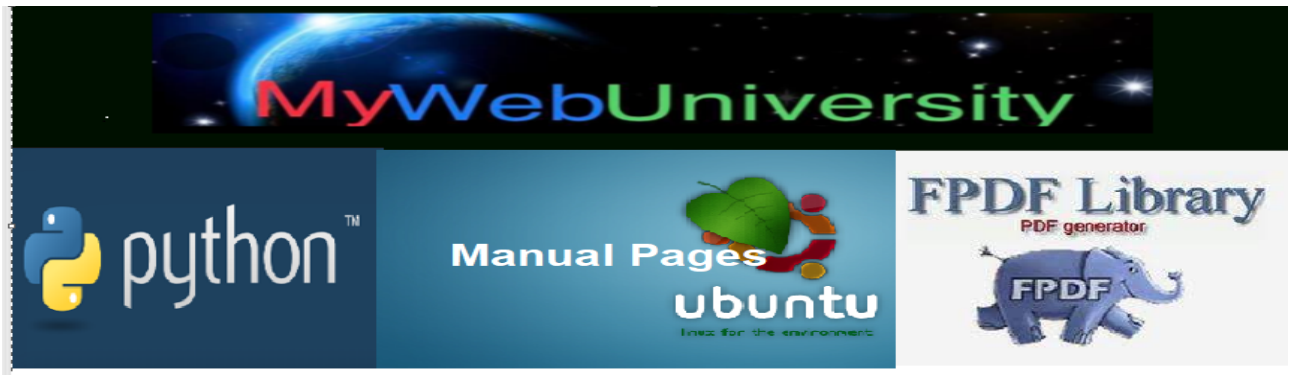

Full credit is given to the above companies including the OS that this PDF file was generated!

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

$ man git-fsck.1

GIT-FSCK(1) Git Manual GIT-FSCK(1)

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] [--[no-]references] [<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 the 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.

--dangling, --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 are 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 trigger 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/.

--progress, --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.

--references, --no-references

Control whether to check the references database consistency via git refs verify. See git-refs(1) for details. The default is to check the references database.

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:

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.fsck.*. 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 must be 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/committer 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.

See the Fsck Messages section of git-fsck(1) for supported values of <msg-id>.

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 are 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 committer 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 must be 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

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.

FSCK MESSAGES

The following lists the types of errors git fsck detects and what each error means, with their default severity. The severity of the error, other than those that are marked as "(FATAL)", can be tweaked by setting the corresponding fsck.<msg-id> configuration variable.

badDate

(ERROR) Invalid date format in an author/committer line.

badDateOverflow

(ERROR) Invalid date value in an author/committer line.

badEmail

(ERROR) Invalid email format in an author/committer line.

badFilemode

(INFO) A tree contains a bad filemode entry.

badGpgsig

(ERROR) A tag contains a bad (truncated) signature (e.g., gpgsig) header.

badHeadTarget

(ERROR) The HEAD ref is a symref that does not refer to a branch.

badHeaderContinuation

(ERROR) A continuation header (such as for gpgsig) is unexpectedly truncated.

badName

(ERROR) An author/committer name is empty.

badObjectSha1

(ERROR) An object has a bad sha1.

badPackedRefEntry

(ERROR) The "packed-refs" file contains an invalid entry.

badPackedRefHeader

(ERROR) The "packed-refs" file contains an invalid header.

badParentSha1

(ERROR) A commit object has a bad parent sha1.

badRefContent

(ERROR) A ref has bad content.

badRefFiletype

(ERROR) A ref has a bad file type.

badRefName

(ERROR) A ref has an invalid format.

badRefOid

(ERROR) A ref points to an invalid object ID.

badReferentName

(ERROR) The referent name of a symref is invalid.

badReftableTableName

(WARN) A reftable table has an invalid name.

badTagName

(INFO) A tag has an invalid format.

badTimezone

(ERROR) Found an invalid time zone in an author/committer line.

badTree

(ERROR) A tree cannot be parsed.

badTreeSha1

(ERROR) A tree has an invalid format.

badType

(ERROR) Found an invalid object type.

duplicateEntries

(ERROR) A tree contains duplicate file entries.

emptyName

(WARN) A path contains an empty name.

emptyPackedRefsFile

(INFO) "packed-refs" file is empty. Report to the git@vger.kernel.org[1] mailing list if you see this error. As only very early versions of Git would create such an empty "packed_refs" file, we might tighten this rule in the future.

extraHeaderEntry

(IGNORE) Extra headers found after tagger.

