextDimmed, oldDimmed, newDimmed, contextBold, oldBold, and newBold (see git-range-diff(1) for details).

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 and grafted for grafted commits.

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. If unset, then the value of color.ui is used (auto by default).

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)

column

column number prefix (when using --column)

match

matching text (same as setting matchContext and matchSelected)

matchContext

matching text in context lines

matchSelected

matching text in selected lines

selected

non-matching text in selected lines

separator

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

color.interactive

When set to always, always use colors for interactive prompts and displays (such as those used by "git-add --interactive" and "git-clean --interactive"). When false (or never), never. When set to true or auto, use colors only when the output is to the terminal. If unset, then the value of color.ui is used (auto by default).

color.interactive.<slot>

Use customized color for git add --interactive and git clean --interactive output. <slot> may be prompt, header, help or error, for four distinct types of normal output from interactive commands.

color.pager

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

color.push

A boolean to enable/disable color in push errors. May be set to always, false (or never) or auto (or true), in which case colors are used only when the error output goes to a terminal. If unset, then the value of color.ui is used (auto by default).

color.push.error

Use customized color for push errors.

color.remote

If set, keywords at the start of the line are highlighted. The keywords are "error", "warning", "hint" and "success", and are matched case-insensitively. May be set to always, false (or never) or auto (or true). If unset, then the value of color.ui is used (auto by default).

color.remote.<slot>

Use customized color for each remote keyword. <slot> may be hint, warning, success or error which match the corresponding keyword.

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. If unset, then the value of color.ui is used (auto by default).

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. If unset, then the value of color.ui is used (auto by default).

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), nobranch (the color the no branch warning is shown in, defaulting to red), localBranch or remoteBranch (the local and remote branch names, respectively, when branch and tracking information is displayed in the status short-format), or unmerged (files which have unmerged changes).

color.transport

A boolean to enable/disable color when pushes are rejected. May be set to always, false (or never) or auto (or true), in which case colors are used only when the error output goes to a terminal. If unset, then the value of color.ui is used (auto by default).

color.transport.rejected

Use customized color when a push was rejected.

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 false or never if you prefer Git commands not to use color

unless enabled explicitly with some other configuration or the --color option.

Set it to always if you want all output not intended for machine consumption to use color, to true or auto (this is the default since Git 1.8.4) if you want such output to use color when written to the terminal.

column.ui

Specify whether supported commands should output in columns. This variable consists of a list of tokens separated by spaces or commas:

These options control when the feature should be enabled (defaults to never):

always

always show in columns

never

never show in columns

auto

show in columns if the output is to the terminal

These options control layout (defaults to column). Setting any of these implies always if none of always, never, or auto are specified.

column

fill columns before rows

row

fill rows before columns

plain

show in one column

Finally, these options can be combined with a layout option (defaults to nodense):

dense

make unequal size columns to utilize more space

nodense

make equal size columns

column.branch

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

column.clean

Specify the layout when list items in git clean -i, which always shows files

and directories 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.cleanup

This setting overrides the default of the --cleanup option in git commit. See git-commit(1) for details. Changing the default can be useful when you always want to keep lines that begin with comment character # in your log message, in which case you would do git config commit.cleanup whitespace (note that you will have to remove the help lines that begin with # in the commit log template yourself, if you do this).

commit.gpgSign

A boolean to specify whether all commits should be GPG signed. Use of this option when doing operations such as rebase can result in a large number of commits being signed. It may be convenient to use an agent to avoid typing your GPG passphrase several times.

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 the pathname of a file to use as the template for new commit messages.

commit.verbose

A boolean or int to specify the level of verbose with git commit. See git-commit(1).

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. Note that multiple helpers may be defined. 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.

credentialCache.ignoreSIGHUP

Tell git-credential-cache?daemon to ignore SIGHUP, instead of quitting.

completion.commands

This is only used by git-completion.bash to add or remove commands from the list of completed commands. Normally only porcelain commands and a few select others are completed. You can add more commands, separated by space, in this variable. Prefixing the command with - will remove it from the existing list.

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.interHunkContext

Show the context between diff hunks, up to the specified number of lines, thereby fusing the hunks that are close to each other. This value serves as the default for the --inter-hunk-context command line 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 and git switch also honor this setting when reporting uncommitted changes. Setting it to all disables the submodule summary normally shown by git commit and git status when status.submoduleSummary is set unless it is overridden by using the --ignore-submodules command-line option. The git submodule commands are not affected by this setting.

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.orderFile

File indicating how to order files within a diff. See the -O option to git-diff(1) for details. If diff.orderFile is a relative pathname, it is treated as relative to the top of the working tree.

diff.renameLimit

The number of files to consider when performing the copy/rename detection; equivalent to the git diff option -l. This setting has no effect if rename detection is turned off.

diff.renames

Whether and how Git detects renames. If set to "false", rename detection is disabled. If set to "true", basic rename detection is enabled. If set to "copies" or "copy", Git will detect copies, as well. Defaults to true. Note that this affects only git diff Porcelain like git-diff(1) and git-log(1), and not lower level commands such as git-diff-files(1).

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 "short" format just shows the names of the commits at the beginning and end of the range. The "log" format lists the commits in the range like git-submodule(1) summary does. The "diff" format shows an inline diff of the changed contents of the submodule. 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

Controls which diff tool is used by git-difftool(1). This variable overrides the value configured in merge.tool. The list below shows the valid built-in values. Any other value is treated as a custom diff tool and requires that a corresponding difftool.<tool>.cmd variable is defined.

diff.guitool

Controls which diff tool is used by git-difftool(1) when the -g/--gui flag is specified. This variable overrides the value configured in merge.guitool. The list below shows the valid built-in values. Any other value is treated as a custom diff tool and requires that a corresponding difftool.<guitool>.cmd variable is defined.

- araxis
- bc
- bc3
- codecompare

? deltawalker

? diffmerge

? diffuse

? ecmerge

? emerge

? examdiff

? guiffy

? gvimdiff

? gvimdiff2

? gvimdiff3

? kdiff3

? kompare

? meld

? opendiff

? p4merge

? smerge

? tkdiff

? vimdiff

? vimdiff2

? vimdiff3

? winmerge

? xxdiff

diff.indentHeuristic

Set this option to false to disable the default heuristics that shift diff hunk boundaries to make patches easier to read.

diff.algorithm

Choose a diff algorithm. The variants are as follows:

default, myers

The basic greedy diff algorithm. Currently, this is the default.

minimal

Spend extra time to make sure the smallest possible diff is produced.

patience

Use "patience diff" algorithm when generating patches.

histogram

This algorithm extends the patience algorithm to "support low-occurrence common elements".

diff.wsErrorHighlight

Highlight whitespace errors in the context, old or new lines of the diff. Multiple values are separated by comma, none resets previous values, default reset the list to new and all is a shorthand for old,new,context. The whitespace errors are colored with color.diff.whitespace. The command line option --ws-error-highlight=<kind> overrides this setting.

diff.colorMoved

If set to either a valid <mode> or a true value, moved lines in a diff are colored differently, for details of valid modes see --color-moved in git-diff(1). If simply set to true the default color mode will be used. When set to false, moved lines are not colored.

diff.colorMovedWS

When moved lines are colored using e.g. the diff.colorMoved setting, this option controls the <mode> how spaces are treated for details of valid modes see --color-moved-ws in git-diff(1).

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.

fastimport.unpackLimit

If the number of objects imported by git-fast-import(1) is below this limit, then the objects will be unpacked into loose object files. However if the number of imported objects equals or exceeds this limit then the pack will be

stored as a pack. Storing the pack from a fast-import can make the import operation complete faster, especially on slow filesystems. If not set, the value of transfer.unpackLimit is used instead.

feature.*

The config settings that start with feature. modify the defaults of a group of other config settings. These groups are created by the Git developer community as recommended defaults and are subject to change. In particular, new config options may be added with different defaults.

feature.experimental

Enable config options that are new to Git, and are being considered for future defaults. Config settings included here may be added or removed with each release, including minor version updates. These settings may have unintended interactions since they are so new. Please enable this setting if you are interested in providing feedback on experimental features. The new default values are:

- pack.useSparse=true uses a new algorithm when constructing a pack-file which can improve git push performance in repos with many files.
- fetch.negotiationAlgorithm=skipping may improve fetch negotiation times by skipping more commits at a time, reducing the number of round trips.
- fetch.writeCommitGraph=true writes a commit-graph after every git fetch command that downloads a pack-file from a remote. Using the --split option, most executions will create a very small commit-graph file on top of the existing commit-graph file(s). Occasionally, these files will merge and the write may take longer. Having an updated commit-graph file helps performance of many Git commands, including git merge-base, git push -f, and git log --graph.

feature.manyFiles

Enable config options that optimize for repos with many files in the working directory. With many files, commands such as git status and git checkout may be slow and these new defaults improve performance:

- index.version=4 enables path-prefix compression in the index.
- core.untrackedCache=true enables the untracked cache. This setting assumes that mtime is working on your machine.

