Git

Christian Külker

2022-05-30

Contents

1 History

Version	Date	Notes
0.1.4	2022-05-30	Change shell to bash for code fences
0.1.3	2022-05-09	Change section levels
0.1.2	2022-04-22	Front matter, history, syntax highlighting
0.1.1	2022-04-21	Data stream error, unknown compression method, unable to unpack HASH header
0.1.0	2022-04-13	Initial release (Reconcile Divergent Branches)

2 Inflate: data stream error and unable to unpack HASH header

WARNING: This section include hacks. Do this on your own risk and make plenty backups.

In case a git repository gets corrupted (due a corrupted file system for example) git shows an error message that seems to point to something else (at least in the first 2 lines).

```
git clone USER@HOST:REPOSITORY.git
Cloning into 'REPOSITORY'...
remote: error: inflate: data stream error (unknown compression method)
remote: error: unable to unpack HASH header
remote: fatal: loose object 81HASH1 (stored in ./objects/81/HASH1) is
 ↪  corrupt
error: git upload-pack: git-pack-objects died with error.
fatal: git upload-pack: aborting due to possible repository corruption on
 ↪  the remote side.
remote: aborting due to possible repository corruption on the remote side.
```

Here `REPOSITORY` and `HASH1` needs to be replaced with discrete values and `81` is probably a different value, `81HASH1` is the complete hash.

Make backups of the local and remote repository.

Usually an `git fsck` local repository should be fine. If there is access to the remote repository execute an `git fsck` with the correct user rights. (Replace `HASH1` and `HASH2` with hashes to imagine the original message).

```
git fsck
error: inflate: data stream error (unknown compression method)
error: unable to unpack header of ./objects/81/HASH1
error: 81HASH1: object corrupt or missing: ./objects/81/HASH1
Checking object directories: 100% (256/256), done.
missing tree 81HASH1
dangling tree HASH2
```

Some parameters to `fsck` will get more information.

```
git --bare fsck-objects --full
error: inflate: data stream error (unknown compression method)
error: unable to unpack header of
 ↪  PATH1/repositories/REPOSITORY.git/objects/81/HASH1
error: 81HASH1: object corrupt or missing:
 ↪  PATH1/repositories/REPOSITORY.git/objects/81/HASH1
```

```
Checking object directories: 100% (256/256), done.
missing tree 81HASH1
dangling tree HASH2
```

```
git fsck --full 81HASH1
Checking object directories: 100% (256/256), done.
Checking objects: 100% (285/285), done.
dangling commit HASH
```

To actually understand what the `81HASH1` is referencing (replace `81HASH1` with the current hash):

```
git ls-tree 81HASH1
040000 tree HASH5    DIRECTORY
120000 blob HASH6    FILE/LINK/OBJECT
```

This tells you that a DIRECTORY and a FILE/LINK/OBJECT is referenced by the `81HASH1`. One as tree the other as blob.

If you use `git log` you can find more information.

```
  # Lets find the PATH
find -name OBJECT
PATH/OBJECT
git log --raw --all --full-history -- PATH/OBJECT
commit HASH6
Author: FIRSTNAME LASTNAME (SOMETHING) <USER@HOST>
Date:   Tue Aug 24 03:35:45 2021 +0200

    add new link

:000000 120000 0000000... HASH7... A  PATH/OBJECT
```

If you have a **known** working (non corrupt) copy (clone) of the remote repository that do not show errors with `git fsck` and you have commit/push rights, you might try a `git push -f`.

(Disclaimer: I do not know enough about the internals of `git` to understand if `git push -f` is even a relevant candidate for “repairing” this situation. However I will try anyways.)

```
git push -f
Counting objects: 845, done.
```

```
Delta compression using up to 8 threads.
Compressing objects: 100% (610/610), done.
Writing objects: 100% (845/845), 207.98 MiB | 253.00 KiB/s, done.
Total 845 (delta 361), reused 268 (delta 85)
remote: error: inflate: data stream error (unknown compression method)
remote: error: unable to unpack 81HASH1 header
remote: fatal: cannot read existing object info 81HASH1
error: unpack failed: index-pack abnormal exit
To HOST:REPOSITORY.git
 ! [remote rejected] master -> master (unpacker error)
error: failed to push some refs to 'USER@HOST:REPOSITORY.git'
```

While it might or might not work, in this example it does not. As the problem lies on the remote site, my impression is, that is that much one can do locally.

