NAME

git-maintenance − Run tasks to optimize Git repository data

SYNOPSIS

git maintenance run [<options>]

DESCRIPTION

Run tasks to optimize Git repository data, speeding up other Git commands and reducing storage requirements for the repository.

Git commands that add repository data, such as **git add** or **git fetch**, are optimized for a responsive user experience. These commands do not take time to optimize the Git data, since such optimizations scale with the full size of the repository while these user commands each perform a relatively small action.

The **git maintenance** command provides flexibility for how to optimize the Git repository.

SUBCOMMANDS

register

Initialize Git config values so any scheduled maintenance will start running on this repository. This adds the repository to the **maintenance.repo** config variable in the current user’s global config and enables some recommended configuration values for **maintenance.<task>.schedule**. The tasks that are enabled are safe for running in the background without disrupting foreground processes.

The **register** subcommand will also set the **maintenance.strategy** config value to **incremental**, if this value is not previously set. The **incremental** strategy uses the following schedule for each maintenance task:

- **gc**: disabled.
- **commit−graph**: hourly.
- **prefetch**: hourly.
- **loose−objects**: daily.
- **incremental−repack**: daily.

git maintenance register will also disable foreground maintenance by setting **maintenance.auto = false** in the current repository. This config setting will remain after a **git maintenance unregister** command.

run

Run one or more maintenance tasks. If one or more **−−task** options are specified, then those tasks are run in that order. Otherwise, the tasks are determined by which **maintenance.<task>.enabled** config options are true. By default, only **maintenance.gc.enabled** is true.

start

Start running maintenance on the current repository. This performs the same config updates as the **register** subcommand, then updates the background scheduler to run **git maintenance run −−scheduled** on an hourly basis.

stop

Halt the background maintenance schedule. The current repository is not removed from the list of maintained repositories, in case the background maintenance is restarted later.

unregister

Remove the current repository from background maintenance. This only removes the repository from the configured list. It does not stop the background maintenance processes from running.

TASKS

commit−graph

The **commit−graph** job updates the **commit−graph** files incrementally, then verifies that the written data is correct. The incremental write is safe to run alongside concurrent Git processes since it will not expire **.graph** files that were in the previous **commit−graph−chain** file. They will be deleted by a later run based on the expiration delay.

prefetch

The **prefetch** task updates the object directory with the latest objects from all registered remotes. For each remote, a **git fetch** command is run. The configured refspec is modified to place all requested refs within **refs/prefetch/**. Also, tags are not updated.

This is done to avoid disrupting the remote−tracking branches. The end users expect these refs to stay unmoved unless they initiate a fetch. With prefetch task, however, the objects necessary to complete a later real fetch would already be obtained, so the real fetch would go faster. In the ideal case, it will just become an update to a bunch of remote−tracking branches without any object transfer.

gc

Clean up unnecessary files and optimize the local repository. "GC" stands for "garbage collection," but this task performs many smaller tasks. This task can be expensive for large repositories, as it repacks all Git objects into a single pack−file. It can also be disruptive in some situations, as it deletes stale data. See **git-gc**(1) for more details on garbage collection in Git.

loose−objects

The **loose−objects** job cleans up loose objects and places them into pack−files. In order to prevent race conditions with concurrent Git commands, it follows a two−step process. First, it deletes any loose objects that already exist in a pack−file; concurrent Git processes will examine the pack−file for the object data instead of the loose object. Second, it creates a new pack−file (starting with "loose−") containing a batch of loose objects. The batch size is limited to 50 thousand objects to prevent the job from taking too long on a repository with many loose objects. The **gc** task writes unreachable objects as loose objects to be cleaned up by a later step only if they are not re−added to a pack−file; for this reason it is not advisable to enable both the **loose−objects** and **gc** tasks at the same time.

incremental−repack

The **incremental−repack** job repacks the object directory using the **multi−pack−index** feature. In order to prevent race conditions with concurrent Git commands, it follows a two−step process. First, it calls **git multi−pack−index expire** to delete pack−files unreferenced by the **multi−pack−index** file. Second, it calls **git multi−pack−index repack** to select several small pack−files and repack them into a bigger one, and then update the **multi−pack−index** entries that refer to the small pack−files to refer to the new pack−file. This prepares those small pack−files for deletion upon the next run of **git multi−pack−index expire**. The selection of the small pack−files is such that the expected size of the big pack−file is at least the batch size; see the **−−batch−size** option for the **repack** subcommand in **git-multi-pack-index**(1). The default batch−size is zero, which is a special case that attempts to repack all pack−files into a single pack−file.

pack−refs

The **pack−refs** task collects the loose reference files and collects them into a single file. This speeds up operations that need to iterate across many references. See **git-pack-refs**(1) for more information.

