



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

\$ man git-ls-files.1

GIT-LS-FILES(1)

Git Manual

GIT-LS-FILES(1)

NAME

git-ls-files - Show information about files in the index and the working tree

SYNOPSIS

git ls-files [-z] [-t] [-v] [-f]

[-c|--cached] [-d|--deleted] [-o|--others] [-i|--ignored]

[-s|--stage] [-u|--unmerged] [-k|--killed] [-m|--modified]

[--resolve-undo]

[--directory [--no-empty-directory]] [--eol]

[--deduplicate]

[-x <pattern>|--exclude=<pattern>]

[-X <file>|--exclude-from=<file>]

[--exclude-per-directory=<file>]

[--exclude-standard]

[--error-unmatch] [--with-tree=<tree-ish>]

[--full-name] [--recurse-submodules]

[--abbrev[=<n>]] [--format=<format>] [--] [<file>...]

DESCRIPTION

This command merges the file listing in the index with the actual working directory list, and shows different combinations of the two.

Several flags can be used to determine which files are shown, and each file may be printed multiple times if there are multiple entries in the index or if multiple statuses are applicable for the relevant file selection options.

OPTIONS

`-c, --cached`

Show all files cached in Git's index, i.e. all tracked files. (This is the default if no `-c/-s/-d/-o/-u/-k/-m/--resolve-undo` options are specified.)

`-d, --deleted`

Show files with an unstaged deletion

`-m, --modified`

Show files with an unstaged modification (note that an unstaged deletion also counts as an unstaged modification)

`-o, --others`

Show other (i.e. untracked) files in the output

`-i, --ignored`

Show only ignored files in the output. Must be used with either an explicit `-c` or `-o`.

When showing files in the index (i.e. when used with `-c`), print only those files matching an exclude pattern. When showing "other" files (i.e. when used with `-o`), show only those matched by an exclude pattern. Standard ignore rules are not automatically activated; therefore, at least one of the `--exclude*` options is required.

`-s, --stage`

Show staged contents' mode bits, object name and stage number in the output.

`--directory`

If a whole directory is classified as "other", show just its name (with a trailing slash) and not its whole contents. Has no effect without `-o/--others`.

`--no-empty-directory`

Do not list empty directories. Has no effect without `--directory`.

`-u, --unmerged`

Show information about unmerged files in the output, but do not show any other tracked files (forces `--stage`, overrides `--cached`).

`-k, --killed`

Show untracked files on the filesystem that need to be removed due to file/directory conflicts for tracked files to be able to be written to the filesystem.

`--resolve-undo`

Show files having resolve-undo information in the index together with their resolve-undo information. (resolve-undo information is what is used to implement "git checkout -m

\$PATH", i.e. to recreate merge conflicts that were accidentally resolved)

-Z

\0 line termination on output and do not quote filenames. See OUTPUT below for more information.

--deduplicate

When only filenames are shown, suppress duplicates that may come from having multiple stages during a merge, or giving --deleted and --modified option at the same time. When any of the -t, --unmerged, or --stage option is in use, this option has no effect.

-x <pattern>, --exclude=<pattern>

Skip untracked files matching pattern. Note that pattern is a shell wildcard pattern.

See EXCLUDE PATTERNS below for more information.

-X <file>, --exclude-from=<file>

Read exclude patterns from <file>; 1 per line.

--exclude-per-directory=<file>

Read additional exclude patterns that apply only to the directory and its subdirectories in <file>. If you are trying to emulate the way Porcelain commands work, using the --exclude-standard option instead is easier and more thorough.

--exclude-standard

Add the standard Git exclusions: .git/info/exclude, .gitignore in each directory, and the user's global exclusion file.

--error-unmatch

If any <file> does not appear in the index, treat this as an error (return 1).

--with-tree=<tree-ish>

When using --error-unmatch to expand the user supplied <file> (i.e. path pattern) arguments to paths, pretend that paths which were removed in the index since the named <tree-ish> are still present. Using this option with -s or -u options does not make any sense.

-t

Show status tags together with filenames. Note that for scripting purposes, git-status(1) --porcelain and git-diff-files(1) --name-status are almost always superior alternatives; users should look at git-status(1) --short or git-diff(1) --name-status for more user-friendly alternatives.

This option provides a reason for showing each filename, in the form of a status tag

(which is followed by a space and then the filename). The status tags are all single characters from the following list:

H

tracked file that is not either unmerged or skip-worktree

S

tracked file that is skip-worktree

M

tracked file that is unmerged

R

tracked file with unstaged removal/deletion

C

tracked file with unstaged modification/change

K

untracked paths which are part of file/directory conflicts which prevent checking out tracked files

?

untracked file

U

file with resolve-undo information

-v

Similar to -t, but use lowercase letters for files that are marked as assume unchanged (see git-update-index(1)).

-f

Similar to -t, but use lowercase letters for files that are marked as fsmonitor valid (see git-update-index(1)).

--full-name

When run from a subdirectory, the command usually outputs paths relative to the current directory. This option forces paths to be output relative to the project top directory.

--recurse-submodules

Recursively calls ls-files on each active submodule in the repository. Currently there is only support for the --cached and --stage modes.

