Git Annex vs. Git LFS

December 8, 2015 7 min read

Many of us have felt the shameful sting of committing a large file to an otherwise pristine repository. "But *it's* important *too*," you pleaded. "Idiot," they snarked, begrudgingly fixing your mistake while secretly acknowledging that you had a point. Years later, we have a resolution.

I've been working with Git Annex thanks to GitLab's adoption early this year. GitHub opted to support Git LFS, which I've only started poking at now that GitLab decided to support it, too.

But which one to use? Each will help you solve that large-file problem in a slightly different way. Think of Git Annex as an experienced librarian waiting at the information desk. You'll need to walk up to ask your question or return your book, but the librarian can help you in a variety of ways by getting books moved around, checking up on things, and generally being a pro at cataloging stuff. Git LFS is more like your large file personal assistant, working alongside you to keep track of things you've pointed out.

In other words, my experience with Annex is that it's full-featured and a bit less focused in its approach. It's easy enough to check in files and sync them among various locations, but there are also testing tools, a web-based GUI, and lots of options you can use in different situations. The git-annex project site reveals a lot: plenty of features, updates, discussions, and enough threads that some sort of trail off.

Git LFS is at the other end of things: a bit nicer-looking, a bit more straightforward, and significantly simpler. Tack it on to your repository, tell it what kind of files to watch, and then pretty much forget about it. If you check in a file (with a normal `git add whatever.mp4`), the magic happens via a pre-push hook where LFS will check your watch list and spring into action if needed. It otherwise blends in after minimal configuration.

Let's take an identical set of files and commit them using each.

Our Fake Project

A markdown file and four PNGs that we're going to treat as large files:

- test.md
- assets/01-north-pole-daylight.png
- assets/02-plus-blue-light.png
- assets/03-night-glow.png
- assets/04-day-glow.png

Git Annex

Once git-annex is installed, we need to configure our repository to use it:

```
$ git annex init
init ok
(recording state in git...)
```

Add our one normal file as usual:

```
$ git add test.md
```

Add our images to the annex:

```
$ git annex add assets/*
add assets/01-north-pole-daylight.png ok
add assets/02-plus-blue-light.png ok
add assets/03-night-glow.png ok
add assets/04-day-glow.png ok
(recording state in git...)
```

The files are turned into symbolic links that point to data in .git/annex/objects. (You can easily undo this if you ever need to, restoring the files and getting rid of the annex.) Push it all:

```
$ git push origin
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 282 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
```

Critically important here is that the PNG files are not yet at the origin, only symbolic links that point to them. Annex knows they're on the local machine, but that's the only place they exist so far. We can ask git-annex where a certain file exists:

```
git annex whereis 04-day-glow.png (1 copy)
 758aaecf-f502-44be-8854-03819db671d6 -- matt@mattBook-Air.local:~/D
```

This can be useful while exploring and figuring out what's going on. Now let's actually sync the file data:

```
$ git annex sync --content
commit ok
pull origin
ok
```

Each file will be copied with a progress report that looks like this:

```
copy assets/01-north-pole-daylight.png copy assets/01-north-pole-day
SHA256E- s1093459--12cbb85304bc084cedd0537830a09d55e3cd2224917e9faae
 1093459 100% 126.44MB/s 0:00:00 (xfer#1, to-check=0/1)

sent 1093758 bytes received 42 bytes 128682.35 bytes/sec
total size is 1093459 speedup is 1.00
ok
```

Before (as seen above) and after the binary transfer, git-annex will make sure the file metadata is properly synced as well.

```
pull origin
ok
(recording state in git...)
push origin
Counting objects: 14, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (14/14), 1.08 KiB | 0 bytes/s, done.
Total 14 (delta 4), reused 0 (delta 0)
To git@gitlab.com:workingconcept/annex-test.git
 4ef0edb..a8f52cd git-annex -> synced/git-annex
ok
```

We can use the same trick from above to confirm that a given PNG now lives in more than one place:

```
git annex whereis 04-day-glow.png (2 copies)
1d3c732a-ff9a-490e-a0c8-18ee13694c7a -- "GitLab" [origin]
758aaecf-f502-44be-8854-03819db671d6 -- matt@mattBook-Air.local:~/Do
```

Success! It's easy enough to clone the repository and use `git annex sync --content` to get everything synced up. A long list of files will go slowly since they're transferred one at a time, but my experience has been pretty solid[1].

Git LFS

Now let's do the same thing with LFS. Same first step, which is to invite git-lfs into the repository:

```
$ git lfs install
Updated pre-push hook.
Git LFS initialized.
```

Simple, yet hugely different: tell LFS what kind of files we'll treat separately:

```
$ git lfs track "assets/*"
Tracking assets/*
```

In this case, everything in the assets folder. It could just as well be extensions like `*.psd` or `*.mp4`, or a handful of paths. Now let's add everything and commit it:

```
$ git add .
$ git commit -m "Test commit."
[master (root-commit) 02a1537] Test commit.
 6 files changed, 16 insertions(+)
 create mode 100644 .gitattributes
 create mode 100644 assets/01-north-pole-daylight.png
 create mode 100644 assets/02-plus-blue-light.png
 create mode 100644 assets/03-night-glow.png
 create mode 100644 assets/04-day-glow.png
 create mode 100644 test.md
```

That's it, no extra step. Push:

```
$ git push origin -u
Git LFS: (4 of 4 files) 3.95 MB / 3.95 MB
Counting objects: 9, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 1.02 KiB | 0 bytes/s, done.
Total 9 (delta 0), reused 0 (delta 0)
To git@gitlab.com:workingconcept/lfs-test.git
 * [new branch] master - master
Branch master set up to track remote branch master from origin.
```

Note that LFS reported back about what it had to transfer, and that after the initialization we just go about our business as we normally would. LFS added `.gitattributes` to store a reference to its watched file patterns, and it jumps in and does its magic only when it needs to. Complete contents of `.gitattributes`:

```
assets/* filter=lfs diff=lfs merge=lfs -text
```

The second thing that LFS quietly adds to your repository is a pre-push hook. Thanks to these two things, you don't have to interact with lfs directly like you would with git-annex. Contents of the pre-push hook:

```
#!/bin/sh
command -v git-lfs >/dev/null 2>&1 || { echo >&2 "\nThis repository
git lfs pre-push "$@"
```

Feel Good Ending

So you have options. I don't know that either one's better, it's just a matter of choosing what works for you and is supported by your Git host.

I've not gotten myself into any complex situations yet, and I'm working on converting some repositories largely by running `uninit` to un-annex files and remove every trace of git-annex from the repository so I can turn around and check those files in with LFS.

Let me know if you've stumbled upon this post with more insights, corrections, or flagrant objections!

1. I've been stumped a few times when cloned repositories just don't sync binary data, but it's been a matter of setting `annex-ignore = false` in .git/config. ↩

by **Matt Stein**
Full stack tinkerer, sporadic blogger and Craft CMS fan occasionally found on the devMode.fm podcast.

Tagged
git git-annex git-lfs

Updated 7/16/19 at 5:24pm

Show Comments

Next Post:
On Visiting Peers

Previous Post:
Git Annex and Tower

plugins *blog* *about* *privacy*