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. See transfer.fsckObjects for what?s checked. Defaults to false. If not set, the value of transfer.fsckObjects is used instead.

fetch.fsck.<msg-id>

Acts like fsck.<msg-id>, but is used by git-fetch-pack(1) instead of git-fsck(1). See the fsck.<msg-id> documentation for details.

fetch.fsck.skipList

Acts like fsck.skipList, but is used by git-fetch-pack(1) instead of git-fsck(1). See the fsck.skipList documentation for details.

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.

fetch.prune

If true, fetch will automatically behave as if the --prune option was given on the command line. See also remote.<name>.prune and the PRUNING section of git-fetch(1).

fetch.pruneTags

If true, fetch will automatically behave as if the refs/tags/*:refs/tags/* refspec was provided when pruning, if not set already. This allows for setting both this option and fetch.prune to maintain a 1=1 mapping to upstream refs.

See also remote.<name>.pruneTags and the PRUNING section of git-fetch(1).

fetch.output

Control how ref update status is printed. Valid values are full and compact.

Default value is full. See section OUTPUT in git-fetch(1) for detail.

fetch.negotiationAlgorithm

Control how information about the commits in the local repository is sent when negotiating the contents of the packfile to be sent by the server. Set to "skipping" to use an algorithm that skips commits in an effort to converge faster, but may result in a larger-than-necessary packfile; The default is "default" which instructs Git to use the default algorithm that never skips commits (unless the server has acknowledged it or one of its descendants). If feature.experimental is enabled, then this setting defaults to "skipping".

Unknown values will cause git fetch to error out.

See also the --negotiation-tip option for git-fetch(1).

fetch.showForcedUpdates

Set to false to enable --no-show-forced-updates in git-fetch(1) and git-pull(1) commands. Defaults to true.

fetch.parallel

Specifies the maximal number of fetch operations to be run in parallel at a time (submodules, or remotes when the --multiple option of git-fetch(1) is in effect).

A value of 0 will give some reasonable default. If unset, it defaults to 1.

For submodules, this setting can be overridden using the submodule.fetchJobs config setting.

fetch.writeCommitGraph

Set to true to write a commit-graph after every git fetch command that downloads a pack-file from a remote. Using the --split option, most executions will create a very small commit-graph file on top of the existing commit-graph file(s). Occasionally, these files will merge and the write may take longer.

Having an updated commit-graph file helps performance of many Git commands, including git merge-base, git push -f, and git log --graph. Defaults to false, unless feature.experimental is true.

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.from

Provides the default value for the --from option to format-patch. Accepts a boolean value, or a name and email address. If false, format-patch defaults to --no-from, using commit authors directly in the "From:" field of patch mails. If true, format-patch defaults to --from, using your committer identity in the "From:" field of patch mails and including a "From:" field in the body of the patch mail if different. If set to a non-boolean value, format-patch uses that value instead of your committer identity. Defaults to false.

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.coverFromDescription

The default mode for format-patch to determine which parts of the cover letter will be populated using the branch?s description. See the --cover-from-description option in git-format-patch(1).

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.signatureFile

Works just like format.signature except the contents of the file specified by this variable will be used as the signature.

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.

format.coverLetter

A boolean that controls whether to generate a cover-letter when format-patch is invoked, but in addition can be set to "auto", to generate a cover-letter only when there?s more than one patch. Default is false.

format.outputDirectory

Set a custom directory to store the resulting files instead of the current working directory. All directory components will be created.

format.useAutoBase

A boolean value which lets you enable the --base=auto option of format-patch by

default.

format.notes

Provides the default value for the --notes option to format-patch. Accepts a boolean value, or a ref which specifies where to get notes. If false, format-patch defaults to --no-notes. If true, format-patch defaults to --notes. If set to a non-boolean value, format-patch defaults to --notes=<ref>, where ref is the non-boolean value. Defaults to false.

If one wishes to use the ref ref/notes/true, please use that literal instead.

This configuration can be specified multiple times in order to allow multiple notes refs to be included. In that case, it will behave similarly to multiple --[no-]notes[=] options passed in. That is, a value of true will show the default notes, a value of <ref> will also show notes from that notes ref and a value of false will negate previous configurations and not show notes.

For example,

```
[format]
        notes = true
        notes = foo
        notes = false
        notes = bar
```

will only show notes from refs/notes/bar.

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.

fsck.<msg-id>

During fsck git may find issues with legacy data which wouldn?t be generated by current versions of git, and which wouldn?t be sent over the wire if transfer.fsckObjects was set. This feature is intended to support working with legacy repositories containing such data.

Setting fsck.<msg-id> will be picked up by git-fsck(1), but to accept pushes of such data set receive.fsck.<msg-id> instead, or to clone or fetch it set

fetch.fsck.<msg-id>.

The rest of the documentation discusses fsck.* for brevity, but the same applies for the corresponding receive.fsck.* and fetch.<msg-id>.*. variables. Unlike variables like color.ui and core.editor the receive.fsck.<msg-id> and fetch.fsck.<msg-id> variables will not fall back on the fsck.<msg-id> configuration if they aren?t set. To uniformly configure the same fsck settings in different circumstances all three of them they must all set to the same values.

When fsck.<msg-id> is set, errors can be switched to warnings and vice versa by configuring the fsck.<msg-id> setting where the <msg-id> is the fsck message ID and the value is one of error, warn or ignore. For convenience, fsck prefixes the error/warning with the message ID, e.g. "missingEmail: invalid author/committer line - missing email" means that setting fsck.missingEmail = ignore will hide that issue.

In general, it is better to enumerate existing objects with problems with fsck.skipList, instead of listing the kind of breakages these problematic objects share to be ignored, as doing the latter will allow new instances of the same breakages go unnoticed.

Setting an unknown fsck.<msg-id> value will cause fsck to die, but doing the same for receive.fsck.<msg-id> and fetch.fsck.<msg-id> will only cause git to warn.

fsck.skipList

The path to a list of object names (i.e. one unabbreviated SHA-1 per line) that are known to be broken in a non-fatal way and should be ignored. On versions of Git 2.20 and later comments (#), empty lines, and any leading and trailing whitespace is ignored. Everything but a SHA-1 per line will error out on older versions.

This feature is useful when an established project should be accepted despite early commits containing errors that can be safely ignored such as invalid committer email addresses. Note: corrupt objects cannot be skipped with this setting.

Like fsck.<msg-id> this variable has corresponding receive.fsck.skipList and fetch.fsck.skipList variants.

Unlike variables like color.ui and core.editor the receive.fsck.skipList and fetch.fsck.skipList variables will not fall back on the fsck.skipList configuration if they aren?t set. To uniformly configure the same fsck settings in different circumstances all three of them they must all set to the same values.

Older versions of Git (before 2.20) documented that the object names list should be sorted. This was never a requirement, the object names could appear in any order, but when reading the list we tracked whether the list was sorted for the purposes of an internal binary search implementation, which could save itself some work with an already sorted list. Unless you had a humongous list there was no reason to go out of your way to pre-sort the list. After Git version 2.20 a hash implementation is used instead, so there?s now no reason to pre-sort the list.

gc.aggressiveDepth

The depth parameter used in the delta compression algorithm used by git gc --aggressive. This defaults to 50, which is the default for the --depth option when --aggressive isn?t in use.

See the documentation for the --depth option in git-repack(1) for more details.

gc.aggressiveWindow

The window size parameter used in the delta compression algorithm used by git gc --aggressive. This defaults to 250, which is a much more aggressive window size than the default --window of 10.

See the documentation for the --window option in git-repack(1) for more details.

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 not only automatic packing based on the number of loose objects, but any other heuristic git gc --auto will otherwise use to determine if there?s work to do, such as gc.autoPackLimit.

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. Setting gc.auto to 0 will also disable this.

See the gc.bigPackThreshold configuration variable below. When in use, it?ll affect how the auto pack limit works.

gc.autoDetach

Make git gc --auto return immediately and run in background if the system supports it. Default is true.

gc.bigPackThreshold

If non-zero, all packs larger than this limit are kept when git gc is run. This is very similar to --keep-base-pack except that all packs that meet the threshold are kept, not just the base pack. Defaults to zero. Common unit suffixes of k, m, or g are supported.

Note that if the number of kept packs is more than gc.autoPackLimit, this configuration variable is ignored, all packs except the base pack will be repacked. After this the number of packs should go below gc.autoPackLimit and gc.bigPackThreshold should be respected again.

