GitHub Large File Push Issue – Resolution Guide

SCENARIO:

While pushing a branch to GitHub, you may encounter an error like:

```
error: GH001: Large files detected. You may want to try Git Large File Storage - https://git-lfs.github.com
```

This error means one or more files in your commit history exceed GitHub's file size limit of 100MB. GitHub will block the push completely until the large files are removed or handled properly.

STEP 1: IDENTIFY FILES CAUSING THE ERROR

After running `git push`, GitHub usually outputs which files exceed the limit. Look for lines like:

```
remote: error: File path/to/filename.ext is 135.13 MB; this exceeds GitHub's file size limit of 100.00 MB
```

These are the files that need to be removed or managed.

Alternatively, you can check for large files in your repo using:

```
find . -type f -size +90M
```

STEP 2: REMOVE LARGE FILES FROM GIT TRACKING

If you committed large files accidentally and want to keep them locally but remove them from Git:

```
git rm --cached path/to/large_file.ext
```

Then commit the removal:

```
git commit -m "Remove large files to fix GitHub push error"
```

STEP 3: PREVENT FUTURE ISSUES

To avoid tracking these files again, add them to `.gitignore`:

```
echo "*.ext" >> .gitignore    # Example: *.dat or *.zip

git add .gitignore

git commit -m "Ignore large files"
```

STEP 4: PUSH CHANGES

Try pushing your branch again:

```
git push origin your-branch-name
```

ALTERNATIVE SOLUTIONS AND WHEN TO USE THEM

1. GIT LFS (Git Large File Storage)

Use When:

- You need to keep large files in your repo (e.g., datasets, media files).
- You're working on projects where large files are frequently updated.

How to Use:

```
git lfs install

git lfs track "*.ext"

git add .gitattributes

git add your-large-file.ext

git commit -m "Track file using Git LFS"

git push origin your-branch-name
```

Avoid If:

- You do not want to depend on LFS for storage.
- Your team/project does not have Git LFS configured or allowed.

2. GIT FILTER-REPO

Use When:

- Large files have already been committed and exist in your Git history.
- You want to completely erase them from all previous commits.

Install from: https://github.com/newren/git-filter-repo

Example:

```
git filter-repo --path-glob "*.dat" --invert-paths
```

Then:

```
git push origin your-branch-name --force
```

Avoid If:

- You are unsure about rewriting Git history.
- You're working on a shared branch without coordination.

3. BFG REPO CLEANER

Use When:

- You want a simpler tool (compared to `filter-branch`) to clean large files from history.
- Download from: https://rtyley.github.io/bfg-repo-cleaner/

Example:

```
java -jar bfg.jar --delete-files "*.dat"
```

Then:

```
git reflog expire --expire=now --all

git gc --prune=now --aggressive

git push origin --force
```

Avoid If:

- You have not backed up your repo.
- You're working with a very complex commit history or shared production branch.

4. INTERACTIVE REBASE (Manual Method)

Use When:

- The large file was added in the last few commits.
- You prefer to manually drop or modify the offending commits.

Example:

```
git rebase -i HEAD~5
```

Then remove or edit the commits involving the large files.

Avoid If:

- You’re not comfortable working with rebase.
- The large files are deeply buried in history.

SUMMARY

- If the large file is in the latest commit and not needed, use `*git rm --cached*` + `*.gitignore*`.
- If you need large files in Git, use Git LFS.
- If the large file is in commit history, use `*git filter-repo*` or BFG.
- Rebase if the issue is within a few recent commits and you want fine control.

CHECKLIST BEFORE PUSHING

- Run `*find . -type f -size +90M*` to check for big files.
- Add known large files to `*.gitignore*`.
- Clean up large files from staging or history as needed.
- Push only after confirming all files are below 100MB.

TROUBLESHOOTING:

- If you still see the large file error after removing files from the current commit, use `*git log --stat*` to inspect recent commits for large files.
- Use `*git status*` to verify no large files are being tracked.
- If needed, repeat `*git rm --cached <file>*` and commit before pushing.