OPTIONS

−−auto

When combined with the **run** subcommand, run maintenance tasks only if certain thresholds are met. For example, the **gc** task runs when the number of loose objects exceeds the number stored in the **gc.auto** config setting, or when the number of pack−files exceeds the **gc.autoPackLimit** config setting. Not compatible with the **−−schedule** option.

−−schedule

When combined with the **run** subcommand, run maintenance tasks only if certain time conditions are met, as specified by the **maintenance.<task>.schedule** config value for each **<task>**. This config value specifies a number of seconds since the last time that task ran, according to the **maintenance.<task>.lastRun** config value. The tasks that are tested are those provided by the **−−task=<task>** option(s) or those with **maintenance.<task>.enabled** set to true.

−−quiet
Do not report progress or other information over **stderr**.

−−task=<task>
If this option is specified one or more times, then only run the specified tasks in the specified order. If no **−−task=<task>** arguments are specified, then only the tasks with **maintenance.<task>.enabled** configured as **true** are considered. See the *TASKS* section for the list of accepted **<task>** values.

−−scheduler=auto|crontab|systemd−timer|launchctl|schtasks
When combined with the **start** subcommand, specify the scheduler for running the hourly, daily and weekly executions of **git maintenance run**. Possible values for **<scheduler>** are **auto**, **crontab** (POSIX), **systemd−timer** (Linux), **launchctl** (macOS), and **schtasks** (Windows). When **auto** is specified, the appropriate platform−specific scheduler is used; on Linux, **systemd−timer** is used if available, otherwise **crontab**. Default is **auto**.

TROUBLESHOOTING

The **git maintenance** command is designed to simplify the repository maintenance patterns while minimizing user wait time during Git commands. A variety of configuration options are available to allow customizing this process. The default maintenance options focus on operations that complete quickly, even on large repositories.

Users may find some cases where scheduled maintenance tasks do not run as frequently as intended. Each **git maintenance run** command takes a lock on the repository's object database, and this prevents other concurrent **git maintenance run** commands from running on the same repository. Without this safeguard, competing processes could leave the repository in an unpredictable state.

The background maintenance schedule runs **git maintenance run** processes on an hourly basis. Each run executes the "hourly" tasks. At midnight, that process also executes the "daily" tasks. At midnight on the first day of the week, that process also executes the "weekly" tasks. A single process iterates over each registered repository, performing the scheduled tasks for that frequency. Depending on the number of registered repositories and their sizes, this process may take longer than an hour. In this case, multiple **git maintenance run** commands may run on the same repository at the same time, colliding on the object database lock. This results in one of the two tasks not running.

If you find that some maintenance windows are taking longer than one hour to complete, then consider reducing the complexity of your maintenance tasks. For example, the **gc** task is much slower than the **incremental−repack** task. However, this comes at a cost of a slightly larger object database. Consider moving more expensive tasks to be run less frequently.

Expert users may consider scheduling their own maintenance tasks using a different schedule than is available through **git maintenance start** and Git configuration options. These users should be aware of the object database lock and how concurrent **git maintenance run** commands behave. Further, the **git gc** command should not be combined with **git maintenance run** commands. **git gc** modifies the object database but does not take the lock in the same way as **git maintenance run**. If possible, use **git maintenance run −−task=gc** instead of **git gc**.

The following sections describe the mechanisms put in place to run background maintenance by **git maintenance start** and how to customize them.

BACKGROUND MAINTENANCE ON POSIX SYSTEMS

The standard mechanism for scheduling background tasks on POSIX systems is cron(8). This tool executes commands based on a given schedule. The current list of user−scheduled tasks can be found by running **crontab −l**. The schedule written by **git maintenance start** is similar to this:

```
# BEGIN GIT MAINTENANCE SCHEDULE
# The following schedule was created by Git
# Any edits made in this region might be
# replaced in the future by a Git command.

0 1−23 * * * "/<path>/git" −−exec−path="/<path>" for−each−repo −−config=maintenance.repo maintenance run −−sched
0 0 * * 1−6 "/<path>/git" −−exec−path="/<path>" for−each−repo −−config=maintenance.repo maintenance run −−schedu
0 0 * * 0 "/<path>/git" −−exec−path="/<path>" for−each−repo −−config=maintenance.repo maintenance run −−schedule=

# END GIT MAINTENANCE SCHEDULE
```

The comments are used as a region to mark the schedule as written by Git. Any modifications within this region will be completely deleted by **git maintenance stop** or overwritten by **git maintenance start**.

The **crontab** entry specifies the full path of the **git** executable to ensure that the executed **git** command is the same one with which **git maintenance start** was issued independent of **PATH**. If the same user runs **git maintenance start** with multiple Git executables, then only the latest executable is used.

These commands use **git for−each−repo −−config=maintenance.repo** to run **git maintenance run −−schedule=<frequency>** on each repository listed in the multi−valued **maintenance.repo** config option. These are typically loaded from the user−specific global config. The **git maintenance** process then determines which maintenance tasks are configured to run on each repository with each **<frequency>** using the **maintenance.<task>.schedule** config options. These values are loaded from the global or repository config values.