If the amount of memory estimated for git repack to run smoothly is not available and gc.bigPackThreshold is not set, the largest pack will also be excluded (this is the equivalent of running git gc with --keep-base-pack).

gc.writeCommitGraph

If true, then gc will rewrite the commit-graph file when git-gc(1) is run. When using git gc --auto the commit-graph will be updated if housekeeping is required. Default is true. See git-commit-graph(1) for details.

gc.logExpiry

If the file gc.log exists, then git gc --auto will print its content and exit with status zero instead of running unless that file is more than gc.logExpiry old. Default is "1.day". See gc.pruneExpire for more ways to specify its value.

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, or "never" may be used to suppress pruning. This feature helps prevent corruption when git gc runs concurrently with another process writing to the repository; see the "NOTES" section of git-gc(1).

gc.worktreePruneExpire

When git gc is run, it calls git worktree prune --expire 3.months.ago. This config variable can be used to set a different grace period. The value "now" may be used to disable the grace period and prune $GIT_DIR/worktrees immediately, or "never" may be used to suppress pruning.

gc.reflogExpire, gc.<pattern>.reflogExpire

git reflog expire removes reflog entries older than this time; defaults to 90 days. The value "now" expires all entries immediately, and "never" suppresses expiration altogether. With "<pattern>" (e.g. "refs/stash") in the middle the setting applies only to the refs that match the <pattern>.

gc.reflogExpireUnreachable, gc.<pattern>.reflogExpireUnreachable

git reflog expire removes reflog entries older than this time and are not reachable from the current tip; defaults to 30 days. The value "now" expires all entries immediately, and "never" suppresses expiration altogether. With "<pattern>" (e.g. "refs/stash") in the middle, the setting applies only to the refs that match the <pattern>.

These types of entries are generally created as a result of using git commit --amend or git rebase and are the commits prior to the amend or rebase occurring. Since these changes are not part of the current project most users will want to expire them sooner, which is why the default is more aggressive than gc.reflogExpire.

gc.rerereResolved

Records of conflicted merge you resolved earlier are kept for this many days when git rerere gc is run. You can also use more human-readable "1.month.ago",

etc. 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. You can also use more human-readable "1.month.ago", etc. 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.column

If set to true, enable the --column 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.

grep.threads

Number of grep worker threads to use. See grep.threads in git-grep(1) for more information.

grep.fallbackToNoIndex

If set to true, fall back to git grep --no-index if git grep is executed outside of a git repository. Defaults to false.

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 $signature - <$file" 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.

gpg.format

Specifies which key format to use when signing with --gpg-sign. Default is "openpgp" and another possible value is "x509".

gpg.<format>.program

Use this to customize the program used for the signing format you chose. (see gpg.program and gpg.format) gpg.program can still be used as a legacy synonym for gpg.openpgp.program. The default value for gpg.x509.program is "gpgsm".

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.displayUntracked

Determines if git-gui(1) shows untracked files in the file list. The default is

"true".

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)). In addition to the syntax understood by curl, it is possible to specify a proxy string with a user name but no password, in which case git will attempt to acquire one in the same way it does for other credentials. See gitcredentials(7) for more information. The syntax thus is [protocol://][user[:password]@]proxyhost[:port]. This can be overridden on a per-remote basis; see remote.<name>.proxy

http.proxyAuthMethod

Set the method with which to authenticate against the HTTP proxy. This only takes effect if the configured proxy string contains a user name part (i.e. is of the form user@host or user@host:port). This can be overridden on a per-remote basis; see remote.<name>.proxyAuthMethod. Both can be overridden by the GIT_HTTP_PROXY_AUTHMETHOD environment variable. Possible values are:

? anyauth - Automatically pick a suitable authentication method. It is assumed that the proxy answers an unauthenticated request with a 407 status code and one or more Proxy-authenticate headers with supported authentication methods. This is the default.

? basic - HTTP Basic authentication

? digest - HTTP Digest authentication; this prevents the password from being transmitted to the proxy in clear text

? negotiate - GSS-Negotiate authentication (compare the --negotiate option of curl(1))

? ntlm - NTLM authentication (compare the --ntlm option of curl(1))

http.emptyAuth

Attempt authentication without seeking a username or password. This can be used to attempt GSS-Negotiate authentication without specifying a username in the URL, as libcurl normally requires a username for authentication.

http.delegation

Control GSSAPI credential delegation. The delegation is disabled by default in libcurl since version 7.21.7. Set parameter to tell the server what it is allowed to delegate when it comes to user credentials. Used with GSS/kerberos. Possible values are:

? none - Don?t allow any delegation.

? policy - Delegates if and only if the OK-AS-DELEGATE flag is set in the Kerberos service ticket, which is a matter of realm policy.

? always - Unconditionally allow the server to delegate.

http.extraHeader

Pass an additional HTTP header when communicating with a server. If more than one such entry exists, all of them are added as extra headers. To allow overriding the settings inherited from the system config, an empty value will reset the extra headers to the empty list.

http.cookieFile

The pathname of a 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 used only as input unless http.saveCookies is set.

http.saveCookies

If set, store cookies received during requests to the file specified by http.cookieFile. Has no effect if http.cookieFile is unset.

http.version

Use the specified HTTP protocol version when communicating with a server. If you want to force the default. The available and default version depend on libcurl. Currently the possible values of this option are:

? HTTP/2

? HTTP/1.1

http.sslVersion

The SSL version to use when negotiating an SSL connection, if you want to force the default. The available and default version depend on whether libcurl was built against NSS or OpenSSL and the particular configuration of the crypto library in use. Internally this sets the CURLOPT_SSL_VERSION option; see the libcurl documentation for more details on the format of this option and for the ssl version supported. Currently the possible values of this option are:

? sslv2

? sslv3

? tlsv1

? tlsv1.0

? tlsv1.1

? tlsv1.2

? tlsv1.3

Can be overridden by the GIT_SSL_VERSION environment variable. To force git to use libcurl?s default ssl version and ignore any explicit http.sslversion option, set GIT_SSL_VERSION to the empty string.

http.sslCipherList

A list of SSL ciphers to use when negotiating an SSL connection. The available ciphers depend on whether libcurl was built against NSS or OpenSSL and the particular configuration of the crypto library in use. Internally this sets the CURLOPT_SSL_CIPHER_LIST option; see the libcurl documentation for more details on the format of this list.

Can be overridden by the GIT_SSL_CIPHER_LIST environment variable. To force git to use libcurl?s default cipher list and ignore any explicit http.sslCipherList option, set GIT_SSL_CIPHER_LIST to the empty string.

http.sslVerify

Whether to verify the SSL certificate when fetching or pushing over HTTPS. Defaults to true. 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.sslBackend

Name of the SSL backend to use (e.g. "openssl" or "schannel"). This option is ignored if cURL lacks support for choosing the SSL backend at runtime.

http.schannelCheckRevoke

Used to enforce or disable certificate revocation checks in cURL when http.sslBackend is set to "schannel". Defaults to true if unset. Only necessary to disable this if Git consistently errors and the message is about checking

the revocation status of a certificate. This option is ignored if cURL lacks support for setting the relevant SSL option at runtime.

http.schannelUseSSLCAInfo

As of cURL v7.60.0, the Secure Channel backend can use the certificate bundle provided via http.sslCAInfo, but that would override the Windows Certificate Store. Since this is not desirable by default, Git will tell cURL not to use that bundle by default when the schannel backend was configured via http.sslBackend, unless http.schannelUseSSLCAInfo overrides this behavior.

http.pinnedpubkey

Public key of the https service. It may either be the filename of a PEM or DER encoded public key file or a string starting with sha256// followed by the base64 encoded sha256 hash of the public key. See also libcurl CURLOPT_PINNEDPUBLICKEY. git will exit with an error if this option is set but not supported by cURL.

http.sslTry

Attempt to use AUTH SSL/TLS and encrypted data transfers when connecting via regular FTP protocol. This might be needed if the FTP server requires it for security reasons or you wish to connect securely whenever remote FTP server supports it. Default is false since it might trigger certificate verification errors on misconfigured servers.

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.

Note that raising this limit is only effective for disabling chunked transfer encoding and therefore should be used only where the remote server or a proxy only supports HTTP/1.0 or is noncompliant with the HTTP standard. Raising this is not, in general, an effective solution for most push problems, but can increase memory consumption significantly since the entire buffer is allocated even for small pushes.

