



Linux Ubuntu 26.04 Manual Pages on command 'git-commit-graph.1'

\$ man git-commit-graph.1

GIT-COMMIT-GRAPH(1) Git Manual GIT-COMMIT-GRAPH(1)

NAME

git-commit-graph - Write and verify Git commit-graph files

SYNOPSIS

git commit-graph verify [--object-dir <dir>] [--shallow] [--[no-]progress]

git commit-graph write [--object-dir <dir>] [--append]

 [--split[=<strategy>]] [--reachable | --stdin-packs | --stdin-commits]

 [--changed-paths] [--[no-]max-new-filters <n>] [--[no-]progress]

 <split-options>

DESCRIPTION

Manage the serialized commit-graph file.

OPTIONS

--object-dir

Use given directory for the location of packfiles and commit-graph file. This parameter exists to specify the location of an alternate that only has the objects directory, not a full .git directory. The commit-graph file is expected to be in the <dir>/info directory and the packfiles are expected to be in <dir>/pack. If the directory could not be made into an absolute path, or does not match any known object directory, git commit-graph ... will exit with non-zero status.

--progress, --no-progress

Turn progress on/off explicitly. If neither is specified, progress is shown if standard error is connected to a terminal.

COMMANDS

write

Write a commit-graph file based on the commits found in packfiles. If the config option `core.commitGraph` is disabled, then this command will output a warning, then return success without writing a commit-graph file.

With the `--stdin-packs` option, generate the new commit graph by walking objects only in the specified pack-indexes. (Cannot be combined with `--stdin-commits` or `--reachable`.)

With the `--stdin-commits` option, generate the new commit graph by walking commits starting at the commits specified in stdin as a list of OIDs in hex, one OID per line.

OIDs that resolve to non-commits (either directly, or by peeling tags) are silently ignored. OIDs that are malformed, or do not exist generate an error. (Cannot be combined with `--stdin-packs` or `--reachable`.)

With the `--reachable` option, generate the new commit graph by walking commits starting at all refs. (Cannot be combined with `--stdin-commits` or `--stdin-packs`.)

With the `--append` option, include all commits that are present in the existing commit-graph file.

With the `--changed-paths` option, compute and write information about the paths changed between a commit and its first parent. This operation can take a while on large repositories. It provides significant performance gains for getting history of a directory or a file with `git log -- <path>`. If this option is given, future commit-graph writes will automatically assume that this option was intended. Use `--no-changed-paths` to stop storing this data. `--changed-paths` is implied by config `commitGraph.changedPaths=true`.

With the `--max-new-filters=<n>` option, generate at most n new Bloom filters (if `--changed-paths` is specified). If n is -1, no limit is enforced. Only commits present in the new layer count against this limit. To retroactively compute Bloom filters over earlier layers, it is advised to use `--split=replace`. Overrides the `commitGraph.maxNewFilters` configuration.

With the `--split[=<strategy>]` option, write the commit-graph as a chain of multiple commit-graph files stored in `<dir>/info/commit-graphs`. Commit-graph layers are merged based on the strategy and other splitting options. The new commits not already in the commit-graph are added in a new "tip" file. This file is merged with the existing file if the following merge conditions are met:

? If `--split=no-merge` is specified, a merge is never performed, and the remaining

options are ignored. `--split=replace` overwrites the existing chain with a new one.

A bare `--split` defers to the remaining options. (Note that merging a chain of commit graphs replaces the existing chain with a length-1 chain where the first and only incremental holds the entire graph).

? If `--size-multiple=<X>` is not specified, let X equal 2. If the new tip file would have N commits and the previous tip has M commits and X times N is greater than M , instead merge the two files into a single file.

? If `--max-commits=<M>` is specified with M a positive integer, and the new tip file would have more than M commits, then instead merge the new tip with the previous tip.

Finally, if `--expire-time=<datetime>` is not specified, let `datetime` be the current time. After writing the split commit-graph, delete all unused commit-graph whose modified times are older than `datetime`.

verify

Read the commit-graph file and verify its contents against the object database. Used to check for corrupted data.

With the `--shallow` option, only check the tip commit-graph file in a chain of split commit-graphs.

EXAMPLES

? Write a commit-graph file for the packed commits in your local `.git` directory.

```
$ git commit-graph write
```

? Write a commit-graph file, extending the current commit-graph file using commits in `<pack-index>`.

```
$ echo <pack-index> | git commit-graph write --stdin-packs
```

? Write a commit-graph file containing all reachable commits.

```
$ git show-ref -s | git commit-graph write --stdin-commits
```

? Write a commit-graph file containing all commits in the current commit-graph file along with those reachable from `HEAD`.

```
$ git rev-parse HEAD | git commit-graph write --stdin-commits --append
```

CONFIGURATION

Everything below this line in this section is selectively included from the `git-config(1)` documentation. The content is the same as what's found there:

`commitGraph.generationVersion`

Specifies the type of generation number version to use when writing or reading the commit-graph file. If version 1 is specified, then the corrected commit dates will not be written or read. Defaults to 2.

`commitGraph.maxNewFilters`

Specifies the default value for the `--max-new-filters` option of `git commit-graph write` (c.f., `git-commit-graph(1)`).

`commitGraph.changedPaths`

If true, then `git commit-graph write` will compute and write changed-path Bloom filters by default, equivalent to passing `--changed-paths`. If false or unset, changed-paths Bloom filters will be written during `git commit-graph write` only if the filters already exist in the current commit-graph file. This matches the default behavior of `git commit-graph write` without any `--[no-]changed-paths` option. To rewrite a commit-graph file without any filters, use the `--no-changed-paths` option. Command-line option `--[no-]changed-paths` always takes precedence over this configuration. Defaults to unset.

`commitGraph.readChangedPaths`

Deprecated. Equivalent to `commitGraph.changedPathsVersion=-1` if true, and `commitGraph.changedPathsVersion=0` if false. (If `commitGraph.changedPathVersion` is also set, `commitGraph.changedPathsVersion` takes precedence.)

`commitGraph.changedPathsVersion`

Specifies the version of the changed-path Bloom filters that Git will read and write. May be -1, 0, 1, or 2. Note that values greater than 1 may be incompatible with older versions of Git which do not yet understand those versions. Use caution when operating in a mixed-version environment.

Defaults to -1.

If -1, Git will use the version of the changed-path Bloom filters in the repository, defaulting to 1 if there are none.

If 0, Git will not read any Bloom filters, and will write version 1 Bloom filters when instructed to write.

If 1, Git will only read version 1 Bloom filters, and will write version 1 Bloom filters.

If 2, Git will only read version 2 Bloom filters, and will write version 2 Bloom filters.

See `git-commit-graph(1)` for more information.

FILE FORMAT

see `gitformat-commit-graph(5)`.

GIT

Part of the `git(1)` suite

Git 2.53.0

03/02/2026

GIT-COMMIT-GRAPH(1)