

**NAME**

`git-fsck` – Verifies the connectivity and validity of the objects in the database

**SYNOPSIS**

```
git fsck [--tags] [--root] [--unreachable] [--cache] [--no-reflogs]
        [--[no-]full] [--strict] [--verbose] [--lost-found]
        [--[no-]dangling] [--[no-]progress] [--connectivity-only]
        [--[no-]name-objects] [<object>*]
```

**DESCRIPTION**

Verifies the connectivity and validity of the objects in the database.

**OPTIONS**

<object>

An object to treat as the head of an unreachability trace.

If no objects are given, *git fsck* defaults to using the index file, all SHA-1 references in **refs** namespace, and all reflogs (unless `--no-reflogs` is given) as heads.

`--unreachable`

Print out objects that exist but that aren't reachable from any of the reference nodes.

`--[no-]dangling`

Print objects that exist but that are never *directly* used (default). `--no-dangling` can be used to omit this information from the output.

`--root`

Report root nodes.

`--tags`

Report tags.

`--cache`

Consider any object recorded in the index also as a head node for an unreachability trace.

`--no-reflogs`

Do not consider commits that are referenced only by an entry in a reflog to be reachable. This option is meant only to search for commits that used to be in a ref, but now aren't, but are still in that corresponding reflog.

`--full`

Check not just objects in `GIT_OBJECT_DIRECTORY` (`$GIT_DIR/objects`), but also the ones found in alternate object pools listed in `GIT_ALTERNATE_OBJECT_DIRECTORIES` or `$GIT_DIR/objects/info/alternates`, and in packed Git archives found in `$GIT_DIR/objects/pack` and corresponding pack subdirectories in alternate object pools. This is now default; you can turn it off with `--no-full`.

`--connectivity-only`

Check only the connectivity of reachable objects, making sure that any objects referenced by a reachable tag, commit, or tree is present. This speeds up the operation by avoiding reading blobs entirely (though it does still check that referenced blobs exist). This will detect corruption in commits and trees, but not do any semantic checks (e.g., for format errors). Corruption in blob objects will not be detected at all.

Unreachable tags, commits, and trees will also be accessed to find the tips of dangling segments of history. Use `--no-dangling` if you don't care about this output and want to speed it up further.

`--strict`

Enable more strict checking, namely to catch a file mode recorded with `g+w` bit set, which was created by older versions of Git. Existing repositories, including the Linux kernel, Git itself, and sparse repository have old objects that triggers this check, but it is recommended to check new projects with

this flag.

`--verbose`

Be chatty.

`--lost-found`

Write dangling objects into `.git/lost-found/commit/` or `.git/lost-found/other/`, depending on type. If the object is a blob, the contents are written into the file, rather than its object name.

`--name-objects`

When displaying names of reachable objects, in addition to the SHA-1 also display a name that describes **how** they are reachable, compatible with `git-rev-parse(1)`, e.g.

**HEAD@{1234567890}~25^2:src/**

`--[no-]progress`

Progress status is reported on the standard error stream by default when it is attached to a terminal, unless `--no-progress` or `--verbose` is specified. `--progress` forces progress status even if the standard error stream is not directed to a terminal.

## CONFIGURATION

`fsck.<msg-id>`

During `fsck git` may find issues with legacy data which wouldn't be generated by current versions of `git`, and which wouldn't be sent over the wire if `transfer.fsckObjects` was set. This feature is intended to support working with legacy repositories containing such data.

Setting `fsck.<msg-id>` will be picked up by `git-fsck(1)`, but to accept pushes of such data set `receive.fsck.<msg-id>` instead, or to clone or fetch it set `fetch.fsck.<msg-id>`.

The rest of the documentation discusses `fsck.*` for brevity, but the same applies for the corresponding `receive.fsck.*` and `fetch.<msg-id>.*` variables.

Unlike variables like `color.ui` and `core.editor` the `receive.fsck.<msg-id>` and `fetch.fsck.<msg-id>` variables will not fall back on the `fsck.<msg-id>` configuration if they aren't set. To uniformly configure the same `fsck` settings in different circumstances all three of them they must all set to the same values.

