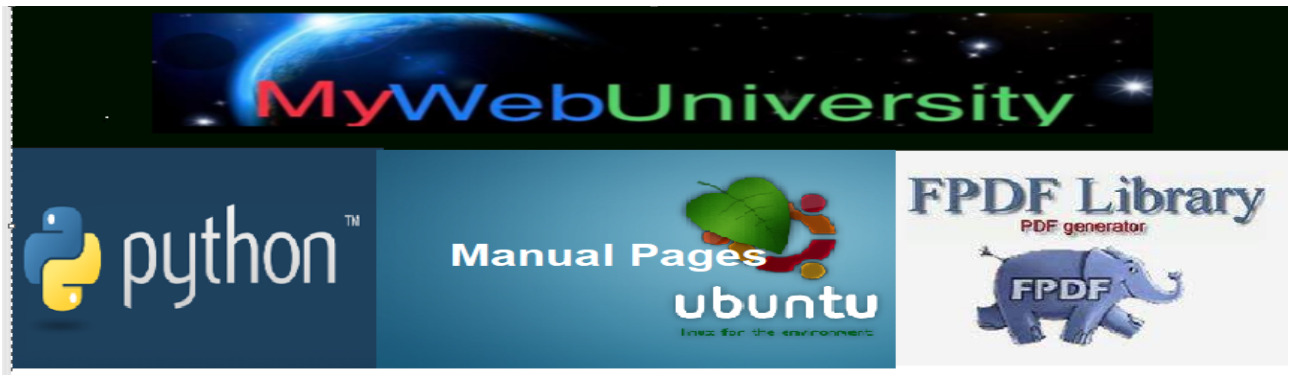

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'git-status.1'

$ man git-status.1

GIT-STATUS(1) Git Manual GIT-STATUS(1)

NAME

git-status - Show the working tree status

SYNOPSIS

git status [<options>] [--] [<pathspec>...]

DESCRIPTION

Displays paths that have differences between the index file and the current HEAD commit, paths that have differences between the working tree and the index file, and paths in the working tree that are not tracked by Git (and are not ignored by gitignore(5)). The first are what you would commit by running git commit; the second and third are what you could commit by running git add before running git commit.

OPTIONS

-s, --short

Give the output in the short-format.

-b, --branch

Show the branch and tracking info even in short-format.

--show-stash

Show the number of entries currently stashed away.

--porcelain[=<version>]

Give the output in an easy-to-parse format for scripts. This is similar to the short output, but will remain stable across Git versions and regardless of user configuration. See below for details.

The <version> parameter is used to specify the format version. This is optional and

defaults to the original version v1 format.

--long

Give the output in the long-format. This is the default.

-v, --verbose

In addition to the names of files that have been changed, also show the textual changes that are staged to be committed (i.e., like the output of git diff --cached). If -v is specified twice, then also show the changes in the working tree that have not yet been staged (i.e., like the output of git diff).

-u[<mode>], --untracked-files[=<mode>]

Show untracked files.

The mode parameter is used to specify the handling of untracked files. It is optional: it defaults to all, and if specified, it must be stuck to the option (e.g. -uno, but not -u no).

The possible options are:

no

Show no untracked files.

normal

Show untracked files and directories.

all

Also show individual files in untracked directories.

When -u option is not used, untracked files and directories are shown (i.e. the same as specifying normal), to help you avoid forgetting to add newly created files. Because it takes extra work to find untracked files in the filesystem, this mode may take some time in a large working tree. Consider enabling untracked cache and split index if supported (see git update-index --untracked-cache and git update-index --split-index), Otherwise you can use no to have git status return more quickly without showing untracked files.

All usual spellings for Boolean value true are taken as normal and false as no.

The default can be changed using the status.showUntrackedFiles configuration variable documented in git-config(1).