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.

http.followRedirects

Whether git should follow HTTP redirects. If set to true, git will transparently follow any redirect issued by a server it encounters. If set to false, git will treat all redirects as errors. If set to initial, git will follow redirects only for the initial request to a remote, but not for subsequent follow-up HTTP requests. Since git uses the redirected URL as the base for the follow-up requests, this is generally sufficient. The default is initial.

http.<url>.*

Any of the http.* options above can be applied selectively to some URLs. For a

config key to match a URL, each element of the config key is compared to that of the URL, in the following order:

1. Scheme (e.g., https in https://example.com/). This field must match exactly between the config key and the URL.
2. Host/domain name (e.g., example.com in https://example.com/). This field must match between the config key and the URL. It is possible to specify a * as part of the host name to match all subdomains at this level. https://*.example.com/ for example would match https://foo.example.com/, but not https://foo.bar.example.com/.
3. Port number (e.g., 8080 in http://example.com:8080/). This field must match exactly between the config key and the URL. Omitted port numbers are automatically converted to the correct default for the scheme before matching.
4. Path (e.g., repo.git in https://example.com/repo.git). The path field of the config key must match the path field of the URL either exactly or as a prefix of slash-delimited path elements. This means a config key with path foo/ matches URL path foo/bar. A prefix can only match on a slash (/) boundary. Longer matches take precedence (so a config key with path foo/bar is a better match to URL path foo/bar than a config key with just path foo/).
5. User name (e.g., user in https://user@example.com/repo.git). If the config key has a user name it must match the user name in the URL exactly. If the config key does not have a user name, that config key will match a URL with any user name (including none), but at a lower precedence than a config key with a user name.

The list above is ordered by decreasing precedence; a URL that matches a config key?s path is preferred to one that matches its user name. For example, if the URL is https://user@example.com/foo/bar a config key match of https://example.com/foo will be preferred over a config key match of https://user@example.com.

All URLs are normalized before attempting any matching (the password part, if embedded in the URL, is always ignored for matching purposes) so that equivalent URLs that are simply spelled differently will match properly.

Environment variable settings always override any matches. The URLs that are matched against are those given directly to Git commands. This means any URLs visited as a result of a redirection do not participate in matching.

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.folder

The folder to drop the mails into, which is typically the Drafts folder. For example: "INBOX.Drafts", "INBOX/Drafts" or "[Gmail]/Drafts". Required.

imap.tunnel

Command used to setup a tunnel to the IMAP server through which commands will be piped instead of using a direct network connection to the server. Required when imap.host is not set.

imap.host

A URL identifying the server. Use an imap:// prefix for non-secure connections and an imaps:// prefix for secure connections. Ignored when imap.tunnel is set, but required otherwise.

imap.user

The username to use when logging in to the server.

imap.pass

The password to use when logging in to the server.

imap.port

An integer port number to connect to on the server. Defaults to 143 for imap:// hosts and 993 for imaps:// hosts. Ignored when imap.tunnel is set.

imap.sslverify

A boolean to enable/disable verification of the server certificate used by the SSL/TLS connection. Default is true. Ignored when imap.tunnel is set.

imap.preformattedHTML

A boolean to enable/disable the use of html encoding when sending a patch. An html encoded patch will be bracketed with <pre> and have a content type of text/html. Ironically, enabling this option causes Thunderbird to send the patch as a plain/text, format=fixed email. Default is false.

imap.authMethod

Specify authenticate method for authentication with IMAP server. If Git was built with the NO_CURL option, or if your curl version is older than 7.34.0, or if you?re running git-imap-send with the --no-curl option, the only supported method is CRAM-MD5. If this is not set then git imap-send uses the basic IMAP plaintext LOGIN command.

index.recordEndOfIndexEntries

Specifies whether the index file should include an "End Of Index Entry" section. This reduces index load time on multiprocessor machines but produces a message "ignoring EOIE extension" when reading the index using Git versions before 2.20. Defaults to true if index.threads has been explicitly enabled, false otherwise.

index.recordOffsetTable

Specifies whether the index file should include an "Index Entry Offset Table" section. This reduces index load time on multiprocessor machines but produces a message "ignoring IEOT extension" when reading the index using Git versions before 2.20. Defaults to true if index.threads has been explicitly enabled, false otherwise.

index.threads

Specifies the number of threads to spawn when loading the index. This is meant to reduce index load time on multiprocessor machines. Specifying 0 or true will cause Git to auto-detect the number of CPU?s and set the number of threads accordingly. Specifying 1 or false will disable multithreading. Defaults to true.

index.version

Specify the version with which new index files should be initialized. This does not affect existing repositories. If feature.manyFiles is enabled, then the default is 4.

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-restore(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; requires the Perl module Term::ReadKey.

interactive.diffFilter

When an interactive command (such as git add --patch) shows a colorized diff, git will pipe the diff through the shell command defined by this configuration variable. The command may mark up the diff further for human consumption, provided that it retains a one-to-one correspondence with the lines in the original diff. Defaults to disabled (no filtering).

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. 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. If auto is specified, then if the output is going to a terminal, the ref names are shown as if short were given, otherwise no ref names are shown. This is the same as the --decorate option of the git log.

log.follow

If true, git log will act as if the --follow option was used when a single <path> is given. This has the same limitations as --follow, i.e. it cannot be used to follow multiple files and does not work well on non-linear history.

log.graphColors

A list of colors, separated by commas, that can be used to draw history lines in git log --graph.

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.

log.showSignature

If true, makes git-log(1), git-show(1), and git-whatchanged(1) assume --show-signature.

log.mailmap

If true, makes git-log(1), git-show(1), and git-whatchanged(1) assume --use-mailmap, otherwise assume --no-use-mailmap. True by default.

mailinfo.scissors

If true, makes git-mailinfo(1) (and therefore git-am(1)) act by default as if the --scissors option was provided on the command-line. When active, this

features removes everything from the message body before a scissors line (i.e. consisting mainly of ">8", "8<" and "-").

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).

mailmap.blob

Like mailmap.file, but consider the value as a reference to a blob in the repository. If both mailmap.file and mailmap.blob are given, both are parsed, with entries from mailmap.file taking precedence. In a bare repository, this defaults to HEAD:.mailmap. In a non-bare repository, it defaults to empty.

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.verifySignatures

If true, this is equivalent to the --verify-signatures command line option. See git-merge(1) for details.

merge.branchdesc

In addition to branch names, populate the log message with the branch description text associated with them. Defaults to false.

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. This setting has no effect if rename detection is turned off.

merge.renames

Whether Git detects renames. If set to "false", rename detection is disabled. If set to "true", basic rename detection is enabled. Defaults to the value of diff.renames.

merge.directoryRenames

Whether Git detects directory renames, affecting what happens at merge time to new files added to a directory on one side of history when that directory was renamed on the other side of history. If merge.directoryRenames is set to "false", directory rename detection is disabled, meaning that such new files

will be left behind in the old directory. If set to "true", directory rename detection is enabled, meaning that such new files will be moved into the new directory. If set to "conflict", a conflict will be reported for such paths. If merge.renames is false, merge.directoryRenames is ignored and treated as false. Defaults to "conflict".

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 tool is used by git-mergetool(1). The list below shows the valid built-in values. Any other value is treated as a custom merge tool and requires that a corresponding mergetool.<tool>.cmd variable is defined.

merge.guitool

Controls which merge tool is used by git-mergetool(1) when the -g/--gui flag is specified. The list below shows the valid built-in values. Any other value is treated as a custom merge tool and requires that a corresponding mergetool.<guitool>.cmd variable is defined.

? araxis

? bc

? bc3

? codecompare

? deltawalker

? diffmerge

? diffuse

? ecmerge

? emerge

? examdiff

? guiffy

? gvimdiff

? gvimdiff2

? gvimdiff3

? kdiff3

? meld

? opendiff

? p4merge

? smerge

? tkdiff

? tortoisemerge

? vimdiff

? vimdiff2

? vimdiff3

? winmerge

? xxdiff

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.meld.hasOutput

Older versions of meld do not support the --output option. Git will attempt to detect whether meld supports --output by inspecting the output of meld --help. Configuring mergetool.meld.hasOutput will make Git skip these checks and use the configured value instead. Setting mergetool.meld.hasOutput to true tells Git to unconditionally use the --output option, and false avoids using --output.

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.writeToTemp

Git writes temporary BASE, LOCAL, and REMOTE versions of conflicting files in the worktree by default. Git will attempt to use a temporary directory for these files when set true. Defaults to false.

mergetool.prompt

Prompt before each invocation of the merge resolution program.

notes.mergeStrategy

Which merge strategy to choose by default when resolving notes conflicts. Must be one of manual, ours, theirs, union, or cat_sort_uniq. Defaults to manual. See "NOTES MERGE STRATEGIES" section of git-notes(1) for more information on each strategy.

notes.<name>.mergeStrategy

Which merge strategy to choose when doing a notes merge into refs/notes/<name>. This overrides the more general "notes.mergeStrategy". See the "NOTES MERGE STRATEGIES" section in git-notes(1) for more information on the available strategies.