As I have access to the server I was curious and run `md5sum` over the file `81/HASH1`.

```
locate HASH1|xargs md5sum
HASH2  /PATH1/repositories/REPOSITORY.git/objects/81/HASH1
HASH2  /PATH2/repositories-backup/REPOSITORY.git/objects/81/HASH1
HASH3  /PATH3/REPOSITORY/.git/objects/81/HASH1
HASH2  /PATH4/repositories/REPOSITORY.git/objects/81/HASH1
HASH2  /PATH5/repositories-backup/REPOSITORY.git/objects/81/HASH1
```

As visible, one `HASH3` was different than the other `HASH3`. While `PATH1` and `PATH2` are the `gitolite` repository on the server after migration, `PATH4` and `PATH5` are the pre migration states of the `gitolite` repository before it moved to the server. All `md5sum` hashes are the same. This means the error occurred prior to migration. Only a local copy of the repository on the server had a different hash, `HASH3`. That was a clone of the repository while it was not corrupted.

2.1 Backup Method

If you have a non corrupted version of the repository as a backup, now it is time to restore the repository from backup.

2.2 Copy Method

In case you have no backup, but some other copies of the repository (bare or working tree). This example copies a good version of `HASH1` from a local clone on the server to the central `gitolite` repository.

```
cp /PATH3/REPOSITORY/.git/objects/81/HASH1
 ↪  /PATH1/repositories/REPOSITORY.git/objects/81/HASH1
```

This “solved” the issue. It is debatable if it really solved the issue. One should carefully examine the repository with the non-corrupted clone before concluding it. However cloning was possible again.

```
git clone USER@HOST:REPOSITORY.git
Cloning into 'REPOSITORY'...
remote: Enumerating objects: 884, done.
remote: Counting objects: 100% (884/884), done.
remote: Compressing objects: 100% (733/733), done.
remote: Total 884 (delta 375), reused 0 (delta 0)1.98 MiB/s
Receiving objects: 100% (884/884), 209.04 MiB | 7.03 MiB/s, done.
Resolving deltas: 100% (375/375), done.
Checking out files: 100% (126/126), done.
```

2.3 Push -f Method

Out of curiosity I made some other tests on the remote machine. First I tested the local clone on the remote machine (after reverting the repository to the bad state):

```
cd PATH3/REPOSITORY
md5sum .git/objects/81/HASH1
HASH3 .git/objects/81/HASH1

git fsck
Checking object directories: 100% (256/256), done.
Checking objects: 100% (612/612), done

git push -f
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 304 bytes | 304.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
To HOST:REPOSITORY.git
   7497095..b6fbedd  master -> master

md5sum .git/objects/81/HASH1
 ↪  /PATH1/repositories/REPOSITORY.git/objects/81/HASH1
```

```
HASH3 .git/objects/81/HASH1
HASH2  /PATH1/repositories/REPOSITORY.git/objects/81/HASH1
```

So this operation do indeed not change `HASH1`. However the `push -f` worked. Doing a commit and a `push -f` from the local clone to the remote server did **not** work however.

```
git push -f
Counting objects: 845, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (610/610), done.
Writing objects: 100% (845/845), 207.98 MiB | 252.00 KiB/s, done.
Total 845 (delta 361), reused 268 (delta 85)
remote: error: inflate: data stream error (unknown compression method)
remote: error: unable to unpack 81HASH1 header
remote: fatal: cannot read existing object info 81HASH1
error: unpack failed: index-pack abnormal exit
To HOST:REPOSITORY.git
 ! [remote rejected] master -> master (unpacker error)
error: failed to push some refs to 'USER@HOST:REPOSITORY.git'
```

So this probably mean that `push -f` is not reliable to tell if the repository is broken or not, and nor does it repair it if execution is succeeding.

2.4 Repair a Blob with `hash-object` Method

This idea comes from git.kernel.org and works well on blobs. Trees are difficult to restore. The information retrieval method with `git log --raw --all --full-history -- PATH/OBJECT` might give information about a blob that can be restored from previous versions.

```
git log --raw --all --full-history -- PATH/OBJECT
commit HASH6
Author: FIRSTNAME LASTNAME (SOMETHING) <USER@HOST>
Date:   Tue Aug 24 03:35:45 2021 +0200

    add new link

:000000 120000 0000000... HASH7... A  PATH/OBJECT
```

Recreate the `OBJECT` either by checkout old version and use editor or by other means.

```
git hash-object -w PATH/NEW_OBJECT
```

However it can be that the object is not OK:

```
git log --raw --all --full-history -- PATH/LINK
error: inflate: data stream error (unknown compression method)
error: unable to unpack 81HASH1 header
fatal: loose object 81HASH1 (stored in ./objects/81/HASH1) is corrupt
```

2.5 Further Not Tested Possible Methods

Other possible solutions to repair a remote repository might be to use the following (but were not performed):

- `git repack`
- `git gc --aggressive`
- In case of big objects on servers with little RAM, one might investigate to change the limits on the git pack and unpack task, that might corrupt or die on the fly. One can see something like `error: pack-objects died of signal  91171/66888)` that something ended prematurely.

