NAME

git-gc - Cleanup unnecessary files and optimize the local repository

SYNOPSIS

git gc [--aggressive] [--auto] [--quiet] [--prune=<date> | --no-prune] [--force] [--keep-largest-pack]

DESCRIPTION

Runs a number of housekeeping tasks within the current repository, such as compressing file revisions (to reduce disk space and increase performance), removing unreachable objects which may have been created from prior invocations of git add, packing refs, pruning reflog, rerere metadata or stale working trees. May also update ancillary indexes such as the commit-graph.

When common porcelain operations that create objects are run, they will check whether the repository has grown substantially since the last maintenance, and if so run git gc automatically. See gc.auto below for how to disable this behavior.

Running git gc manually should only be needed when adding objects to a repository without regularly running such porcelain commands, to do a one-off repository optimization, or e.g. to clean up a suboptimal mass-import. See the "PACKFILE OPTIMIZATION" section in git-fast-import(1) for more details on the import case.

OPTIONS

--aggressive

Usually git gc runs very quickly while providing good disk space utilization and performance. This option will cause git gc to more aggressively optimize the repository at the expense of taking much more time. The effects of this optimization are mostly persistent. See the "AGGRESSIVE" section below for details.

--auto

With this option, git gc checks whether any housekeeping is required; if not, it exits without performing any work.

See the gc.auto option in the "CONFIGURATION" section below for how this heuristic works.

Once housekeeping is triggered by exceeding the limits of configuration options such as gc.auto and gc.autoPackLimit, all other housekeeping tasks (e.g. rerere, working trees, reflog...) will be performed as well.

--prune=<date>

Prune loose objects older than date (default is 2 weeks ago, overridable by the config variable gc.pruneExpire). --prune=now prunes loose objects regardless of their age and increases the risk of corruption if another process is writing to the repository concurrently; see "NOTES" below. --prune is on by default.

--no-prune

Do not prune any loose objects.

--quiet

Suppress all progress reports.

--force

Force git gc to run even if there may be another git gc instance running on this repository.

--keep-largest-pack

All packs except the largest pack and those marked with a .keep files are consolidated into a single pack. When this option is

used, gc.bigPackThreshold is ignored.

AGGRESSIVE

When the --aggressive option is supplied, git-repack(1) will be invoked with the -f flag, which in turn will pass --no-reuse-delta to git-pack-objects(1). This will throw away any existing deltas and re-compute them, at the expense of spending much more time on the repacking.

The effects of this are mostly persistent, e.g. when packs and loose objects are coalesced into one another pack the existing deltas in that pack might get re-used, but there are also various cases where we might pick a sub-optimal delta from a newer pack instead.

Furthermore, supplying --aggressive will tweak the --depth and --window options passed to git-repack(1). See the gc.aggressiveDepth and gc.aggressiveWindow settings below. By using a larger window size we're more likely to find more optimal deltas.

It's probably not worth it to use this option on a given repository without running tailored performance benchmarks on it. It takes a lot more time, and the resulting space/delta optimization may or may not be worth it. Not using this at all is the right trade-off for most users and their repositories.

CONFIGURATION

The below documentation is the same as what's found in git-config(1):

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-largest-pack except that all packs that meet the threshold are kept, not just the largest 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-largest-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).

NOTES

git gc tries very hard not to delete objects that are referenced anywhere in your repository. In particular, it will keep not only objects referenced by your current set of branches and tags, but also objects referenced by the index, remote-tracking branches, reflogs (which may reference commits in branches that were later amended or rewound), and anything else in the refs/* namespace. Note that a note (of the kind created by git notes) attached to an object does not contribute in keeping the object alive. If you are expecting some objects to be deleted and they aren't, check all of those locations and decide whether it makes sense in your case to remove those references.

On the other hand, when git gc runs concurrently with another process, there is a risk of it deleting an object that the other process is using but hasn't created a reference to. This may just cause the other process to fail or may corrupt the repository if the other process later adds a reference to the deleted object. Git has two features that significantly mitigate this problem:

1. Any object with modification time newer than the --prune date is kept, along with everything reachable from it.

2. Most operations that add an object to the database update the modification time of the object if it is already present so that #1 applies.

However, these features fall short of a complete solution, so users who run commands concurrently have to live with some risk of corruption (which seems to be low in practice).

HOOKS

The git gc --auto command will run the pre-auto-gc hook. See githooks(5) for more information.

SEE ALSO

git-prune(1) git-reflog(1) git-repack(1) git-rerere(1)

GIT

Part of the git(1) suite