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, cat_sort_uniq, 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. Maximum value is 4095.

pack.windowMemory

The maximum size of memory that is consumed by each thread in git-pack-objects(1) for pack window memory when no limit is given on the command line. The value can be suffixed with "k", "m", or "g". When left unconfigured (or set explicitly to 0), there will be 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.island

An extended regular expression configuring a set of delta islands. See "DELTA ISLANDS" in git-pack-objects(1) for details.

pack.islandCore

Specify an island name which gets to have its objects be packed first. This creates a kind of pseudo-pack at the front of one pack, so that the objects from the specified island are hopefully faster to copy into any pack that should be served to a user requesting these objects. In practice this means that the island specified should likely correspond to what is the most commonly cloned in the repo. See also "DELTA ISLANDS" in git-pack-objects(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. Maximum value is 65535.

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") 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). Reaching this limit results in the creation of multiple packfiles; which in turn prevents bitmaps from being created. The minimum size allowed is limited to 1 MiB. The default is unlimited. Common unit suffixes of k, m, or g are supported.

pack.useBitmaps

When true, git will use pack bitmaps (if available) when packing to stdout (e.g., during the server side of a fetch). Defaults to true. You should not generally need to turn this off unless you are debugging pack bitmaps.

pack.useSparse

When true, git will default to using the --sparse option in git pack-objects when the --revs option is present. This algorithm only walks trees that appear in paths that introduce new objects. This can have significant performance benefits when computing a pack to send a small change. However, it is possible that extra objects are added to the pack-file if the included commits contain certain types of direct renames. Default is false unless feature.experimental is enabled.

pack.writeBitmaps (deprecated)

This is a deprecated synonym for repack.writeBitmaps.

pack.writeBitmapHashCache

When true, git will include a "hash cache" section in the bitmap index (if one is written). This cache can be used to feed git?s delta heuristics, potentially leading to better deltas between bitmapped and non-bitmapped objects (e.g., when serving a fetch between an older, bitmapped pack and objects that have been pushed since the last gc). The downside is that it consumes 4 bytes per object of disk space. Defaults to true.

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.

protocol.allow

If set, provide a user defined default policy for all protocols which don?t explicitly have a policy (protocol.<name>.allow). By default, if unset, known-safe protocols (http, https, git, ssh) have a default policy of always, known-dangerous protocols (ext) have a default policy of never, and all other protocols (including file) have a default policy of user. Supported policies:

? always - protocol is always able to be used.

? never - protocol is never able to be used.

? user - protocol is only able to be used when GIT_PROTOCOL_FROM_USER is either unset or has a value of 1. This policy should be used when you want a protocol to be directly usable by the user but don?t want it used by commands which execute clone/fetch/push commands without user input, e.g.

recursive submodule initialization.

protocol.<name>.allow

Set a policy to be used by protocol <name> with clone/fetch/push commands. See protocol.allow above for the available policies.

The protocol names currently used by git are:

- file: any local file-based path (including file:// URLs, or local paths)
- git: the anonymous git protocol over a direct TCP connection (or proxy, if configured)
- ssh: git over ssh (including host:path syntax, ssh://, etc).
- http: git over http, both "smart http" and "dumb http". Note that this does not include https; if you want to configure both, you must do so individually.
- any external helpers are named by their protocol (e.g., use hg to allow the git-remote-hg helper)

protocol.version

Experimental. If set, clients will attempt to communicate with a server using the specified protocol version. If unset, no attempt will be made by the client to communicate using a particular protocol version, this results in protocol version 0 being used. Supported versions:

- 0 - the original wire protocol.
- 1 - the original wire protocol with the addition of a version string in the initial response from the server.
- 2 - wire protocol version 2[2].

pull.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). This setting overrides merge.ff when pulling.

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.

When merges, pass the --rebase-merges option to git rebase so that the local merge commits are included in the rebase (see git-rebase(1) for details).

When preserve (deprecated in favor of merges), also pass --preserve-merges along to git rebase so that locally committed merge commits will not be flattened by running git pull.

When the value is interactive, the rebase is run in interactive mode.

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 explicitly given. Different values are well-suited for specific workflows; for instance, in a purely central workflow (i.e. the fetch source is equal to the push destination), upstream is probably what you want. Possible values are:

? nothing - do not push anything (error out) unless a refspec is explicitly given. This is primarily meant for people who want to avoid mistakes by always being explicit.

? current - push the current branch to update a branch with the same name on the receiving end. Works in both central and non-central workflows.

? upstream - push the current branch back to the branch whose changes are usually integrated into the current branch (which is called @{upstream}). This mode only makes sense if you are pushing to the same repository you would normally pull from (i.e. central workflow).

? tracking - This is a deprecated synonym for upstream.

? simple - in centralized workflow, work like upstream with an added safety to refuse to push if the upstream branch?s name is different from the local one.

When pushing to a remote that is different from the remote you normally

pull from, work as current. This is the safest option and is suited for beginners.

This mode has become the default in Git 2.0.

? matching - push all branches having the same name on both ends. This makes the repository you are pushing to remember the set of branches that will be pushed out (e.g. if you always push maint and master there and no other branches, the repository you push to will have these two branches, and your local maint and master will be pushed there).

To use this mode effectively, you have to make sure all the branches you would push out are ready to be pushed out before running git push, as the whole point of this mode is to allow you to push all of the branches in one go. If you usually finish work on only one branch and push out the result, while other branches are unfinished, this mode is not for you. Also this mode is not suitable for pushing into a shared central repository, as other people may add new branches there, or update the tip of existing branches outside your control.

This used to be the default, but not since Git 2.0 (simple is the new default).

push.followTags

If set to true enable --follow-tags option by default. You may override this configuration at time of push by specifying --no-follow-tags.

push.gpgSign

May be set to a boolean value, or the string if-asked. A true value causes all pushes to be GPG signed, as if --signed is passed to git-push(1). The string if-asked causes pushes to be signed if the server supports it, as if --signed=if-asked is passed to git push. A false value may override a value from a lower-priority config file. An explicit command-line flag always overrides this config option.

push.pushOption

When no --push-option=<option> argument is given from the command line, git push behaves as if each <value> of this variable is given as --push-option=<value>.

This is a multi-valued variable, and an empty value can be used in a higher

priority configuration file (e.g. .git/config in a repository) to clear the values inherited from a lower priority configuration files (e.g. $HOME/.gitconfig).

Example:

/etc/gitconfig push.pushoption = a push.pushoption = b

~/.gitconfig push.pushoption = c

repo/.git/config push.pushoption = push.pushoption = b

This will result in only b (a and c are cleared).

push.recurseSubmodules

Make sure all submodule commits used by the revisions to be pushed are available on a remote-tracking branch. If the value is check then Git will verify that all submodule commits that changed in the revisions to be pushed are available on at least one remote of the submodule. If any commits are missing, the push will be aborted and exit with non-zero status. If the value is on-demand then all submodules that changed in the revisions to be pushed will be pushed. If on-demand was not able to push all necessary revisions it will also be aborted and exit with non-zero status. If the value is no then default behavior of ignoring submodules when pushing is retained. You may override this configuration at time of push by specifying --recurse-submodules=check|on-demand|no.

rebase.useBuiltin

Unused configuration variable. Used in Git versions 2.20 and 2.21 as an escape hatch to enable the legacy shellscript implementation of rebase. Now the built-in rewrite of it in C is always used. Setting this will emit a warning, to alert any remaining users that setting this now does nothing.

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.

rebase.autoStash

When set to true, automatically create a temporary stash entry before the operation begins, and apply it after the operation ends. This means that you

can run rebase on a dirty worktree. However, use with care: the final stash application after a successful rebase might result in non-trivial conflicts.

