NAME

git-update-index - Register file contents in the working tree to the index

SYNOPSIS

```
git update-index
             [--add] [--remove | --force-remove] [--replace]
             [--refresh] [-q] [--unmerged] [--ignore-missing]
             [(--cacheinfo <mode>,<object>,<file>)...]
             [--chmod=(+|-)x]
             [--[no-]assume-unchanged]
             [--[no-]skip-worktree]
             [--[no-]ignore-skip-worktree-entries]
             [--[no-]fsmonitor-valid]
             [--ignore-submodules]
             [--[no-]split-index]
             [--[no-|test-|force-]untracked-cache]
             [--[no-]fsmonitor]
             [--really-refresh] [--unresolve] [--again | -g]
             [--info-only] [--index-info]
             [-z] [--stdin] [--index-version <n>]
             [--verbose]
             [--] [<file>...]
```

DESCRIPTION

Modifies the index. Each file mentioned is updated into the index and any unmerged or needs updating state is cleared.

See also git-add(1) for a more user-friendly way to do some of the most common operations on the index.

The way git update-index handles files it is told about can be modified using the various options:

OPTIONS

--add
If a specified file isn't in the index already then it's added. Default behaviour is to ignore new files.

--remove
If a specified file is in the index but is missing then it's removed. Default behavior is to ignore removed file.

--refresh
Looks at the current index and checks to see if merges or updates are needed by checking stat() information.

-q
Quiet. If --refresh finds that the index needs an update, the default behavior is to error out. This option makes git update-index continue anyway.

--ignore-submodules
Do not try to update submodules. This option is only respected when passed before --refresh.

--unmerged
If --refresh finds unmerged changes in the index, the default behavior is to error out. This option makes git update-index continue anyway.

--ignore-missing
Ignores missing files during a --refresh

--cacheinfo <mode>,<object>,<path>, --cacheinfo <mode> <object> <path>
Directly insert the specified info into the index. For backward

compatibility, you can also give these three arguments as three separate parameters, but new users are encouraged to use a single-parameter form.

--index-info
Read index information from stdin.

--chmod=(+|-)x
Set the execute permissions on the updated files.

--[no-]assume-unchanged
When this flag is specified, the object names recorded for the paths are not updated. Instead, this option sets/unsets the "assume unchanged" bit for the paths. When the "assume unchanged" bit is on, the user promises not to change the file and allows Git to assume that the working tree file matches what is recorded in the index. If you want to change the working tree file, you need to unset the bit to tell Git. This is sometimes helpful when working with a big project on a filesystem that has very slow lstat(2) system call (e.g. cifs).

Git will fail (gracefully) in case it needs to modify this file in the index e.g. when merging in a commit; thus, in case the assumed-untracked file is changed upstream, you will need to handle the situation manually.

--really-refresh
Like --refresh, but checks stat information unconditionally, without regard to the "assume unchanged" setting.

--[no-]skip-worktree
When one of these flags is specified, the object name recorded for the paths are not updated. Instead, these options set and unset the "skip-worktree" bit for the paths. See section "Skip-worktree bit" below for more information.

--[no-]ignore-skip-worktree-entries
Do not remove skip-worktree (AKA "index-only") entries even when the --remove option was specified.

--[no-]fsmonitor-valid
When one of these flags is specified, the object name recorded for the paths are not updated. Instead, these options set and unset the "fsmonitor valid" bit for the paths. See section "File System Monitor" below for more information.

-g, --again
Runs git update-index itself on the paths whose index entries are different from those from the HEAD commit.

--unresolve
Restores the unmerged or needs updating state of a file during a merge if it was cleared by accident.

--info-only
Do not create objects in the object database for all <file> arguments that follow this flag; just insert their object IDs into the index.

--force-remove
Remove the file from the index even when the working directory still has such a file. (Implies --remove.)

--replace
By default, when a file path exists in the index, git update-index refuses an attempt to add path/file. Similarly if a file path/file exists, a file path cannot be added. With --replace flag, existing entries that conflict with the entry being added are automatically

removed with warning messages.

--stdin
Instead of taking list of paths from the command line, read list of paths from the standard input. Paths are separated by LF (i.e. one path per line) by default.

--verbose
Report what is being added and removed from index.