When `fsck.<msg-id>` is set, errors can be switched to warnings and vice versa by configuring the `fsck.<msg-id>` setting where the `<msg-id>` is the `fsck` message ID and the value is one of **error**, **warn** or **ignore**. For convenience, `fsck` prefixes the error/warning with the message ID, e.g. "missingEmail: invalid author/commmitter line – missing email" means that setting `fsck.missingEmail = ignore` will hide that issue.

In general, it is better to enumerate existing objects with problems with `fsck.skipList`, instead of listing the kind of breakages these problematic objects share to be ignored, as doing the latter will allow new instances of the same breakages go unnoticed.

Setting an unknown `fsck.<msg-id>` value will cause `fsck` to die, but doing the same for `receive.fsck.<msg-id>` and `fetch.fsck.<msg-id>` will only cause `git` to warn.

`fsck.skipList`

The path to a list of object names (i.e. one unabbreviated SHA-1 per line) that are known to be broken in a non-fatal way and should be ignored. On versions of `Git 2.20` and later comments (`#`), empty lines, and any leading and trailing whitespace is ignored. Everything but a SHA-1 per line will error out on older versions.

This feature is useful when an established project should be accepted despite early commits containing errors that can be safely ignored such as invalid commmitter email addresses. Note: corrupt objects cannot be skipped with this setting.

Like `fsck.<msg-id>` this variable has corresponding `receive.fsck.skipList` and `fetch.fsck.skipList` variants.

Unlike variables like `color.ui` and `core.editor` the `receive.fsck.skipList` and `fetch.fsck.skipList` variables will not fall back on the `fsck.skipList` configuration if they aren't set. To uniformly configure the same fsck settings in different circumstances all three of them they must all set to the same values.

Older versions of Git (before 2.20) documented that the object names list should be sorted. This was never a requirement, the object names could appear in any order, but when reading the list we tracked whether the list was sorted for the purposes of an internal binary search implementation, which could save itself some work with an already sorted list. Unless you had a humongous list there was no reason to go out of your way to pre-sort the list. After Git version 2.20 a hash implementation is used instead, so there's now no reason to pre-sort the list.

## DISCUSSION

`git-fsck` tests SHA-1 and general object sanity, and it does full tracking of the resulting reachability and everything else. It prints out any corruption it finds (missing or bad objects), and if you use the `--unreachable` flag it will also print out objects that exist but that aren't reachable from any of the specified head nodes (or the default set, as mentioned above).

Any corrupt objects you will have to find in backups or other archives (i.e., you can just remove them and do an `rsync` with some other site in the hopes that somebody else has the object you have corrupted).

If `core.commitGraph` is true, the `commit-graph` file will also be inspected using `git commit-graph verify`. See `git-commit-graph(1)`.

## EXTRACTED DIAGNOSTICS

expect dangling commits – potential heads – due to lack of head information

You haven't specified any nodes as heads so it won't be possible to differentiate between un-parented commits and root nodes.

missing sha1 directory `<dir>`

The directory holding the sha1 objects is missing.

unreachable `<type>` `<object>`

The `<type>` object `<object>`, isn't actually referred to directly or indirectly in any of the trees or commits seen. This can mean that there's another root node that you're not specifying or that the tree is corrupt. If you haven't missed a root node then you might as well delete unreachable nodes since they can't be used.

missing `<type>` `<object>`

The `<type>` object `<object>`, is referred to but isn't present in the database.

dangling `<type>` `<object>`

The `<type>` object `<object>`, is present in the database but never *directly* used. A dangling commit could be a root node.

hash mismatch `<object>`

The database has an object whose hash doesn't match the object database value. This indicates a serious data integrity problem.

## ENVIRONMENT VARIABLES

`GIT_OBJECT_DIRECTORY`

used to specify the object database root (usually `$GIT_DIR/objects`)

`GIT_INDEX_FILE`

used to specify the index file of the index

`GIT_ALTERNATE_OBJECT_DIRECTORIES`

used to specify additional object database roots (usually unset)

**GIT**

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