2.6 Summary:

To solve the issue with a corrupt repository, either use a good backup (preferably) or copy a correct object over the corrupt object (if you are bold). To decide if something is a good backup `md5sum` can help to see if git objects differ (have a bit error). The command `git push -f` seem irrelevant, the `hash-object` and `git log` approach will gather more information and will give you a better understanding what you are actually doing, however it will probably only be feasible on broken blobs and not on trees and it might be not reliable, due to the fact that the restoring method relies on the knowledge of the content of the (to be restored) object itself or the luck to find a minimal changed prior commit that is not corrupted.

3 Reconcile Divergent Branches

From git 2.27.0 onwards the user is confronted with a similar message from git, when using `git pull`.

```
1  Warning: Pulling without specifying how to reconcile divergent branches is
2  discouraged. You can squelch this message by running one of the following
3  commands sometime before your next pull:
4
5    git config pull.rebase false  # merge (the default strategy)
6    git config pull.rebase true   # rebase
7    git config pull.ff only       # fast-forward only
```

Newer version do basically the same:

```
hint: Pulling without specifying how to reconcile divergent branches is
hint: discouraged. You can squelch this message by running one of the
    following
hint: commands sometime before your next pull:
hint:
hint:   git config pull.rebase false  # merge (the default strategy)
hint:   git config pull.rebase true   # rebase
hint:   git config pull.ff only       # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a
    default
hint: preference for all repositories. You can also pass --rebase, --no-
    rebase,
hint: or --ff-only on the command line to override the configured default
    per
hint: invocation.
```

Which solution should be chosen?

I am not an expert on this, but here are my thoughts. This question is about which strategy to be used in case of a pull that would create a minor local “disturbance”: 1) merge 2) rebase or 3) fast-forward only.

It seems that the previous default (merge) strategy was not sane and the git developers decided to ask the user to chose a sane (or at least known) strategy (to the user). I think it is a good practice to ask and not to change the default behavior without asking and therefore risk to break work flow or code of users. The previous default strategy (merge) had the risk of the so called foxtrot merges where the order of the first HEAD and the second entry gets messed up. So `git config pull.rebase false` seems a risky option.

The `git pull --help` page looks innocent at the beginning. However the default is that this is a short form of `git fetch&&git merge FETCH_HEAD`. That can result in a merge commit (with or without foxtrot does not matter so much). This means that pulling from a remote repository is not a harmless operation as this is not a pure download and it might change the commit history by adding (not committed) stuff from your hard disk. And it might even include your own work into git, even though you did not anticipated it or you deliberately wanted to commit it later or not at all. And you might not even notice it. Let’s be honest. Who is reading all the gibberish git is writing all the time?

So if the first (default) option is evil, one should take the second one and make a `git fetch&&git rebase` for every `git pull`? Well, this circumvent foxtrot merges as the commit history is linear (clean). The local master would be on top of the remote origin/master. But still it changes the commit history (sometimes) without asking. Other (old) side effects, like the git history is a bunch of lies not considered.

In my opinion the `git pull --ff-only` is the best solution. This will complain if the operation would need a merge, or a re-base or whatever and not a clean fast-forward download. So executing `git config pull.ff only` will get rid of the warning. If you are even more convinced of this solution you can make it global: `git config --global pull.ff only`

While researching, almost at the end of writing, I stumbled over the sffc's Tech Blog with nice graphics that has a simliar view on the topic but explain it better.

To summarize (and a suggestion to improve up on the message [attention this is humor!]):

```
- `git config pull.rebase false` # occasionally making sneaky commits of own
                                   work with a dirty history that sometimes
                                   dance foxtrot (and may break the
    repository)
- `git config pull.rebase true`  # occasionally making sneaky commits of own
                                   work with a clean history (of lies)
- `git config pull.ff only`      # stop downloading if local work would be
                                   overwritten or committed and do not
    change
                                   (or lie about) history
```

4 Disclaimer of Warranty

5 Limitation of Liability