This option can be overridden by the --no-autostash and --autostash options of git-rebase(1). Defaults to false.

rebase.missingCommitsCheck

If set to "warn", git rebase -i will print a warning if some commits are removed (e.g. a line was deleted), however the rebase will still proceed. If set to "error", it will print the previous warning and stop the rebase, git rebase --edit-todo can then be used to correct the error. If set to "ignore", no checking is done. To drop a commit without warning or error, use the drop command in the todo list. Defaults to "ignore".

rebase.instructionFormat

A format string, as specified in git-log(1), to be used for the todo list during an interactive rebase. The format will automatically have the long commit hash prepended to the format.

rebase.abbreviateCommands

If set to true, git rebase will use abbreviated command names in the todo list resulting in something like this:

```
p deadbee The oneline of the commit
p fa1afe1 The oneline of the next commit
...
```

instead of:

```
pick deadbee The oneline of the commit
pick fa1afe1 The oneline of the next commit
...
```

Defaults to false.

rebase.rescheduleFailedExec

Automatically reschedule exec commands that failed. This only makes sense in interactive mode (or when an --exec option was provided). This is the same as specifying the --reschedule-failed-exec option.

receive.advertiseAtomic

By default, git-receive-pack will advertise the atomic push capability to its clients. If you don?t want to advertise this capability, set this variable to

false.

receive.advertisePushOptions

When set to true, git-receive-pack will advertise the push options capability to its clients. False 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.certNonceSeed

By setting this variable to a string, git receive-pack will accept a git push --signed and verifies it by using a "nonce" protected by HMAC using this string as a secret key.

receive.certNonceSlop

When a git push --signed sent a push certificate with a "nonce" that was issued by a receive-pack serving the same repository within this many seconds, export the "nonce" found in the certificate to GIT_PUSH_CERT_NONCE to the hooks (instead of what the receive-pack asked the sending side to include). This may allow writing checks in pre-receive and post-receive a bit easier. Instead of checking GIT_PUSH_CERT_NONCE_SLOP environment variable that records by how many seconds the nonce is stale to decide if they want to accept the certificate, they only can check GIT_PUSH_CERT_NONCE_STATUS is OK.

receive.fsckObjects

If it is set to true, git-receive-pack will check all received objects. See transfer.fsckObjects for what?s checked. Defaults to false. If not set, the value of transfer.fsckObjects is used instead.

receive.fsck.<msg-id>

Acts like fsck.<msg-id>, but is used by git-receive-pack(1) instead of git-fsck(1). See the fsck.<msg-id> documentation for details.

receive.fsck.skipList

Acts like fsck.skipList, but is used by git-receive-pack(1) instead of git-fsck(1). See the fsck.skipList documentation for details.

receive.keepAlive

After receiving the pack from the client, receive-pack may produce no output (if --quiet was specified) while processing the pack, causing some networks to

drop the TCP connection. With this option set, if receive-pack does not transmit any data in this phase for receive.keepAlive seconds, it will send a short keepalive packet. The default is 5 seconds; set to 0 to disable keepalives entirely.

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.maxInputSize

If the size of the incoming pack stream is larger than this limit, then git-receive-pack will error out, instead of accepting the pack file. If not set or set to 0, then the size is unlimited.

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".

Another option is "updateInstead" which will update the working tree if pushing into the current branch. This option is intended for synchronizing working directories when one side is not easily accessible via interactive ssh (e.g. a live web site, hence the requirement that the working directory be clean). This mode also comes in handy when developing inside a VM to test and fix code on

different Operating Systems.

By default, "updateInstead" will refuse the push if the working tree or the index have any difference from the HEAD, but the push-to-checkout hook can be used to customize this. See githooks(5).

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.hideRefs

This variable is the same as transfer.hideRefs, but applies only to receive-pack (and so affects pushes, but not fetches). An attempt to update or delete a hidden ref by git push is rejected.

receive.updateServerInfo

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

receive.shallowUpdate

If set to true, .git/shallow can be updated when new refs require new shallow roots. Otherwise those refs are rejected.

remote.pushDefault

The remote to push to by default. Overrides branch.<name>.remote for all branches, and is overridden by branch.<name>.pushRemote for specific branches.

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>.proxyAuthMethod

For remotes that require curl (http, https and ftp), the method to use for authenticating against the proxy in use (probably set in remote.<name>.proxy).

See http.proxyAuthMethod.

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.

remote.<name>.prune

When set to true, fetching from this remote by default will also remove any remote-tracking references that no longer exist on the remote (as if the --prune option was given on the command line). Overrides fetch.prune settings,

if any.

remote.<name>.pruneTags

When set to true, fetching from this remote by default will also remove any local tags that no longer exist on the remote if pruning is activated in general via remote.<name>.prune, fetch.prune or --prune. Overrides fetch.pruneTags settings, if any.

See also remote.<name>.prune and the PRUNING section of git-fetch(1).

remote.<name>.promisor

When set to true, this remote will be used to fetch promisor objects.

remote.<name>.partialclonefilter

The filter that will be applied when fetching from this promisor remote.

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.

repack.packKeptObjects

If set to true, makes git repack act as if --pack-kept-objects was passed. See git-repack(1) for details. Defaults to false normally, but true if a bitmap index is being written (either via --write-bitmap-index or repack.writeBitmaps).

repack.useDeltaIslands

If set to true, makes git repack act as if --delta-islands was passed. Defaults to false.

repack.writeBitmaps

When true, git will write a bitmap index when packing all objects to disk (e.g., when git repack -a is run). This index can speed up the "counting objects" phase of subsequent packs created for clones and fetches, at the cost of some disk space and extra time spent on the initial repack. This has no

effect if multiple packfiles are created. Defaults to true on bare repos, false otherwise.

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.

reset.quiet

When set to true, git reset will default to the --quiet option.

safe.directory

These config entries specify Git-tracked directories that are considered safe even if they are owned by someone other than the current user. By default, Git will refuse to even parse a Git config of a repository owned by someone else, let alone run its hooks, and this config setting allows users to specify exceptions, e.g. for intentionally shared repositories (see the --shared option in git-init(1)).

This is a multi-valued setting, i.e. you can add more than one directory via git config --add. To reset the list of safe directories (e.g. to override any such directories specified in the system config), add a safe.directory entry with an empty value.

This config setting is only respected when specified in a system or global config, not when it is specified in a repository config or via the command line option -c safe.directory=<path>.

The value of this setting is interpolated, i.e. ~/<path> expands to a path relative to the home directory and %(prefix)/<path> expands to a path relative to Git?s (runtime) prefix.

To completely opt-out of this security check, set safe.directory to the string *. This will allow all repositories to be treated as if their directory was listed in the safe.directory list. If safe.directory=* is set in system config

and you want to re-enable this protection, then initialize your list with an empty value before listing the repositories that you deem safe.

As explained, Git only allows you to access repositories owned by yourself, i.e. the user who is running Git, by default. When Git is running as root in a non Windows platform that provides sudo, however, git checks the SUDO_UID environment variable that sudo creates and will allow access to the uid recorded as its value in addition to the id from root. This is to make it easy to perform a common sequence during installation "make && sudo make install". A git process running under sudo runs as root but the sudo command exports the environment variable to record which id the original user has. If that is not what you would prefer and want git to only trust repositories that are owned by root instead, then you can remove the SUDO_UID variable from root?s environment before invoking git.

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)

Deprecated alias for sendemail.smtpEncryption = ssl.

sendemail.smtpsslcertpath

Path to ca-certificates (either a directory or a single file). Set it to an empty string to disable certificate verification.

sendemail.<identity>.*

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

sendemail.aliasesFile, sendemail.aliasFileType, sendemail.annotate, 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.tocmd, sendemail.smtpDomain, sendemail.smtpServer, sendemail.smtpServerPort, sendemail.smtpServerOption, sendemail.smtpUser, sendemail.thread, sendemail.transferEncoding, sendemail.validate, sendemail.xmailer

See git-send-email(1) for description.

sendemail.signedoffcc (deprecated)

Deprecated alias for sendemail.signedoffbycc.

sendemail.smtpBatchSize

Number of messages to be sent per connection, after that a relogin will happen. If the value is 0 or undefined, send all messages in one connection. See also the --batch-size option of git-send-email(1).

sendemail.smtpReloginDelay

Seconds wait before reconnecting to smtp server. See also the --relogin-delay option of git-send-email(1).

sequence.editor

Text editor used by git rebase -i for editing the rebase instruction 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.

showBranch.default

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

splitIndex.maxPercentChange

When the split index feature is used, this specifies the percent of entries the split index can contain compared to the total number of entries in both the split index and the shared index before a new shared index is written. The value should be between 0 and 100. If the value is 0 then a new shared index is always written, if it is 100 a new shared index is never written. By default the value is 20, so a new shared index is written if the number of entries in the split index would be greater than 20 percent of the total number of entries. See git-update-index(1).

splitIndex.sharedIndexExpire

When the split index feature is used, shared index files that were not modified since the time this variable specifies will be removed when a new shared index

file is created. The value "now" expires all entries immediately, and "never" suppresses expiration altogether. The default value is "2.weeks.ago". Note that a shared index file is considered modified (for the purpose of expiration) each time a new split-index file is either created based on it or read from it. See git-update-index(1).

ssh.variant

By default, Git determines the command line arguments to use based on the basename of the configured SSH command (configured using the environment variable GIT_SSH or GIT_SSH_COMMAND or the config setting core.sshCommand). If the basename is unrecognized, Git will attempt to detect support of OpenSSH options by first invoking the configured SSH command with the -G (print configuration) option and will subsequently use OpenSSH options (if that is successful) or no options besides the host and remote command (if it fails).

