Rocky Enterprise Linux 9.2 Manual Pages on command 'git-fast-export.1'

C:\>man git-fast-export.1

GIT-FAST-EXPORT(1) Git Manual GIT-FAST-EXPORT(1)

NAME

git-fast-export - Git data exporter

SYNOPSIS

git fast-export [<options>] | git fast-import

DESCRIPTION

This program dumps the given revisions in a form suitable to be piped into git fast-import.

You can use it as a human-readable bundle replacement (see git-bundle(1)), or as a format that can be edited before being fed to git fast-import in order to do history rewrites (an ability relied on by tools like git filter-repo).

OPTIONS

--progress=<n>

Insert progress statements every <n> objects, to be shown by git fast-import during import.

--signed-tags=(verbatim|warn|warn-strip|strip|abort)

Specify how to handle signed tags. Since any transformation after the export can change the tag names (which can also happen when excluding revisions) the signatures will not match.

When asking to abort (which is the default), this program will die when encountering a signed tag. With strip, the tags will silently be made unsigned, with warn-strip they will be made unsigned but a warning will be displayed,

with verbatim, they will be silently exported and with warn, they will be exported, but you will see a warning.

--tag-of-filtered-object=(abort|drop|rewrite)

Specify how to handle tags whose tagged object is filtered out. Since revisions and files to export can be limited by path, tagged objects may be filtered completely.

When asking to abort (which is the default), this program will die when encountering such a tag. With drop it will omit such tags from the output. With rewrite, if the tagged object is a commit, it will rewrite the tag to tag an ancestor commit (via parent rewriting; see git-rev-list(1))

-M, -C

Perform move and/or copy detection, as described in the git-diff(1) manual page, and use it to generate rename and copy commands in the output dump. Note that earlier versions of this command did not complain and produced incorrect results if you gave these options.

--export-marks=<file>

Dumps the internal marks table to <file> when complete. Marks are written one per line as :markid SHA-1. Only marks for revisions are dumped; marks for blobs are ignored. Backends can use this file to validate imports after they have been completed, or to save the marks table across incremental runs. As <file> is only opened and truncated at completion, the same path can also be safely given to --import-marks. The file will not be written if no new object has been marked/exported.

--import-marks=<file>

Before processing any input, load the marks specified in <file>. The input file must exist, must be readable, and must use the same format as produced by --export-marks.

--mark-tags

In addition to labelling blobs and commits with mark ids, also label tags. This is useful in conjunction with --export-marks and --import-marks, and is also useful (and necessary) for exporting of nested tags. It does not hurt other cases and would be the default, but many fast-import frontends are not prepared to accept tags with mark identifiers.

Any commits (or tags) that have already been marked will not be exported again. If the backend uses a similar --import-marks file, this allows for incremental bidirectional exporting of the repository by keeping the marks the same across runs.

--fake-missing-tagger

Some old repositories have tags without a tagger. The fast-import protocol was pretty strict about that, and did not allow that. So fake a tagger to be able to fast-import the output.

--use-done-feature

Start the stream with a feature done stanza, and terminate it with a done command.

--no-data

Skip output of blob objects and instead refer to blobs via their original SHA-1 hash. This is useful when rewriting the directory structure or history of a repository without touching the contents of individual files. Note that the resulting stream can only be used by a repository which already contains the necessary objects.

--full-tree

This option will cause fast-export to issue a "deleteall" directive for each commit followed by a full list of all files in the commit (as opposed to just listing the files which are different from the commit?s first parent).

--anonymize

Anonymize the contents of the repository while still retaining the shape of the history and stored tree. See the section on ANONYMIZING below.

--reference-excluded-parents

By default, running a command such as git fast-export master~5..master will not include the commit master~5 and will make master~4 no longer have master~5 as a parent (though both the old master~4 and new master~4 will have all the same files). Use --reference-excluded-parents to instead have the stream refer to commits in the excluded range of history by their sha1sum. Note that the resulting stream can only be used by a repository which already contains the necessary parent commits.

--show-original-ids

Add an extra directive to the output for commits and blobs, original-oid <SHA1SUM>. While such directives will likely be ignored by importers such as git-fast-import, it may be useful for intermediary filters (e.g. for rewriting commit messages which refer to older commits, or for stripping blobs by id).

--reencode=(yes|no|abort)

Specify how to handle encoding header in commit objects. When asking to abort (which is the default), this program will die when encountering such a commit object. With yes, the commit message will be re-encoded into UTF-8. With no, the original encoding will be preserved.

--refspec

Apply the specified refspec to each ref exported. Multiple of them can be specified.

[<git-rev-list-args>...]

A list of arguments, acceptable to git rev-parse and git rev-list, that specifies the specific objects and references to export. For example, master~10..master causes the current master reference to be exported along with all objects added since its 10th ancestor commit and (unless the --reference-excluded-parents option is specified) all files common to master~9 and master~10.

EXAMPLES

$ git fast-export --all | (cd /empty/repository && git fast-import)

This will export the whole repository and import it into the existing empty repository. Except for reencoding commits that are not in UTF-8, it would be a one-to-one mirror.

$ git fast-export master~5..master |
sed "s|refs/heads/master|refs/heads/other|" |
git fast-import

This makes a new branch called other from master~5..master (i.e. if master has linear history, it will take the last 5 commits).

Note that this assumes that none of the blobs and commit messages referenced by that revision range contains the string refs/heads/master.

ANONYMIZING

If the --anonymize option is given, git will attempt to remove all identifying

information from the repository while still retaining enough of the original tree and history patterns to reproduce some bugs. The goal is that a git bug which is found on a private repository will persist in the anonymized repository, and the latter can be shared with git developers to help solve the bug.

With this option, git will replace all refnames, paths, blob contents, commit and tag messages, names, and email addresses in the output with anonymized data. Two instances of the same string will be replaced equivalently (e.g., two commits with the same author will have the same anonymized author in the output, but bear no resemblance to the original author string). The relationship between commits, branches, and tags is retained, as well as the commit timestamps (but the commit messages and refnames bear no resemblance to the originals). The relative makeup of the tree is retained (e.g., if you have a root tree with 10 files and 3 trees, so will the output), but their names and the contents of the files will be replaced.

If you think you have found a git bug, you can start by exporting an anonymized stream of the whole repository:

```
$ git fast-export --anonymize --all >anon-stream
```

Then confirm that the bug persists in a repository created from that stream (many bugs will not, as they really do depend on the exact repository contents):

```
$ git init anon-repo
$ cd anon-repo
$ git fast-import <../anon-stream
$ ... test your bug ...
```

If the anonymized repository shows the bug, it may be worth sharing anon-stream along with a regular bug report. Note that the anonymized stream compresses very well, so gzipping it is encouraged. If you want to examine the stream to see that it does not contain any private data, you can peruse it directly before sending.

You may also want to try:

```
$ perl -pe 's/\d+/X/g' <anon-stream | sort -u | less
```

which shows all of the unique lines (with numbers converted to "X", to collapse "User 0", "User 1", etc into "User X"). This produces a much smaller output, and it is usually easy to quickly confirm that there is no private data in the stream.

LIMITATIONS

Since git fast-import cannot tag trees, you will not be able to export the

linux.git repository completely, as it contains a tag referencing a tree instead of a commit.

SEE ALSO

git-fast-import(1)

GIT

Part of the git(1) suite