--ignore-submodules[=<when>]

Ignore changes to submodules when looking for changes. <when> can be either none, untracked, dirty or all, which is the default.

none

will consider the submodule modified when it either contains untracked or modified files or its HEAD differs from the commit recorded in the superproject and can be used to override any settings of the ignore option in git-config(1) or gitmodules(5).

untracked
: submodules are not considered dirty when they only contain untracked content (but they are still scanned for modified content).

dirty
: ignore all changes to the work tree of submodules, only changes to the commits stored in the superproject are shown (this was the behavior before 1.7.0).

all
: hide all changes to submodules (and suppresses the output of submodule summaries when the config option status.submoduleSummary is set).

--ignored[=<mode>]
: Show ignored files as well.

The mode parameter is used to specify the handling of ignored files. It is optional: it defaults to traditional.

The possible options are:

traditional
: Show ignored files and directories, unless --untracked-files=all is specified, in which case individual files in ignored directories are displayed.

no
: Show no ignored files.

matching
: Show ignored files and directories matching an ignore pattern.

Paths that explicitly match an ignored pattern are shown. If a directory matches an ignore pattern, then it is shown, but not paths contained in the ignored directory.

If a directory does not match an ignore pattern, but all contents are ignored, then the directory is not shown, but all contents are shown.

-z
: Terminate entries with NUL, instead of LF. This implies the --porcelain=v1 output format if no other format is given.

--column[=<options>], --no-column

Display untracked files in columns. See configuration variable column.status for option syntax. --column and --no-column without options are equivalent to always and never respectively.

--ahead-behind, --no-ahead-behind

Display or do not display detailed ahead/behind counts for the branch relative to its upstream branch. Defaults to true.

--renames, --no-renames

Turn on/off rename detection regardless of user configuration. See also git-diff(1) --no-renames.

--find-renames[=<n>]

Turn on rename detection, optionally setting the similarity threshold. See also git-diff(1) --find-renames.

<pathspec>...

See the pathspec entry in gitglossary(7).

OUTPUT

The output from this command is designed to be used as a commit template comment. The default, long format, is designed to be human readable, verbose and descriptive. Its contents and format are subject to change at any time.

The paths mentioned in the output, unlike many other Git commands, are made relative to the current directory if you are working in a subdirectory (this is on purpose, to help cutting and pasting). See the status.relativePaths config option below.

Short Format

In the short-format, the status of each path is shown as one of these forms

<xy> <path>

<xy> <orig-path> -> <path>

where <orig-path> is where the renamed/copied contents came from. <orig-path> is only shown when the entry is renamed or copied. The <xy> is a two-letter status code XY.

The fields (including the ->) are separated from each other by a single space. If a filename contains whitespace or other nonprintable characters, that field will be quoted in the manner of a C string literal: surrounded by ASCII double quote (34) characters, and with interior special characters backslash-escaped.

There are three different types of states that are shown using this format, and each one uses the <xy> syntax differently:

? When a merge is occurring and the merge was successful, or outside of a merge situation, X shows the status of the index and Y shows the status of the working tree.

? When a merge conflict has occurred and has not yet been resolved, X and Y show the state introduced by each head of the merge, relative to the common ancestor. These paths are said to be unmerged.

? When a path is untracked, X and Y are always the same, since they are unknown to the index. ?? is used for untracked paths. Ignored files are not listed unless --ignored is used; if it is, ignored files are indicated by !!.

Note that the term merge here also includes rebases using the default --merge strategy, cherry-picks, and anything else using the merge machinery.

In the following table, these three classes are shown in separate sections, and these characters are used for X and Y fields for the first two sections that show tracked paths:

' '

unmodified

M

modified

T

file type changed (regular file, symbolic link or submodule)

A

added

D

deleted

R

renamed

C

copied (if config option status.renames is set to "copies")

U

updated but unmerged

??

? X ? Y ? Meaning ?

??

? ? ? ?