The config variable ssh.variant can be set to override this detection. Valid values are ssh (to use OpenSSH options), plink, putty, tortoiseplink, simple (no options except the host and remote command). The default auto-detection can be explicitly requested using the value auto. Any other value is treated as ssh. This setting can also be overridden via the environment variable GIT_SSH_VARIANT.

The current command-line parameters used for each variant are as follows:

- ssh - [-p port] [-4] [-6] [-o option] [username@]host command
- simple - [username@]host command
- plink or putty - [-P port] [-4] [-6] [username@]host command
- tortoiseplink - [-P port] [-4] [-6] -batch [username@]host command

Except for the simple variant, command-line parameters are likely to change as git gains new features.

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.short

Set to true to enable --short by default in git-status(1). The option --no-short takes precedence over this variable.

status.branch

Set to true to enable --branch by default in git-status(1). The option --no-branch takes precedence over this variable.

status.aheadBehind

Set to true to enable --ahead-behind and false to enable --no-ahead-behind by default in git-status(1) for non-porcelain status formats. Defaults to true.

status.displayCommentPrefix

If set to true, git-status(1) will insert a comment prefix before each output line (starting with core.commentChar, i.e. # by default). This was the behavior of git-status(1) in Git 1.8.4 and previous. Defaults to false.

status.renameLimit

The number of files to consider when performing rename detection in git-status(1) and git-commit(1). Defaults to the value of diff.renameLimit.

status.renames

Whether and how Git detects renames in git-status(1) and git-commit(1) . If set to "false", rename detection is disabled. If set to "true", basic rename detection is enabled. If set to "copies" or "copy", Git will detect copies, as well. Defaults to the value of diff.renames.

status.showStash

If set to true, git-status(1) will display the number of entries currently stashed away. Defaults to false.

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 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)). Please note that the summary output command will be suppressed for all submodules when diff.ignoreSubmodules is set to all or only for those submodules where submodule.<name>.ignore=all. The only exception to that rule is that status and commit will show staged submodule changes. To also view the summary for ignored submodules you can either use the --ignore-submodules=dirty command-line option or the git submodule summary command, which shows a similar output but does not honor these settings.

stash.useBuiltin

Set to false to use the legacy shell script implementation of git-stash(1). Is true by default, which means use the built-in rewrite of it in C.

The C rewrite is first included with Git version 2.22 (and Git for Windows version 2.19). This option serves as an escape hatch to re-enable the legacy version in case any bugs are found in the rewrite. This option and the shell script version of git-stash(1) will be removed in some future release.

If you find some reason to set this option to false, other than one-off testing, you should report the behavior difference as a bug in Git (see https://git-scm.com/community for details).

stash.showPatch

If this is set to true, the git stash show command without an option will show the stash entry in patch form. Defaults to false. See description of show command in git-stash(1).

stash.showStat

If this is set to true, the git stash show command without an option will show diffstat of the stash entry. Defaults to true. See description of show command in git-stash(1).

submodule.<name>.url

The URL for a submodule. This variable is copied from the .gitmodules file to the git config via git submodule init. The user can change the configured URL

before obtaining the submodule via git submodule update. If neither submodule.<name>.active or submodule.active are set, the presence of this variable is used as a fallback to indicate whether the submodule is of interest to git commands. See git-submodule(1) and gitmodules(5) for details.

submodule.<name>.update

The method by which a submodule is updated by git submodule update, which is the only affected command, others such as git checkout --recurse-submodules are unaffected. It exists for historical reasons, when git submodule was the only command to interact with submodules; settings like submodule.active and pull.rebase are more specific. It is populated by git submodule init from the gitmodules(5) file. See description of update command in git-submodule(1).

submodule.<name>.branch

The remote branch name for a submodule, used by git submodule update --remote. Set this option to override the value 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 (but it will nonetheless show up in the output of status and commit when it has been staged), "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. The git submodule commands are not affected by this setting.

submodule.<name>.active

Boolean value indicating if the submodule is of interest to git commands. This config option takes precedence over the submodule.active config option. See gitsubmodules(7) for details.

submodule.active

A repeated field which contains a pathspec used to match against a submodule?s path to determine if the submodule is of interest to git commands. See gitsubmodules(7) for details.

submodule.recurse

Specifies if commands recurse into submodules by default. This applies to all commands that have a --recurse-submodules option, except clone. Defaults to false.

submodule.fetchJobs

Specifies how many submodules are fetched/cloned at the same time. A positive integer allows up to that number of submodules fetched in parallel. A value of 0 will give some reasonable default. If unset, it defaults to 1.

submodule.alternateLocation

Specifies how the submodules obtain alternates when submodules are cloned. Possible values are no, superproject. By default no is assumed, which doesn?t add references. When the value is set to superproject the submodule to be cloned computes its alternates location relative to the superprojects alternate.

submodule.alternateErrorStrategy

Specifies how to treat errors with the alternates for a submodule as computed via submodule.alternateLocation. Possible values are ignore, info, die. Default is die. Note that if set to ignore or info, and if there is an error with the computed alternate, the clone proceeds as if no alternate was specified.

tag.forceSignAnnotated

A boolean to specify whether annotated tags created should be GPG signed. If --annotate is specified on the command line, it takes precedence over this option.

tag.sort

This variable controls the sort ordering of tags when displayed by git-tag(1).

Without the "--sort=<value>" option provided, the value of this variable will be used as the default.

tag.gpgSign

A boolean to specify whether all tags should be GPG signed. Use of this option when running in an automated script can result in a large number of tags being signed. It is therefore convenient to use an agent to avoid typing your gpg passphrase several times. Note that this option doesn?t affect tag signing behavior enabled by "-u <keyid>" or "--local-user=<keyid>" options.

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).

Trace2 config settings are only read from the system and global config files; repository local and worktree config files and -c command line arguments are not respected.

trace2.normalTarget

This variable controls the normal target destination. It may be overridden by the GIT_TRACE2 environment variable. The following table shows possible values.

trace2.perfTarget

This variable controls the performance target destination. It may be overridden by the GIT_TRACE2_PERF environment variable. The following table shows possible values.

trace2.eventTarget

This variable controls the event target destination. It may be overridden by the GIT_TRACE2_EVENT environment variable. The following table shows possible values.

- 0 or false - Disables the target.
- 1 or true - Writes to STDERR.
- [2-9] - Writes to the already opened file descriptor.
- <absolute-pathname> - Writes to the file in append mode. If the target already exists and is a directory, the traces will be written to files (one per process) underneath the given directory.

? af_unix:[<socket_type>:]<absolute-pathname> - Write to a Unix DomainSocket (on platforms that support them). Socket type can be either stream or dgram; if omitted Git will try both.

trace2.normalBrief

Boolean. When true time, filename, and line fields are omitted from normal output. May be overridden by the GIT_TRACE2_BRIEF environment variable. Defaults to false.

trace2.perfBrief

Boolean. When true time, filename, and line fields are omitted from PERF output. May be overridden by the GIT_TRACE2_PERF_BRIEF environment variable. Defaults to false.

trace2.eventBrief

Boolean. When true time, filename, and line fields are omitted from event output. May be overridden by the GIT_TRACE2_EVENT_BRIEF environment variable. Defaults to false.

trace2.eventNesting

Integer. Specifies desired depth of nested regions in the event output. Regions deeper than this value will be omitted. May be overridden by the GIT_TRACE2_EVENT_NESTING environment variable. Defaults to 2.

trace2.configParams

A comma-separated list of patterns of "important" config settings that should be recorded in the trace2 output. For example, core.*,remote.*.url would cause the trace2 output to contain events listing each configured remote. May be overridden by the GIT_TRACE2_CONFIG_PARAMS environment variable. Unset by default.

trace2.destinationDebug

Boolean. When true Git will print error messages when a trace target destination cannot be opened for writing. By default, these errors are suppressed and tracing is silently disabled. May be overridden by the GIT_TRACE2_DST_DEBUG environment variable.

trace2.maxFiles

Integer. When writing trace files to a target directory, do not write additional traces if we would exceed this many files. Instead, write a sentinel

file that will block further tracing to this directory. Defaults to 0, which disables this check.

transfer.fsckObjects

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

When set, the fetch or receive will abort in the case of a malformed object or a link to a nonexistent object. In addition, various other issues are checked for, including legacy issues (see fsck.<msg-id>), and potential security issues like the existence of a .GIT directory or a malicious .gitmodules file (see the release notes for v2.2.1 and v2.17.1 for details). Other sanity and security checks may be added in future releases.

On the receiving side, failing fsckObjects will make those objects unreachable, see "QUARANTINE ENVIRONMENT" in git-receive-pack(1). On the fetch side, malformed objects will instead be left unreferenced in the repository.

