NAME

git−lfs−prune − Delete old LFS files from local storage

SYNOPSIS

git lfs prune *options*

DESCRIPTION

Deletes local copies of LFS files which are old, thus freeing up disk space. Prune operates by enumerating all the locally stored objects, and then deleting any which are not referenced by at least ONE of the following:

- the current checkout
- all existing stashes
- a 'recent branch'; see *RECENT FILES*
- a 'recent commit' on the current branch or recent branches; see *RECENT FILES*
- a commit which has not been pushed; see *UNPUSHED LFS FILES*
- any other worktree checkouts; see git−worktree(1)

In general terms, prune will delete files you're not currently using and which are not 'recent', so long as they've been pushed i.e. the local copy is not the only one.

The reflog is not considered, only commits. Therefore LFS objects that are only referenced by orphaned commits are always deleted.

Note: you should not run **git lfs prune** if you have different repositories sharing the same custom storage directory; see git−lfs−config(1) for more details about **lfs.storage** option.

OPTIONS

- **−−dry−run −d** Don't actually delete anything, just report on what would have been done
- **−−force −f** Prune all objects except unpushed objects, including objects required for currently checked out refs. Implies **−−recent**.
- **−−recent** Prune even objects that would normally be preserved by the configuration options specified below in *RECENT FILES*.
- **−−verify−remote −c** Contact the remote and check that copies of the files we would delete definitely exist before deleting. See *VERIFY REMOTE*.
- **−−no−verify−remote** Disables remote verification if lfs.pruneverifyremotealways was enabled in settings. See *VERIFY REMOTE*.
- **−−verbose −v** Report the full detail of what is/would be deleted.

RECENT FILES

Prune won't delete LFS files referenced by 'recent' commits, in case you want to use them again without having to download. The definition of 'recent' is derived from the one used by git−lfs−fetch(1) to download recent objects with the **−−recent** option, with an offset of a number of days (default 3) to ensure that we always keep files you download for a few days.

Here are the git−config(1) settings that control this behaviour:

- **lfs.pruneoffsetdays**
  The number of extra days added to the fetch recent settings when using them to decide when to prune. So for a reference to be considered old enough to prune, it has to be this many days older than the oldest reference that would be downloaded via **git lfs fetch −−recent**. Only used if the relevant fetch recent 'days' setting is non−zero. Default 3 days.

- **lfs.fetchrecentrefsdays**
  lfs.fetchrecentremoterefs
  lfs.fetchrecentcommitsdays
  These have the same meaning as git−lfs−fetch(1) with the **−−recent** option, they are used as a base for the offset above. Anything which falls outside of this offsetted window is considered old enough to prune. If a day value is zero, that condition is not used at all to retain objects and they will be pruned.

UNPUSHED LFS FILES

When the only copy of an LFS file is local, and it is still reachable from any reference, that file can never be pruned, regardless of how old it is.

To determine whether an LFS file has been pushed, we check the difference between local refs and remote refs; where the local ref is ahead, any LFS files referenced in those commits is unpushed and will not be deleted. This works because the LFS pre−push hook always ensures that LFS files are pushed before the remote branch is updated.

See *DEFAULT REMOTE*, for which remote is considered 'pushed' for pruning purposes.

VERIFY REMOTE

The **−−verify−remote** option calls the remote to ensure that any LFS files to be deleted have copies on the remote before actually deleting them.

Usually the check performed by *UNPUSHED LFS FILES* is enough to determine that files have been pushed, but if you want to be extra sure at the expense of extra overhead you can make prune actually call the remote API and verify the presence of the files you're about to delete locally. See *DEFAULT REMOTE* for which remote is checked.

You can make this behaviour the default by setting **lfs.pruneverifyremotealways** to true.

In addition to the overhead of calling the remote, using this option also requires prune to distinguish between totally unreachable files (e.g. those that were added to the index but never committed, or referenced only by orphaned commits), and files which are still referenced, but by commits which are prunable. This makes the prune process take longer.

DEFAULT REMOTE

When identifying *UNPUSHED LFS FILES* and performing *VERIFY REMOTE*, a single remote, 'origin', is normally used as the reference. This one remote is considered canonical; even if you use multiple remotes, you probably want to retain your local copies until they've made it to that remote. 'origin' is used by default because that will usually be a main central repo, or your fork of it − in both cases that's a valid remote backup of your work. If origin doesn't exist then by default nothing will be pruned because everything is treated as 'unpushed'.

You can alter the remote via git config: **lfs.pruneremotetocheck**. Set this to a different remote name to check that one instead of 'origin'.

SEE ALSO

git−lfs−fetch(1)

Part of the git−lfs(1) suite.