fullPathname

(WARN) A path contains the full path starting with "/".

gitattributesBlob

(ERROR) A non-blob found at .gitattributes.

gitattributesLarge

(ERROR) The .gitattributes blob is too large.

gitattributesLineLength

(ERROR) The .gitattributes blob contains too long lines.

gitattributesMissing

(ERROR) Unable to read .gitattributes blob.

gitattributesSymlink

(INFO) .gitattributes is a symlink.

gitignoreSymlink

(INFO) .gitignore is a symlink.

gitmodulesBlob

(ERROR) A non-blob found at .gitmodules.

gitmodulesLarge

(ERROR) The .gitmodules file is too large to parse.

gitmodulesMissing

(ERROR) Unable to read .gitmodules blob.

gitmodulesName

(ERROR) A submodule name is invalid.

gitmodulesParse

(INFO) Could not parse .gitmodules blob.

gitmodulesPath

(ERROR) .gitmodules path is invalid.

gitmodulesSymlink

(ERROR) .gitmodules is a symlink.

gitmodulesUpdate

(ERROR) Found an invalid submodule update setting.

gitmodulesUrl

(ERROR) Found an invalid submodule url.

hasDot

(WARN) A tree contains an entry named ..

hasDotdot

(WARN) A tree contains an entry named ...

hasDotgit

(WARN) A tree contains an entry named .git.

largePathname

(WARN) A tree contains an entry with a very long path name. If the value of fsck.largePathname contains a colon, that value is used as the maximum allowable length (e.g., "warn:10" would complain about any path component of 11 or more bytes). The default value is 4096.

mailmapSymlink

(INFO) .mailmap is a symlink.

missingAuthor

(ERROR) Author is missing.

missingCommitter

(ERROR) Committer is missing.

missingEmail

(ERROR) Email is missing in an author/committer line.

missingNameBeforeEmail

(ERROR) Missing name before an email in an author/committer line.

missingObject

(ERROR) Missing object line in tag object.

missingSpaceBeforeDate

(ERROR) Missing space before date in an author/committer line.

missingSpaceBeforeEmail

(ERROR) Missing space before the email in an author/committer line.

missingTag

(ERROR) Unexpected end after type line in a tag object.

missingTagEntry

(ERROR) Missing tag line in a tag object.

missingTaggerEntry

(INFO) Missing tagger line in a tag object.

missingTree

(ERROR) Missing tree line in a commit object.

missingType

(ERROR) Invalid type value on the type line in a tag object.

missingTypeEntry

(ERROR) Missing type line in a tag object.

multipleAuthors

(ERROR) Multiple author lines found in a commit.

nulInCommit

(WARN) Found a NUL byte in the commit object body.

nulInHeader

(FATAL) NUL byte exists in the object header.

nullSha1

(WARN) Tree contains entries pointing to a null sha1.

packedRefEntryNotTerminated

(ERROR) The "packed-refs" file contains an entry that is not terminated by a newline.

packedRefUnsorted

(ERROR) The "packed-refs" file is not sorted.

refMissingNewline

(INFO) A loose ref that does not end with newline(LF). As valid implementations of Git never created such a loose ref file, it may become an error in the future. Report to the

git@vger.kernel.org[1] mailing list if you see this error, as we need to know what tools created such a file.

symlinkRef

(INFO) A symbolic link is used as a symref. Report to the git@vger.kernel.org[1] mailing list if you see this error, as we are assessing the feasibility of dropping the support to drop creating symbolic links as symrefs.

symrefTargetIsNotARef

(INFO) The target of a symbolic reference points neither to a root reference nor to a reference starting with "refs/". Although we allow create a symref pointing to the referent which is outside the "ref" by using git symbolic-ref, we may tighten the rule in the future. Report to the git@vger.kernel.org[1] mailing list if you see this error, as we need to know what tools created such a file.

trailingRefContent

(INFO) A loose ref has trailing content. As valid implementations of Git never created such a loose ref file, it may become an error in the future. Report to the git@vger.kernel.org[1] mailing list if you see this error, as we need to know what tools created such a file.

treeNotSorted

(ERROR) A tree is not properly sorted.

unknownType

(ERROR) Found an unknown object type.

unterminatedHeader

(FATAL) Missing end-of-line in the object header.

zeroPaddedDate

(ERROR) Found a zero padded date in an author/committer line.

zeroPaddedFilemode

(WARN) Found a zero padded filemode in a tree.

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

NOTES

1. git@vger.kernel.org

   mailto:git@vger.kernel.org