--abbrev[=<n>]

Instead of showing the full 40-byte hexadecimal object lines, show the shortest prefix

that is at least <n> hexdigits long that uniquely refers the object. Non default number of digits can be specified with --abbrev=<n>.

--debug

After each line that describes a file, add more data about its cache entry. This is intended to show as much information as possible for manual inspection; the exact format may change at any time.

--eol

Show <eolinfo> and <eolattr> of files. <eolinfo> is the file content identification used by Git when the "text" attribute is "auto" (or not set and core.autocrlf is not false).

<eolinfo> is either "-text", "none", "lf", "crlf", "mixed" or "".

"" means the file is not a regular file, it is not in the index or not accessible in the working tree.

<eolattr> is the attribute that is used when checking out or committing, it is either "", "-text", "text", "text=auto", "text eol=lf", "text eol=crlf". Since Git 2.10

"text=auto eol=lf" and "text=auto eol=crlf" are supported.

Both the <eolinfo> in the index ("i/<eolinfo>") and in the working tree ("w/<eolinfo>") are shown for regular files, followed by the ("attr/<eolattr>").

--sparse

If the index is sparse, show the sparse directories without expanding to the contained files. Sparse directories will be shown with a trailing slash, such as "x/" for a sparse directory "x".

--format=<format>

A string that interpolates %(fieldname) from the result being shown. It also interpolates %% to %, and %xXX where XX are hex digits interpolates to character with hex code XX; for example %x00 interpolates to \0 (NUL), %x09 to \t (TAB) and %x0a to \n (LF). --format cannot be combined with -s, -o, -k, -t, --resolve-undo and --eol.

--

Do not interpret any more arguments as options.

<file>

Files to show. If no files are given all files which match the other specified criteria are shown.

OUTPUT

git ls-files just outputs the filenames unless --stage is specified in which case it

outputs:

```
[<tag> ]<mode> <object> <stage> <file>
```

git ls-files --eol will show

```
i/<eolinfo><SPACES>w/<eolinfo><SPACES>attr/<eolattr><SPACE*><TAB><file>
```

git ls-files --unmerged and git ls-files --stage can be used to examine detailed information on unmerged paths.

For an unmerged path, instead of recording a single mode/SHA-1 pair, the index records up to three such pairs; one from tree O in stage 1, A in stage 2, and B in stage 3. This information can be used by the user (or the porcelain) to see what should eventually be recorded at the path. (see git-read-tree(1) for more information on state)

Without the -z option, pathnames with "unusual" characters are quoted as explained for the configuration variable core.quotePath (see git-config(1)). Using -z the filename is output verbatim and the line is terminated by a NUL byte.

It is possible to print in a custom format by using the --format option, which is able to interpolate different fields using a %(fieldname) notation. For example, if you only care about the "objectname" and "path" fields, you can execute with a specific "--format" like

```
git ls-files --format='%(objectname) %(path)'
```

FIELD NAMES

The way each path is shown can be customized by using the --format=<format> option, where the %(fieldname) in the <format> string for various aspects of the index entry are interpolated. The following "fieldname" are understood:

objectmode

The mode of the file which is recorded in the index.

objecttype

The object type of the file which is recorded in the index.

objectname

The name of the file which is recorded in the index.

objectsize[:padded]

The object size of the file which is recorded in the index ("- " if the object is a commit or tree). It also supports a padded format of size with "%(objectsize:padded)".

stage

The stage of the file which is recorded in the index.

eolinfo:index, eolinfo:worktree

The <eolinfo> (see the description of the --eol option) of the contents in the index or in the worktree for the path.

eolattr

The <eolattr> (see the description of the --eol option) that applies to the path.

path

The pathname of the file which is recorded in the index.

EXCLUDE PATTERNS

git ls-files can use a list of "exclude patterns" when traversing the directory tree and finding files to show when the flags --others or --ignored are specified. gitignore(5) specifies the format of exclude patterns.

These exclude patterns can be specified from the following places, in order:

1. The command-line flag --exclude=<pattern> specifies a single pattern. Patterns are ordered in the same order they appear in the command line.
2. The command-line flag --exclude-from=<file> specifies a file containing a list of patterns. Patterns are ordered in the same order they appear in the file.
3. The command-line flag --exclude-per-directory=<name> specifies a name of the file in each directory git ls-files examines, normally .gitignore. Files in deeper directories take precedence. Patterns are ordered in the same order they appear in the files.

A pattern specified on the command line with --exclude or read from the file specified with --exclude-from is relative to the top of the directory tree. A pattern read from a file specified by --exclude-per-directory is relative to the directory that the pattern file appears in.

Generally, you should be able to use --exclude-standard when you want the exclude rules applied the same way as what Porcelain commands do. To emulate what --exclude-standard specifies, you can give --exclude-per-directory=.gitignore, and then specify:

1. The file specified by the core.excludesfile configuration variable, if exists, or the \$XDG_CONFIG_HOME/git/ignore file.
2. The \$GIT_DIR/info/exclude file.

via the --exclude-from= option.

SEE ALSO

git-read-tree(1), gitignore(5)

GIT

Part of the git(1) suite

