The Future Of Large Files In Git Is Git

If Git had a nemesis, it'd be large files.

Large files bloat Git's storage, slow down `git clone`, and wreak havoc on Git forges.

In 2015, GitHub released Git LFS—a Git extension that hacked around problems with large files. But Git LFS added new complications and storage costs.

Meanwhile, the Git project has been quietly working on large files. And while LFS ain't dead yet, the latest Git release shows the path towards a future where LFS is, finally, obsolete.

What you can do today: replace Git LFS with Git partial clone

Git LFS works by storing large files outside your repo.

When you clone a project via LFS, you get the repo's history and small files, but skip large files. Instead, Git LFS downloads only the large files you need for your working copy.

In 2017, the Git project introduced **partial clones** that provide the same benefits as Git LFS:

> *Partial clone allows us to avoid downloading [large binary assets] in advance during clone and fetch operations and thereby reduce download times and disk usage.*
>
> *– Partial Clone Design Notes, git-scm.com*

Git's partial clone and LFS both make for:

1. **Small checkouts** – On clone, you get the latest copy of big files instead of **every** copy.
2. **Fast clones** – Because you avoid downloading large files, each clone is fast.

3. **Quick setup** – Unlike shallow clones, you get the entire history of the project—you can get to work right away.

What is a partial clone?

A Git partial clone is a clone with a `--filter`.

For example, to avoid downloading files bigger than 100KB, you'd use:

```
git clone --filter='blobs:size=100k' <repo>
```

Later, Git will lazily download any files over 100KB you need for your checkout.

By default, if I `git clone` a repo with many revisions of a noisome 25 MB PNG file, then cloning is slow and the checkout is obnoxiously large:

```
$ time git clone https://github.com/thcipriani/noise-over-git
Cloning into '/tmp/noise-over-git'...
...
Receiving objects: 100% (153/153), 1.19 GiB

real    3m49.052s
```

Almost four minutes to check out a single 25MB file!

```
$ du --max-depth=0 --human-readable noise-over-git/.
1.3G    noise-over-git/.
$ ^ 😡
```

And 50 revisions of that single 25MB file eat 1.3GB of space.

But a partial clone side-steps these problems:

```
$ git config --global alias.pclone 'clone --filter=blob:limit=100k'
$ time git pclone https://github.com/thcipriani/noise-over-git
Cloning into '/tmp/noise-over-git'...
...
Receiving objects: 100% (1/1), 24.03 MiB

real    0m6.132s
$ du --max-depth=0 --human-readable noise-over-git/.
49M     noise-over-git/
$ ^ 😻 (the same size as a git lfs checkout)
```

My filter made cloning 97% faster (3m 49s → 6s), and it reduced my checkout size by 96% (1.3GB → 49M)!

But there are still some caveats here.

If you run a command that needs data you filtered out, Git will need to make a trip to the server to get it. So, commands like `git diff`, `git blame`, and `git checkout` will require a trip to your Git host to run.

But, for large files, this is the same behavior as Git LFS.

Plus, I can't remember the last time I ran `git blame` on a PNG 🙃.

Why go to the trouble? What's wrong with Git LFS?

Git LFS foists Git's problems with large files onto users.

And the problems are significant:

- 🖕 **High vendor lock-in** – When GitHub wrote Git LFS, the other large file systems—Git Fat, Git Annex, and Git Media—were agnostic about the server-side. But GitHub locked users to their proprietary server implementation and charged folks to use it.[1]
- 💸 **Costly** – GitHub won because it let users host repositories for free. But Git LFS started as a paid product. Nowadays, there's a free tier, but you're dependent on the whims of GitHub to set pricing. Today, a 50GB repo on GitHub will cost $40/year for storage. In contrast, storing 50GB on Amazon's S3 standard storage is $13/year.
- 😰 **Hard to undo** – Once you've moved to Git LFS, it's impossible to undo the move without rewriting history.
- 🌀 **Ongoing set-up costs** – All your collaborators need to install Git LFS. Without Git LFS installed, your collaborators will get confusing, metadata-filled text files instead of the large files they expect.

The future: Git large object promisors

Large files create problems for Git forges, too.

GitHub and GitLab put limits on file size[2] because big files cost more money to host. Git LFS keeps server-side costs low by offloading large files to CDNs.

But the Git project has a new solution.

Earlier this year, Git merged a new feature: **large object promisers**. Large object promisors aim to provide the same server-side benefits as LFS, minus the hassle to users.

> *This effort aims to especially improve things on the server side, and especially for large blobs that are already compressed in a binary format.*
>
> *This effort aims to provide an alternative to Git LFS*
>
> *– Large Object Promisors, git-scm.com*

What is a large object promisor?

Large object promisors are special Git remotes that only house large files.

In the bright, shiny future, large object promisors will work like this:

1. You push a large file to your Git host.
2. In the background, your Git host offloads that large file to a large object promisor.
3. When you clone, the Git host tells your Git client about the promisor.
4. Your client will clone from the Git host, and automagically nab large files from the promisor remote.

But we're still a ways off from that bright, shiny future.

Git large object promisors are still a work in progress. Pieces of large object promisors merged to Git in March of 2025. But there's more to do and open questions yet to answer.

And so, for today, you're stuck with Git LFS for giant files. But once large object promisors see broad adoption, maybe GitHub will let you push files bigger than 100MB.

The future of large files in Git is Git.

The Git project is thinking hard about large files, so you don't have to.

Today, we're stuck with Git LFS.

But soon, the only obstacle for large files in Git will be your half-remembered, ominous hunch that it's a bad idea to stow your MP3 library in Git.

Edited by Refactoring English

1. Later, other Git forges made their own LFS servers. Today, you can push to multiple Git forges or use an LFS transfer agent, but all this makes set up harder for contributors. You're pretty much locked-in unless you put in extra effort to get unlocked.↩

2. File size limits: 100MB for GitHub, 100MB for GitLab.com↩

Add a comment (Comment Policy)

Copyright © 2025 Tyler Cipriani

Last edited Fri 2025-08-15 08:05:00 PM Created Fri 2025-08-15 07:55:33 PM