--index-version <n>
Write the resulting index out in the named on-disk format version. Supported versions are 2, 3 and 4. The current default version is 2 or 3, depending on whether extra features are used, such as git add -N.

Version 4 performs a simple pathname compression that reduces index size by 30%-50% on large repositories, which results in faster load time. Version 4 is relatively young (first released in 1.8.0 in October 2012). Other Git implementations such as JGit and libgit2 may not support it yet.

-z
Only meaningful with --stdin or --index-info; paths are separated with NUL character instead of LF.

--split-index, --no-split-index
Enable or disable split index mode. If split-index mode is already enabled and --split-index is given again, all changes in $GIT_DIR/index are pushed back to the shared index file.

These options take effect whatever the value of the core.splitIndex configuration variable (see git-config(1)). But a warning is emitted when the change goes against the configured value, as the configured value will take effect next time the index is read and this will remove the intended effect of the option.

--untracked-cache, --no-untracked-cache
Enable or disable untracked cache feature. Please use --test-untracked-cache before enabling it.

These options take effect whatever the value of the core.untrackedCache configuration variable (see git-config(1)). But a warning is emitted when the change goes against the configured value, as the configured value will take effect next time the index is read and this will remove the intended effect of the option.

--test-untracked-cache
Only perform tests on the working directory to make sure untracked cache can be used. You have to manually enable untracked cache using --untracked-cache or --force-untracked-cache or the core.untrackedCache configuration variable afterwards if you really want to use it. If a test fails the exit code is 1 and a message explains what is not working as needed, otherwise the exit code is 0 and OK is printed.

--force-untracked-cache
Same as --untracked-cache. Provided for backwards compatibility with older versions of Git where --untracked-cache used to imply --test-untracked-cache but this option would enable the extension unconditionally.

--fsmonitor, --no-fsmonitor
Enable or disable files system monitor feature. These options take effect whatever the value of the core.fsmonitor configuration variable (see git-config(1)). But a warning is emitted when the change goes against the configured value, as the configured value will take effect next time the index is read and this will remove

the intended effect of the option.

--
Do not interpret any more arguments as options.

<file>
Files to act on. Note that files beginning with . are discarded. This includes ./file and dir/./file. If you don't want this, then use cleaner names. The same applies to directories ending / and paths with //

USING --REFRESH

--refresh does not calculate a new sha1 file or bring the index up to date for mode/content changes. But what it does do is to "re-match" the stat information of a file with the index, so that you can refresh the index for a file that hasn't been changed but where the stat entry is out of date.

For example, you'd want to do this after doing a git read-tree, to link up the stat index details with the proper files.

USING --CACHEINFO OR --INFO-ONLY

--cacheinfo is used to register a file that is not in the current working directory. This is useful for minimum-checkout merging.

To pretend you have a file at path with mode and sha1, say:

```
$ git update-index --add --cacheinfo <mode>,<sha1>,<path>
```

--info-only is used to register files without placing them in the object database. This is useful for status-only repositories.

Both --cacheinfo and --info-only behave similarly: the index is updated but the object database isn't. --cacheinfo is useful when the object is in the database but the file isn't available locally. --info-only is useful when the file is available, but you do not wish to update the object database.

USING --INDEX-INFO

--index-info is a more powerful mechanism that lets you feed multiple entry definitions from the standard input, and designed specifically for scripts. It can take inputs of three formats:

1. mode SP type SP sha1 TAB path

   This format is to stuff git ls-tree output into the index.

2. mode SP sha1 SP stage TAB path

   This format is to put higher order stages into the index file and matches git ls-files --stage output.

3. mode SP sha1 TAB path

   This format is no longer produced by any Git command, but is and will continue to be supported by update-index --index-info.

To place a higher stage entry to the index, the path should first be removed by feeding a mode=0 entry for the path, and then feeding necessary input lines in the third format.

