Troubleshooting the 2 GB push limit - GitHub Docs

4-5 minutes

Learn how to work around the 2 GB push limit.

About the push limit

GitHub has a maximum 2 GB limit for a single push. You might hit this limit when trying to upload very large repositories for the first time, importing large repositories from other platforms, or when trying to rewrite the history of large existing repositories.

If you hit this limit, you may see one of the following error messages:

- `fatal: the remote end hung up unexpectedly`
- `remote: fatal: pack exceeds maximum allowed size`

You can either split up your push into smaller parts, or delete the Git history and start from scratch. If you have made a single commit that's larger than 2 GB and you can't delete the Git history and start from scratch, then you will need to perform an interactive rebase to split the large commit into multiple smaller ones.

Splitting up a large push

You can avoid hitting the limit by breaking your push into smaller parts, each of which should be under 2 GB in size. If a branch is within this size limit, you can push it all at once. However, if a branch is larger than 2 GB, you'll need to split the push into even smaller portions and push only a few commits at a time.

1. If you haven't configured the remote yet, add the repository as a new remote. For more information, see "Managing remote repositories."
2. To find suitable commits spread out along the history of the main branch in your local repository, run the following command:

   ```
   git log --oneline --reverse refs/heads/BRANCH-NAME | awk 'NR %
   1000 == 0'
   ```

   This command reveals every 1000th commit. You can increase or decrease the number to adjust the step size.
3. Push each of these commits one at a time to your GitHub hosted repository.

   ```
   git push REMOTE-NAME +<YOUR_COMMIT_SHA_NUMBER>:refs/heads/BRANCH-
   NAME
   ```

   If you see the message `remote: fatal: pack exceeds maximum allowed size`, reduce the step size in step 2 and try again.

4. Go through the same process for every commit you identified in the history from step 2.
5. If this is the first time this repository is being pushed to GitHub, perform a final mirror push to ensure any remaining refs are pushed up.

   ```
   git push REMOTE-NAME  --mirror
   ```

   If this is still too large, you'll need to push up other branches in stages using the same steps.

Once you're familiar with the procedure, you can automate steps 2 to 4 to simplify the process. For example:

```
step_commits=$(git log --oneline --reverse refs/heads/BRANCH-NAME |
awk 'NR % 1000 == 0')
echo "$step_commits" | while read commit message; do git push REMOTE-
NAME +$commit:refs/heads/BRANCH-NAME; done
```

Starting from scratch

If the repository does not have any history, or your initial commit was over 2 GB on its own and you don't mind resetting the Git history, you can also start from scratch.

1. On your local copy, delete the hidden `.git` folder to remove all the previous Git history and convert it back into a normal folder full of files.
2. Create a new empty folder.
3. Run `git init` and `git lfs install` on the new folder, and add the new empty GitHub repository as a remote.
4. If you already use Git Large File Storage and have all of the Git LFS tracking rules you intend to use already listed in the `.gitattributes` file in the old folder, that should be the first file you copy across to the new folder. You should ensure the tracking rules are in place before you add any other files, so that there's no chance things intended for Git LFS will be committed to regular Git storage.

   If you do not already use Git LFS, you can skip this step, or you can set up the tracking rules you intend to use in the `.gitattributes` file in the new folder before you copy any other files across. For more information, see "Configuring Git Large File Storage."
5. Move batches of files that are smaller than 2 GB from the old folder to the new folder. After each batch is moved, create a commit and push it before moving the next batch. You can take a cautious approach and stick to around 2 GB. Alternatively, if you have a folder with files meant for Git LFS, you can ignore those files when considering the 2 GB limit per batch.

Once the old folder is empty, the GitHub repository should contain everything. If you are using Git LFS, all files meant for Git LFS should be pushed to Git LFS storage.