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.

Users are encouraged to run this task on a regular basis within each repository to maintain good disk space utilization and good operating performance.

Some git commands may automatically run *git gc*; see the **−−auto** flag below for details. If you know what you're doing and all you want is to disable this behavior permanently without further considerations, just do:

$ git config −−global gc.auto 0

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 persistent, so this option only needs to be used occasionally; every few hundred changesets or so.

−−auto
: With this option, *git gc* checks whether any housekeeping is required; if not, it exits without performing any work. Some git commands run **git gc −−auto** after performing operations that could create many loose objects. Housekeeping is required if there are too many loose objects or too many packs in the repository.

  If the number of loose objects exceeds the value of the **gc.auto** configuration variable, then all loose objects are combined into a single pack using **git repack −d −l**. Setting the value of **gc.auto** to 0 disables automatic packing of loose objects.

  If the number of packs exceeds the value of **gc.autoPackLimit**, then existing packs (except those marked with a **.keep** file or over **gc.bigPackThreshold** limit) are consolidated into a single pack by using the **−A** option of *git repack*. If the amount of memory is estimated not enough for **git repack** to run smoothly and **gc.bigPackThreshold** is not set, the largest pack will also be excluded (this is the equivalent of running **git gc** with **−−keep−base−pack**). Setting **gc.autoPackLimit** to 0 disables automatic consolidation of packs.

  If houskeeping is required due to many loose objects or packs, 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=all 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.

CONFIGURATION

The optional configuration variable **gc.reflogExpire** can be set to indicate how long historical entries within each branch's reflog should remain available in this repository. The setting is expressed as a length of time, for example *90 days* or *3 months*. It defaults to *90 days*.

The optional configuration variable **gc.reflogExpireUnreachable** can be set to indicate how long historical reflog entries which are not part of the current branch should remain available in this repository. 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. This option defaults to *30 days*.

The above two configuration variables can be given to a pattern. For example, this sets non−default expiry values only to remote−tracking branches:

```
[gc "refs/remotes/*"]
    reflogExpire = never
    reflogExpireUnreachable = 3 days
```

The optional configuration variable **gc.rerereResolved** indicates how long records of conflicted merge you resolved earlier are kept. This defaults to 60 days.

The optional configuration variable **gc.rerereUnresolved** indicates how long records of conflicted merge you have not resolved are kept. This defaults to 15 days.

The optional configuration variable **gc.packRefs** determines if *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. This defaults to true.

The optional configuration variable **gc.commitGraph** determines if *git gc* should run *git commit−graph write*. This can be set to a boolean value. This defaults to false.

The optional configuration variable **gc.aggressiveWindow** controls how much time is spent optimizing the delta compression of the objects in the repository when the −−aggressive option is specified. The larger the value, the more time is spent optimizing the delta compression. See the documentation for the −−window option in **git-repack**(1) for more details. This defaults to 250.

Similarly, the optional configuration variable **gc.aggressiveDepth** controls −−depth option in **git-repack**(1). This defaults to 50.

The optional configuration variable **gc.pruneExpire** controls how old the unreferenced loose objects have to be before they are pruned. The default is "2 weeks ago".

Optional configuration variable **gc.worktreePruneExpire** controls how old a stale working tree should be before **git worktree prune** deletes it. Default is "3 months ago".

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, refs saved by *git filter−branch* in refs/original/, or reflogs (which may reference commits in branches that were later amended or rewound). 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) unless they turn off automatic garbage collection with *git config gc.auto 0*.

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