The Git project began nearly 15 years ago, on April 7, 2005, and is now the version control system of choice. Yet, there are certain types of projects that often do not use Git, particularly projects that have many large binary files, such as video games. One reason projects with large binary files don't use Git is because, when a Git repository is cloned, Git will download every version of every file in the repository. For most use cases, downloading this history is a useful feature, but it slows cloning and fetching for projects with large binary files, assuming the project even fits on your computer.

Partial Clone is a new feature of Git that replaces Git LFS and makes working with very large repositories better by teaching Git how to work without downloading every file. Partial Clone has been years in the making, with code contributions from GitLab, GitHub, Microsoft and Google. Today it is experimentally available in Git and GitLab, and can be enabled by administrators (docs).

Partial Clone speeds up fetching and cloning because less data is transferred, and reduces disk usage on your local computer. For example, cloning `gitlab-com/www-gitlab-com` using Partial Clone (`--filter=blob:none`) is at least 50% faster, and transfers 70% less data.

Note: Partial Clone is one specific performance optimization for very large repositories. Sparse Checkout is a related optimization that is particularly focused on repositories with tremendously large numbers of files and revisions such as Windows code base.

A brief history of large files

"What about Git LFS?" you may ask. Doesn't LFS stand for "large file storage"?

Previously, extra tools were required to store large files in Git. In 2010,git-annex was released, and five years later in 2015, Git LFS was released. Both git-annex and Git LFS added large file support to Git in a similar way: Instead of storing a large file in Git, store a pointer file that links to the large file. Then, when someone needs a large file, they can download it on-demand using the pointer.

The criticism of this approach is that there are now two places to store files, in Git or in Git LFS. Which means that everyone must remember that big files need to go in Git LFS to keep the repository small and fast. There are downsides to this approach. Besides being susceptible to human error, the pointer encodes decisions based on bandwidth and file type into the structure of the repository that influence all the people using the repository. Our assumptions about bandwidth and storage are likely to change over time, and vary by the location, but decisions encoded in the repository are not flexible. Administrators and developers alike benefit from flexibility in where to store large files, and which files to download.

Partial Clone solves these problems by removing the need for two classes of storage, and special pointers. Let's walk through an example to understand how.

Getting started with Partial Clone

Let's continue to use `gitlab-com/www-gitlab-com` as an example project, since it has quite a lot of images. For a larger repository, like a video game with detailed textures and models, the benefits will be even more significant.

Instead of a vanilla `git clone`, we will include a filter spec which controls what is excluded when fetching data. In this situation, we just want to exclude large binary files. I've included `--no-checkout` so we can more clearly observe what is happening.

```
git clone --filter=blob:none --no-checkout git@gitlab.co
m/gitlab-com/www-gitlab-com.git
# Cloning into 'www-gitlab-com'...
# remote: Enumerating objects: 624541, done.
# remote: Counting objects: 100% (624541/624541), done.
# remote: Compressing objects: 100% (151886/151886), don
e.
# remote: Total 624541 (delta 432983), reused 622339 (del
ta 430843), pack-reused 0
# Receiving objects: 100% (624541/624541), 74.61 MiB | 8.
14 MiB/s, done.
# Resolving deltas: 100% (432983/432983), done.
```

Above we explicitly told Git not to checkout the default branch. Normally `checkout` doesn't require fetching any data from the server, because we have everything locally. When using Partial Clone, since we are deliberately not downloaded everything, Git will need to fetch any missing files when doing a checkout.

```
git checkout master
# remote: Enumerating objects: 12080, done.
# remote: Counting objects: 100% (12080/12080), done.
# remote: Compressing objects: 100% (11640/11640), done.
# remote: Total 12080 (delta 442), reused 9773 (delta 40
9), pack-reused 0
# Receiving objects: 100% (12080/12080), 1.10 GiB | 8.49
 MiB/s, done.
# Resolving deltas: 100% (442/442), done.
# Updating files: 100% (12342/12342), done.
# Filtering content: 100% (3/3), 131.24 MiB | 4.73 MiB/s,
done.
```