Due to the non-quarantine nature of the fetch.fsckObjects implementation it cannot be relied upon to leave the object store clean like receive.fsckObjects can.

As objects are unpacked they?re written to the object store, so there can be cases where malicious objects get introduced even though the "fetch" failed, only to have a subsequent "fetch" succeed because only new incoming objects are checked, not those that have already been written to the object store. That difference in behavior should not be relied upon. In the future, such objects may be quarantined for "fetch" as well.

For now, the paranoid need to find some way to emulate the quarantine environment if they?d like the same protection as "push". E.g. in the case of an internal mirror do the mirroring in two steps, one to fetch the untrusted objects, and then do a second "push" (which will use the quarantine) to another internal repo, and have internal clients consume this pushed-to repository, or embargo internal fetches and only allow them once a full "fsck" has run (and no new fetches have happened in the meantime).

transfer.hideRefs

String(s) receive-pack and upload-pack use to decide which refs to omit from their initial advertisements. Use more than one definition to specify multiple

prefix strings. A ref that is under the hierarchies listed in the value of this variable is excluded, and is hidden when responding to git push or git fetch. See receive.hideRefs and uploadpack.hideRefs for program-specific versions of this config.

You may also include a ! in front of the ref name to negate the entry, explicitly exposing it, even if an earlier entry marked it as hidden. If you have multiple hideRefs values, later entries override earlier ones (and entries in more-specific config files override less-specific ones).

If a namespace is in use, the namespace prefix is stripped from each reference before it is matched against transfer.hiderefs patterns. For example, if refs/heads/master is specified in transfer.hideRefs and the current namespace is foo, then refs/namespaces/foo/refs/heads/master is omitted from the advertisements but refs/heads/master and refs/namespaces/bar/refs/heads/master are still advertised as so-called "have" lines. In order to match refs before stripping, add a ^ in front of the ref name. If you combine ! and ^, ! must be specified first.

Even if you hide refs, a client may still be able to steal the target objects via the techniques described in the "SECURITY" section of the gitnamespaces(7) man page; it?s best to keep private data in a separate repository.

transfer.unpackLimit

When fetch.unpackLimit or receive.unpackLimit are not set, the value of this variable is used instead. The default value is 100.

uploadarchive.allowUnreachable

If true, allow clients to use git archive --remote to request any tree, whether reachable from the ref tips or not. See the discussion in the "SECURITY" section of git-upload-archive(1) for more details. Defaults to false.

uploadpack.hideRefs

This variable is the same as transfer.hideRefs, but applies only to upload-pack (and so affects only fetches, not pushes). An attempt to fetch a hidden ref by git fetch will fail. See also uploadpack.allowTipSHA1InWant.

uploadpack.allowTipSHA1InWant

When uploadpack.hideRefs is in effect, allow upload-pack to accept a fetch request that asks for an object at the tip of a hidden ref (by default, such a

request is rejected). See also uploadpack.hideRefs. Even if this is false, a client may be able to steal objects via the techniques described in the "SECURITY" section of the gitnamespaces(7) man page; it?s best to keep private data in a separate repository.

uploadpack.allowReachableSHA1InWant

Allow upload-pack to accept a fetch request that asks for an object that is reachable from any ref tip. However, note that calculating object reachability is computationally expensive. Defaults to false. Even if this is false, a client may be able to steal objects via the techniques described in the "SECURITY" section of the gitnamespaces(7) man page; it?s best to keep private data in a separate repository.

uploadpack.allowAnySHA1InWant

Allow upload-pack to accept a fetch request that asks for any object at all. Defaults to false.

uploadpack.keepAlive

When upload-pack has started pack-objects, there may be a quiet period while pack-objects prepares the pack. Normally it would output progress information, but if --quiet was used for the fetch, pack-objects will output nothing at all until the pack data begins. Some clients and networks may consider the server to be hung and give up. Setting this option instructs upload-pack to send an empty keepalive packet every uploadpack.keepAlive seconds. Setting this option to 0 disables keepalive packets entirely. The default is 5 seconds.

uploadpack.packObjectsHook

If this option is set, when upload-pack would run git pack-objects to create a packfile for a client, it will run this shell command instead. The pack-objects command and arguments it would have run (including the git pack-objects at the beginning) are appended to the shell command. The stdin and stdout of the hook are treated as if pack-objects itself was run. I.e., upload-pack will feed input intended for pack-objects to the hook, and expects a completed packfile on stdout.

Note that this configuration variable is ignored if it is seen in the repository-level config (this is a safety measure against fetching from untrusted repositories).

uploadpack.allowFilter

If this option is set, upload-pack will support partial clone and partial fetch object filtering.

uploadpack.allowRefInWant

If this option is set, upload-pack will support the ref-in-want feature of the protocol version 2 fetch command. This feature is intended for the benefit of load-balanced servers which may not have the same view of what OIDs their refs point to due to replication delay.

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.

Note that any protocol restrictions will be applied to the rewritten URL. If the rewrite changes the URL to use a custom protocol or remote helper, you may need to adjust the protocol.*.allow config to permit the request. In particular, protocols you expect to use for submodules must be set to always rather than the default of user. See the description of protocol.allow above.

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.name, user.email, author.name, author.email, committer.name, committer.email

The user.name and user.email variables determine what ends up in the author and

committer field of commit objects. If you need the author or committer to be different, the author.name, author.email, committer.name or committer.email variables can be set. Also, all of these can be overridden by the GIT_AUTHOR_NAME, GIT_AUTHOR_EMAIL, GIT_COMMITTER_NAME, GIT_COMMITTER_EMAIL and EMAIL environment variables.

Note that the name forms of these variables conventionally refer to some form of a personal name. See git-commit(1) and the environment variables section of git(1) for more information on these settings and the credential.username option if you?re looking for authentication credentials instead.

user.useConfigOnly

Instruct Git to avoid trying to guess defaults for user.email and user.name, and instead retrieve the values only from the configuration. For example, if you have multiple email addresses and would like to use a different one for each repository, then with this configuration option set to true in the global config along with a name, Git will prompt you to set up an email before making new commits in a newly cloned repository. Defaults to false.

user.signingKey

If git-tag(1) or git-commit(1) is not selecting the key you want it to automatically when creating a signed tag or commit, 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.

versionsort.prereleaseSuffix (deprecated)

Deprecated alias for versionsort.suffix. Ignored if versionsort.suffix is set.

versionsort.suffix

Even when version sort is used in git-tag(1), tagnames with the same base version but different suffixes are still sorted lexicographically, resulting e.g. in prerelease tags appearing after the main release (e.g. "1.0-rc1" after "1.0"). This variable can be specified to determine the sorting order of tags with different suffixes.

By specifying a single suffix in this variable, any tagname containing that suffix will appear before the corresponding main release. E.g. if the variable is set to "-rc", then all "1.0-rcX" tags will appear before "1.0". If specified

multiple times, once per suffix, then the order of suffixes in the configuration will determine the sorting order of tagnames with those suffixes. E.g. if "-pre" appears before "-rc" in the configuration, then all "1.0-preX" tags will be listed before any "1.0-rcX" tags. The placement of the main release tag relative to tags with various suffixes can be determined by specifying the empty suffix among those other suffixes. E.g. if the suffixes "-rc", "", "-ck" and "-bfs" appear in the configuration in this order, then all "v4.8-rcX" tags are listed first, followed by "v4.8", then "v4.8-ckX" and finally "v4.8-bfsX".

If more than one suffixes match the same tagname, then that tagname will be sorted according to the suffix which starts at the earliest position in the tagname. If more than one different matching suffixes start at that earliest position, then that tagname will be sorted according to the longest of those suffixes. The sorting order between different suffixes is undefined if they are in multiple config files.

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.

worktree.guessRemote

If no branch is specified and neither -b nor -B nor --detach is used, then git worktree add defaults to creating a new branch from HEAD. If worktree.guessRemote is set to true, worktree add tries to find a remote-tracking branch whose name uniquely matches the new branch name. If such a branch exists, it is checked out and set as "upstream" for the new branch. If no such match can be found, it falls back to creating a new branch from the current HEAD.

BUGS

When using the deprecated [section.subsection] syntax, changing a value will result in adding a multi-line key instead of a change, if the subsection is given with at least one uppercase character. For example when the config looks like

```
[section.subsection]
  key = value1
```

and running git config section.Subsection.key value2 will result in

[section.subsection]

key = value1

key = value2

GIT

Part of the git(1) suite

NOTES

1. the multi-pack-index design document

   file:///usr/share/doc/git/html/technical/multi-pack-index.html

2. wire protocol version 2

   file:///usr/share/doc/git/html/technical/protocol-v2.html