? ? [AMD] ? not updated ?

```
??????????????????????????????????????????????????
?         ?       ?                       ?
?    M     ? [ MTD] ? updated in index          ?
??????????????????????????????????????????????????
?         ?       ?                       ?
?    T     ? [ MTD] ? type changed in index     ?
??????????????????????????????????????????????????
?         ?       ?                       ?
?    A     ? [ MTD] ? added to index            ?
??????????????????????????????????????????????????
?         ?       ?                       ?
?    D     ?        ? deleted from index        ?
??????????????????????????????????????????????????
?         ?       ?                       ?
?    R     ? [ MTD] ? renamed in index          ?
??????????????????????????????????????????????????
?         ?       ?                       ?
?    C     ? [ MTD] ? copied in index           ?
??????????????????????????????????????????????????
?         ?       ?                       ?
? [MTARC]  ?        ? index and work tree       ?
?         ?       ? matches                   ?
??????????????????????????????????????????????????
?         ?       ?                       ?
? [ MTARC] ?    M    ? work tree changed since   ?
?         ?       ? index                     ?
??????????????????????????????????????????????????
?         ?       ?                       ?
? [ MTARC] ?    T    ? type changed in work tree ?
?         ?       ? since index               ?
??????????????????????????????????????????????????
?         ?       ?                       ?
? [ MTARC] ?    D    ? deleted in work tree      ?
```

```
????????????????????????????????????????????????
?        ?       ?                    ?
?        ?   R    ? renamed in work tree      ?
????????????????????????????????????????????????
?        ?       ?                    ?
?        ?   C    ? copied in work tree       ?
????????????????????????????????????????????????
?        ?       ?                    ?
?   D    ?   D    ? unmerged, both deleted    ?
????????????????????????????????????????????????
?        ?       ?                    ?
?   A    ?   U    ? unmerged, added by us     ?
????????????????????????????????????????????????
?        ?       ?                    ?
?   U    ?   D    ? unmerged, deleted by them ?
????????????????????????????????????????????????
?        ?       ?                    ?
?   U    ?   A    ? unmerged, added by them   ?
????????????????????????????????????????????????
?        ?       ?                    ?
?   D    ?   U    ? unmerged, deleted by us   ?
????????????????????????????????????????????????
?        ?       ?                    ?
?   A    ?   A    ? unmerged, both added      ?
????????????????????????????????????????????????
?        ?       ?                    ?
?   U    ?   U    ? unmerged, both modified   ?
????????????????????????????????????????????????
?        ?       ?                    ?
?   ?    ?   ?    ? untracked                 ?
????????????????????????????????????????????????
?        ?       ?                    ?
?   !    ?   !    ? ignored                   ?
```

??

Submodules have more state and instead report

M

the submodule has a different HEAD than recorded in the index

m

the submodule has modified content

?

the submodule has untracked files

This is since modified content or untracked files in a submodule cannot be added via git add in the superproject to prepare a commit.

m and ? are applied recursively. For example if a nested submodule in a submodule contains an untracked file, this is reported as ? as well.

If -b is used the short-format status is preceded by a line

<branchname> <tracking-info>

Porcelain Format Version 1

Version 1 porcelain format is similar to the short format, but is guaranteed not to change in a backwards-incompatible way between Git versions or based on user configuration. This makes it ideal for parsing by scripts. The description of the short format above also describes the porcelain format, with a few exceptions:

1. The user?s color.status configuration is not respected; color will always be off.
2. The user?s status.relativePaths configuration is not respected; paths shown will always be relative to the repository root.

There is also an alternate -z format recommended for machine parsing. In that format, the status field is the same, but some other things change. First, the -> is omitted from rename entries and the field order is reversed (e.g from -> to becomes to from). Second, a NUL (ASCII 0) follows each filename, replacing space as a field separator and the terminating newline (but a space still separates the status field from the first filename). Third, filenames containing special characters are not specially formatted; no quoting or backslash-escaping is performed.

Any submodule changes are reported as modified M instead of m or single ?.

Porcelain Format Version 2

Version 2 format adds more detailed information about the state of the worktree and changed items. Version 2 also defines an extensible set of easy to parse optional headers.

Header lines start with # and are added in response to specific command line arguments.

Parsers should ignore headers they don?t recognize.

Branch Headers

If --branch is given, a series of header lines are printed with information about the current branch.