If we checkout a different branch or commit, we'll need to download more missing files.

```
git checkout 92d1f39b60f957d0bc3c5621bb3e17a3984bdf72
# remote: Enumerating objects: 1968, done.
# remote: Counting objects: 100% (1968/1968), done.
# remote: Compressing objects: 100% (1953/1953), done.
# remote: Total 1968 (delta 23), reused 1623 (delta 15),
 pack-reused 0
# Receiving objects: 100% (1968/1968), 327.44 MiB | 8.83
 MiB/s, done.
# Resolving deltas: 100% (23/23), done.
# Updating files: 100% (2255/2255), done.
# Note: switching to '92d1f39b60f957d0bc3c5621bb3e17a3984
bdf72'.
```

Git remembers the filter spec we provided when cloning the repository so that fetching updates will also exclude large files until we need them.

```
git config remote.origin.promisor
# true

git config remote.origin.partialclonefilter
# blob:none
```

When committing changes, you simply commit binary files like you would any other file. There is no extra tool to install or configure, no need to treat big files differently to small files.

Network and Storage

If you are already using Git LFS today, you might be aware that large files are stored and transferred differently to regular Git objects. On GitLab.com, Git LFS objects are stored in object storage (like AWS S3) rather than fast attached storage (like SSD), and transferred over HTTP even when using SSH for regular Git objects. Using object storage has the advantage of reducing storage costs for large binary files, while using simpler HTTP requests for large downloads allows the possibility of resumable and parallel downloads.

Partial Clone alreadysupports more than one remote, and work is underway to allow large files to be stored in a different location such as object storage. Unlike Git LFS, however, the repository or instance administrator will be able to choose which objects should be stored where, and change this configuration over time if needed.

Follow the epic for improved large file storage to learn more and follow our progress.

Performance

When fetching new objects from the Git server using a filter spec to exclude objects from the response, Git will check each object and exclude any that match the filter spec. In Git 2.25, the most recent version, filtering has not been optimized for performance.

Jeff King (Peff) (GitHub) recently contributedperformance improvements for blob size filtering, which will likely be included in Git 2.26, and our plan is to include it in GitLab 12.10 release.

Optimizing the sparse filter spec option (`--filter:sparse`), which filters based on file path is more complex because blobs, which contain the file content, do not include file path information. The directory structure of a repository is stored in tree objects.

Follow the epic for partial clone performance improvements to learn more and follow our progress.

Usability

One of the drawbacks of Git LFS was that it required installing an additional tool. In comparison, Partial Clone does not require any additional tools. However, it does require learning new options and configurations, such as to clone using the `--filter` option.

We want to make it easy for people get their work done, who simply desire Git to just work. They shouldn't need to work out which is the optimal blob size filter spec for a project? Or what even is a filter spec? While Partial Clone remains experimental, we haven't made any changes to the GitLab interface to highlight Partial Clone, but we are investigating this and welcome your feedback. Please join the conversation on thisissue.

File locking and tool integrations

Any conversation of large binary files, particularly in regards to video games is incomplete without discussing file locking and tooling integrations.

Unlike plain text source code, resolving conflicts between different versions of a binary file is often impossible. To prevent conflicts in binary file editing, an exclusive file lock is used, meaning only one person at a time can edit a file, regardless of branches. If conflicts can't be resolved, allowing multiple versions of a file to be created in parallel on different branches is a bug, not a feature. GitLab already has basic file locking support, but it is really only useful for plain text because it only applies to the default branch, and is not integrated with any local tools.

Local tooling integrations are important for binary asset workflows, to automatically propagate file locks to the local development environment, and to allow artists to work on assets without needing to use Git from the command line. Propagating file locks quickly to local development environments is also important because it prevents work from being wasted before it even happens.

Follow the file locking and integrations epics for more information about what we're working on.

Conclusion

Large files are necessary for many projects, and Git will soon support this natively, without the need for extra tools. Although Partial Clone is still an experimental feature, we are making improvements with every release and the feature is now ready for testing.

Thank you to the Git community for your work over the past years on improving support for enormous repositories. Particularly, thank you to Jeff King (GitHub) and Christian Couder (senior backend engineer on Gitaly at GitLab) for your early experimentation with partial clone, Jonathan Tan (Google) and Jeff Hostetler (Microsoft) for contributing the first implementation of Partial Clone and promisor remotes, and the many others who've also contributed.

If you are already using partial clone, or would like to help us test partial clone on a large project, please get in touch with me, James Ramsay (group manager, product for Create at GitLab), Jordi Mon (senior product marketing manager for Dev at GitLab), or your account manager.