For example, starting with this index:

```
$ git ls-files -s
100644 8a1218a1024a212bb3db30becd860315f9f3ac52 0       frotz
```

you can feed the following input to --index-info:

```
$ git update-index --index-info
0 0000000000000000000000000000000000000000      frotz
100644 8a1218a1024a212bb3db30becd860315f9f3ac52 1       frotz
100755 8a1218a1024a212bb3db30becd860315f9f3ac52 2       frotz
```

The first line of the input feeds 0 as the mode to remove the path; the SHA-1 does not matter as long as it is well formatted. Then the second and third line feeds stage 1 and stage 2 entries for that path. After the above, we would end up with this:

```
$ git ls-files -s
100644 8a1218a1024a212bb3db30becd860315f9f3ac52 1       frotz
100755 8a1218a1024a212bb3db30becd860315f9f3ac52 2       frotz
```

USING "ASSUME UNCHANGED" BIT

Many operations in Git depend on your filesystem to have an efficient lstat(2) implementation, so that st_mtime information for working tree files can be cheaply checked to see if the file contents have changed from the version recorded in the index file. Unfortunately, some filesystems have inefficient lstat(2). If your filesystem is one of them, you can set "assume unchanged" bit to paths you have not changed to cause Git not to do this check. Note that setting this bit on a path does not mean Git will check the contents of the file to see if it has changed -- it makes Git to omit any checking and assume it has not changed. When you make changes to working tree files, you have to explicitly tell Git about it by dropping "assume unchanged" bit, either before or after you modify them.

In order to set "assume unchanged" bit, use --assume-unchanged option. To unset, use --no-assume-unchanged. To see which files have the "assume unchanged" bit set, use git ls-files -v (see git-ls-files(1)).

The command looks at core.ignorestat configuration variable. When this is true, paths updated with git update-index paths... and paths updated with other Git commands that update both index and working tree (e.g. git apply --index, git checkout-index -u, and git read-tree -u) are automatically marked as "assume unchanged". Note that "assume unchanged" bit is not set if git update-index --refresh finds the working tree file matches the index (use git update-index --really-refresh if you want to mark them as "assume unchanged").

EXAMPLES

To update and refresh only the files already checked out:

```
$ git checkout-index -n -f -a && git update-index --ignore-missing --refresh
```

On an inefficient filesystem with core.ignorestat set

```
$ git update-index --really-refresh              (1)
$ git update-index --no-assume-unchanged foo.c   (2)
$ git diff --name-only                           (3)
$ edit foo.c
$ git diff --name-only                           (4)
M foo.c
$ git update-index foo.c                         (5)
$ git diff --name-only                           (6)
$ edit foo.c
$ git diff --name-only                           (7)
$ git update-index --no-assume-unchanged foo.c   (8)
$ git diff --name-only                           (9)
M foo.c
```

1. forces lstat(2) to set "assume unchanged" bits for paths that match index.
2. mark the path to be edited.
3. this does lstat(2) and finds index matches the path.
4. this does lstat(2) and finds index does not match the path.
5. registering the new version to index sets "assume unchanged"

bit.
6. and it is assumed unchanged.
7. even after you edit it.
8. you can tell about the change after the fact.
9. now it checks with lstat(2) and finds it has been changed.

SKIP-WORKTREE BIT

Skip-worktree bit can be defined in one (long) sentence: When reading an entry, if it is marked as skip-worktree, then Git pretends its working directory version is up to date and read the index version instead.

To elaborate, "reading" means checking for file existence, reading file attributes or file content. The working directory version may be present or absent. If present, its content may match against the index version or not. Writing is not affected by this bit, content safety is still first priority. Note that Git can update working directory file, that is marked skip-worktree, if it is safe to do so (i.e. working directory version matches index version)

Although this bit looks similar to assume-unchanged bit, its goal is different from assume-unchanged bit's. Skip-worktree also takes precedence over assume-unchanged bit when both are set.

SPLIT INDEX

This mode is designed for repositories with very large indexes, and aims at reducing the time it takes to repeatedly write these indexes.

In this mode, the index is split into two files, $GIT_DIR/index and $GIT_DIR/sharedindex.<SHA-1>. Changes are accumulated in $GIT_DIR/index, the split index, while the shared index file contains all index entries and stays unchanged.

All changes in the split index are pushed back to the shared index file when the number of entries in the split index reaches a level specified by the splitIndex.maxPercentChange config variable (see git-config(1)).

Each time a new shared index file is created, the old shared index files are deleted if their modification time is older than what is specified by the splitIndex.sharedIndexExpire config variable (see git-config(1)).

To avoid deleting a shared index file that is still used, its modification time is updated to the current time every time a new split index based on the shared index file is either created or read from.