Line	Notes
# branch.oid <commit> | (initial)	Current commit.
# branch.head <branch> | (detached)	Current branch.
# branch.upstream <upstream-branch>	If upstream is set.
# branch.ab +<ahead> -<behind>	If upstream is set and the commit is present.

Stash Information

If --show-stash is given, one line is printed showing the number of stash entries if non-zero:

stash <N>

Changed Tracked Entries

Following the headers, a series of lines are printed for tracked entries. One of three different line formats may be used to describe an entry depending on the type of change. Tracked entries are printed in an undefined order; parsers should allow for a mixture of the 3 line types in any order.

Ordinary changed entries have the following format:

1 <XY> <sub> <mH> <mI> <mW> <hH> <hI> <path>

Renamed or copied entries have the following format:

2 <XY> <sub> <mH> <mI> <mW> <hH> <hI> <X><score> <path><sep><origPath>

```
????????????????????????????????????????????????
? Field      ? Meaning                          ?
????????????????????????????????????????????????
?            ?                                  ?
? <XY>       ? A 2 character field containing   ?
?            ? the staged and unstaged XY values ?
?            ? described in the short format,   ?
?            ? with unchanged indicated by a "." ?
?            ? rather than a space.             ?
????????????????????????????????????????????????
?            ?                                  ?
? <sub>      ? A 4 character field describing   ?
?            ? the submodule state. "N..." when  ?
?            ? the entry is not a submodule.    ?
?            ? S<c><m><u> when the entry is a   ?
?            ? submodule.                       ?
?            ?                                  ?
?            ?                                  ?
?            ?           ?  <c> is "C" if the   ?
?            ?              commit changed;     ?
?            ?              otherwise ".".      ?
?            ?                                  ?
?            ?           ?  <m> is "M" if it    ?
?            ?              has tracked         ?
?            ?              changes; otherwise  ?
?            ?              ".".                ?
?            ?                                  ?
?            ?           ?  <u> is "U" if       ?
?            ?              there are           ?
?            ?              untracked changes;  ?
```

? ? otherwise ".". ?

??

? ? ?

? <mH> ? The octal file mode in HEAD. ?

??

? ? ?

? <mI> ? The octal file mode in the index. ?

??

? ? ?

? <mW> ? The octal file mode in the ?

? ? worktree. ?

??

? ? ?

? <hH> ? The object name in HEAD. ?

??

? ? ?

? <hI> ? The object name in the index. ?

??

? ? ?

? <X><score> ? The rename or copy score ?

? ? (denoting the percentage of ?

? ? similarity between the source and ?

? ? target of the move or copy). For ?

? ? example "R100" or "C75". ?

??

? ? ?

? <path> ? The pathname. In a renamed/copied ?

? ? entry, this is the target path. ?

??

? ? ?

? <sep> ? When the -z option is used, the 2 ?

? ? pathnames are separated with a ?

? ? NUL (ASCII 0x00) byte; otherwise, ?

? ? a TAB (ASCII 0x09) byte separates ?

? ? them. ?

??

? ? ?

? <origPath> ? The pathname in the commit at ?

? ? HEAD or in the index. This is ?

? ? only present in a renamed/copied ?

? ? entry, and tells where the ?

? ? renamed/copied contents came ?

? ? from. ?

??

Unmerged entries have the following format; the first character is a "u" to distinguish from ordinary changed entries.

u <XY> <sub> <m1> <m2> <m3> <mW> <h1> <h2> <h3> <path>

??

? Field ? Meaning ?

??

? ? ?

? <XY> ? A 2 character field describing ?

? ? the conflict type as described in ?

? ? the short format. ?

??

? ? ?

? <sub> ? A 4 character field describing ?

? ? the submodule state as described ?

? ? above. ?

??

? ? ?

? <m1> ? The octal file mode in stage 1. ?

??

? ? ?

? <m2> ? The octal file mode in stage 2. ?

??

? ? ?

? <m3> ? The octal file mode in stage 3. ?

??