If the config values are insufficient to achieve your desired background maintenance schedule, then you can create your own schedule. If you run **crontab −e**, then an editor will load with your user−specific **cron** schedule. In that editor, you can add your own schedule lines. You could start by adapting the default schedule listed earlier, or you could read the crontab(5) documentation for advanced scheduling techniques. Please do use the full path and **−−exec−path** techniques from the default schedule to ensure you are executing the correct binaries in your schedule.

BACKGROUND MAINTENANCE ON LINUX SYSTEMD SYSTEMS

While Linux supports **cron**, depending on the distribution, **cron** may be an optional package not necessarily installed. On modern Linux distributions, systemd timers are superseding it.

If user systemd timers are available, they will be used as a replacement of **cron**.

In this case, **git maintenance start** will create user systemd timer units and start the timers. The current list of user−scheduled tasks can be found by running **systemctl −−user list−timers**. The timers written by **git maintenance start** are similar to this:

```
$ systemctl −−user list−timers
NEXT                         LEFT          LAST                         PASSED       UNIT                         ACTIVATES
Thu 2021−04−29 19:00:00 CEST 42min left    Thu 2021−04−29 18:00:11 CEST 17min ago    git−maintenance@hourly.tir
Fri 2021−04−30 00:00:00 CEST 5h 42min left Thu 2021−04−29 00:00:11 CEST 18h ago      git−maintenance@daily.timer
Mon 2021−05−03 00:00:00 CEST 3 days left   Mon 2021−04−26 00:00:11 CEST 3 days ago   git−maintenance@weekly.ti
```

One timer is registered for each **−−schedule=<frequency>** option.

The definition of the systemd units can be inspected in the following files:

```
~/.config/systemd/user/git−maintenance@.timer
```

```
˜/.config/systemd/user/git−maintenance@.service
˜/.config/systemd/user/timers.target.wants/git−maintenance@hourly.timer
˜/.config/systemd/user/timers.target.wants/git−maintenance@daily.timer
˜/.config/systemd/user/timers.target.wants/git−maintenance@weekly.timer
```

git maintenance start will overwrite these files and start the timer again with **systemctl −−user**, so any customization should be done by creating a drop−in file, i.e. a **.conf** suffixed file in the **˜/.config/systemd/user/git−maintenance@.service.d** directory.

git maintenance stop will stop the user systemd timers and delete the above mentioned files.

For more details, see **systemd.timer(5)**.

BACKGROUND MAINTENANCE ON MACOS SYSTEMS

While macOS technically supports **cron**, using **crontab −e** requires elevated privileges and the executed process does not have a full user context. Without a full user context, Git and its credential helpers cannot access stored credentials, so some maintenance tasks are not functional.

Instead, **git maintenance start** interacts with the **launchctl** tool, which is the recommended way to schedule timed jobs in macOS. Scheduling maintenance through **git maintenance (start|stop)** requires some **launchctl** features available only in macOS 10.11 or later.

Your user−specific scheduled tasks are stored as XML−formatted **.plist** files in **˜/Library/LaunchAgents/**. You can see the currently−registered tasks using the following command:

```
$ ls ˜/Library/LaunchAgents/org.git−scm.git*
org.git−scm.git.daily.plist
org.git−scm.git.hourly.plist
org.git−scm.git.weekly.plist
```

One task is registered for each **−−schedule=<frequency>** option. To inspect how the XML format describes each schedule, open one of these **.plist** files in an editor and inspect the **<array>** element following the **<key>StartCalendarInterval</key>** element.

git maintenance start will overwrite these files and register the tasks again with **launchctl**, so any customizations should be done by creating your own **.plist** files with distinct names. Similarly, the **git maintenance stop** command will unregister the tasks with **launchctl** and delete the **.plist** files.

To create more advanced customizations to your background tasks, see launchctl.plist(5) for more information.

BACKGROUND MAINTENANCE ON WINDOWS SYSTEMS

Windows does not support **cron** and instead has its own system for scheduling background tasks. The **git maintenance start** command uses the **schtasks** command to submit tasks to this system. You can inspect all background tasks using the Task Scheduler application. The tasks added by Git have names of the form **Git Maintenance (<frequency>)**. The Task Scheduler GUI has ways to inspect these tasks, but you can also export the tasks to XML files and view the details there.

Note that since Git is a console application, these background tasks create a console window visible to the current user. This can be changed manually by selecting the "Run whether user is logged in or not" option in Task Scheduler. This change requires a password input, which is why **git maintenance start** does not select it by default.

If you want to customize the background tasks, please rename the tasks so future calls to **git maintenance**

(start|stop) do not overwrite your custom tasks.

GIT

Part of the **git**(1) suite