UNTRACKED CACHE

This cache is meant to speed up commands that involve determining untracked files such as git status.

This feature works by recording the mtime of the working tree directories and then omitting reading directories and stat calls against files in those directories whose mtime hasn't changed. For this to work the underlying operating system and file system must change the st_mtime field of directories if files in the directory are added, modified or deleted.

You can test whether the filesystem supports that with the --test-untracked-cache option. The --untracked-cache option used to implicitly perform that test in older versions of Git, but that's no longer the case.

If you want to enable (or disable) this feature, it is easier to use the core.untrackedCache configuration variable (see git-config(1)) than using the --untracked-cache option to git update-index in each repository, especially if you want to do so across all repositories you use, because you can set the configuration variable to true (or false) in your $HOME/.gitconfig just once and have it affect all repositories

you touch.

When the core.untrackedCache configuration variable is changed, the untracked cache is added to or removed from the index the next time a command reads the index; while when --[no-|force-]untracked-cache are used, the untracked cache is immediately added to or removed from the index.

Before 2.17, the untracked cache had a bug where replacing a directory with a symlink to another directory could cause it to incorrectly show files tracked by git as untracked. See the "status: add a failing test showing a core.untrackedCache bug" commit to git.git. A workaround for that is (and this might work for other undiscovered bugs in the future):

```
$ git -c core.untrackedCache=false status
```

This bug has also been shown to affect non-symlink cases of replacing a directory with a file when it comes to the internal structures of the untracked cache, but no case has been reported where this resulted in wrong "git status" output.

There are also cases where existing indexes written by git versions before 2.17 will reference directories that don't exist anymore, potentially causing many "could not open directory" warnings to be printed on "git status". These are new warnings for existing issues that were previously silently discarded.

As with the bug described above the solution is to one-off do a "git status" run with core.untrackedCache=false to flush out the leftover bad data.

FILE SYSTEM MONITOR

This feature is intended to speed up git operations for repos that have large working directories.

It enables git to work together with a file system monitor (see the "fsmonitor-watchman" section of githooks(5)) that can inform it as to what files have been modified. This enables git to avoid having to lstat() every file to find modified files.

When used in conjunction with the untracked cache, it can further improve performance by avoiding the cost of scanning the entire working directory looking for new files.

If you want to enable (or disable) this feature, it is easier to use the core.fsmonitor configuration variable (see git-config(1)) than using the --fsmonitor option to git update-index in each repository, especially if you want to do so across all repositories you use, because you can set the configuration variable in your $HOME/.gitconfig just once and have it affect all repositories you touch.

When the core.fsmonitor configuration variable is changed, the file system monitor is added to or removed from the index the next time a command reads the index. When --[no-]fsmonitor are used, the file system monitor is immediately added to or removed from the index.

CONFIGURATION

The command honors core.filemode configuration variable. If your repository is on a filesystem whose executable bits are unreliable, this should be set to false (see git-config(1)). This causes the command to ignore differences in file modes recorded in the index and the file mode on the filesystem if they differ only on executable bit. On such an unfortunate filesystem, you may need to use git update-index --chmod=.

Quite similarly, if core.symlinks configuration variable is set to false (see git-config(1)), symbolic links are checked out as plain

files, and this command does not modify a recorded file mode from symbolic link to regular file.

The command looks at core.ignorestat configuration variable. See Using "assume unchanged" bit section above.

The command also looks at core.trustctime configuration variable. It can be useful when the inode change time is regularly modified by something outside Git (file system crawlers and backup systems use ctime for marking files processed) (see git-config(1)).

The untracked cache extension can be enabled by the core.untrackedCache configuration variable (see git-config(1)).

NOTES

Users often try to use the assume-unchanged and skip-worktree bits to tell Git to ignore changes to files that are tracked. This does not work as expected, since Git may still check working tree files against the index when performing certain operations. In general, Git does not provide a way to ignore changes to tracked files, so alternate solutions are recommended.

For example, if the file you want to change is some sort of config file, the repository can include a sample config file that can then be copied into the ignored name and modified. The repository can even include a script to treat the sample file as a template, modifying and copying it automatically.

SEE ALSO

git-config(1), git-add(1), git-ls-files(1)

GIT

Part of the git(1) suite