? ? ?

? <mW> ? The octal file mode in the ?

? ? worktree. ?

??

? ? ?

? <h1> ? The object name in stage 1. ?

??

? ? ?

? <h2> ? The object name in stage 2. ?

??

? ? ?

? <h3> ? The object name in stage 3. ?

??

? ? ?

? <path> ? The pathname. ?

??

Other Items

Following the tracked entries (and if requested), a series of lines will be printed for untracked and then ignored items found in the worktree.

Untracked items have the following format:

? <path>

Ignored items have the following format:

! <path>

Pathname Format Notes and -z

When the -z option is given, pathnames are printed as is and without any quoting and lines are terminated with a NUL (ASCII 0x00) byte.

Without the -z option, pathnames with "unusual" characters are quoted as explained for the configuration variable core.quotePath (see git-config(1)).

CONFIGURATION

The command honors color.status (or status.color ? they mean the same thing and the latter

is kept for backward compatibility) and color.status.<slot> configuration variables to colorize its output.

If the config variable status.relativePaths is set to false, then all paths shown are relative to the repository root, not to the current directory.

If status.submoduleSummary is set to a non zero number or true (identical to -1 or an unlimited number), the submodule summary will be enabled for the long format 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 from the status command will be suppressed for all submodules when diff.ignoreSubmodules is set to all or only for those submodules where submodule.<name>.ignore=all. 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.

BACKGROUND REFRESH

By default, git status will automatically refresh the index, updating the cached stat information from the working tree and writing out the result. Writing out the updated index is an optimization that isn?t strictly necessary (status computes the values for itself, but writing them out is just to save subsequent programs from repeating our computation). When status is run in the background, the lock held during the write may conflict with other simultaneous processes, causing them to fail. Scripts running status in the background should consider using git --no-optional-locks status (see git(1) for details).

UNTRACKED FILES AND PERFORMANCE

git status can be very slow in large worktrees if/when it needs to search for untracked files and directories. There are many configuration options available to speed this up by either avoiding the work or making use of cached results from previous Git commands. There is no single optimum set of settings right for everyone. We?ll list a summary of the relevant options to help you, but before going into the list, you may want to run git status again, because your configuration may already be caching git status results, so it could be faster on subsequent runs.

? The --untracked-files=no flag or the status.showUntrackedFiles=no config (see above for both): indicate that git status should not report untracked files. This is the fastest option. git status will not list the untracked files, so you need to be careful to remember if you create any new files and manually git add them.

? advice.statusUoption=false (see git-config(1)): setting this variable to false disables

the warning message given when enumerating untracked files takes more than 2 seconds. In a large project, it may take longer and the user may have already accepted the trade off (e.g. using -uno may not be an acceptable option for the user), in which case, there is no point issuing the warning message, and in such a case, disabling the warning may be the best.

? core.untrackedCache=true (see git-update-index(1)): enable the untracked cache feature and only search directories that have been modified since the previous git status command. Git remembers the set of untracked files within each directory and assumes that if a directory has not been modified, then the set of untracked files within has not changed. This is much faster than enumerating the contents of every directory, but still not without cost, because Git still has to search for the set of modified directories. The untracked cache is stored in the .git/index file. The reduced cost of searching for untracked files is offset slightly by the increased size of the index and the cost of keeping it up-to-date. That reduced search time is usually worth the additional size.

? core.untrackedCache=true and core.fsmonitor=true or core.fsmonitor=<hook-command-pathname> (see git-update-index(1)): enable both the untracked cache and FSMonitor features and only search directories that have been modified since the previous git status command. This is faster than using just the untracked cache alone because Git can also avoid searching for modified directories. Git only has to enumerate the exact set of directories that have changed recently. While the FSMonitor feature can be enabled without the untracked cache, the benefits are greatly reduced in that case.

Note that after you turn on the untracked cache and/or FSMonitor features it may take a few git status commands for the various caches to warm up before you see improved command times. This is normal.

SEE ALSO

gitignore(5)

GIT

Part of the git(1) suite