GitHub and Git LFS: Managing Large Files and Authentication

This guide provides a comprehensive summary of the key steps for using Git LFS to handle large files, managing GitHub authentication, and cleaning Git history when necessary.

1. Handling Large Files with Git LFS

Git Large File Storage (Git LFS) is used to manage large files such as model files, datasets, or binary files, especially those exceeding GitHub's 100 MB limit.

Steps to Set Up Git LFS:

1. **Install Git LFS** in your repository:

   ```
   git lfs install
   ```

2. **Track specific file types** (e.g., `.joblib`, `.pkl`, `.h5`):

   ```
   git lfs track "*.joblib"
   git lfs track "*.pkl"
   git lfs track "*.h5"
   ```

3. **Verify `.gitattributes`** file:
   - Ensure `.gitattributes` has entries like:

     ```
     *.joblib filter=lfs diff=lfs merge=lfs -text
     *.pkl filter=lfs diff=lfs merge=lfs -text
     *.h5 filter=lfs diff=lfs merge=lfs -text
     ```

4. **Add and commit tracked files**:

   ```
   git add .gitattributes
   git commit -m "Track large files using Git LFS"
   git add "path/to/your/large/files/*.joblib"
   git add "path/to/your/large/files/*.pkl"
   git add "path/to/your/large/files/*.h5"
   git commit -m "Add large files using Git LFS"
   ```

5. **Push to GitHub**:

   ```
   git push origin main
   ```

Reference Code for Git LFS Setup

Below is a full reference example for setting up Git LFS in your project:

```
# Step 1: Install Git LFS
$ git lfs install

# Step 2: Track large file types
```

```
$ git lfs track "*.joblib"
$ git lfs track "*.pkl"
$ git lfs track "*.h5"

# Step 3: Verify .gitattributes file
$ cat .gitattributes
# Output should include lines like:
# *.joblib filter=lfs diff=lfs merge=lfs -text
# *.pkl filter=lfs diff=lfs merge=lfs -text
# *.h5 filter=lfs diff=lfs merge=lfs -text

# Step 4: Add and commit tracked files
$ git add .gitattributes
$ git commit -m "Track large files using Git LFS"
$ git add "path/to/your/large/files/*.joblib"
$ git add "path/to/your/large/files/*.pkl"
$ git add "path/to/your/large/files/*.h5"
$ git commit -m "Add large files using Git LFS"

# Step 5: Push to GitHub
$ git push origin main
```

Tips for Managing Large Files:

- **Monitor LFS usage**: GitHub provides a limited amount of free Git LFS storage and bandwidth. Keep an eye on your usage to avoid exceeding your limits.
- **Avoid tracking unnecessary files**: Only track files that exceed GitHub's size limits and are essential for your project to prevent unnecessary storage costs.

2. Overcoming GitHub's 100 MB File Limit

GitHub restricts file sizes for regular commits to **100 MB**. Use Git LFS to store files up to **2 GB** each. Additionally, make sure to track large files before committing them to avoid push rejections.

Reference Code for Handling Large Files

```
# Attempting to add a large file (>100 MB) without LFS will fail.
# To fix this, use Git LFS to track the file type before adding it:
$ git lfs track "*.largefile"
$ git add "largefile.ext"
$ git commit -m "Add large file using Git LFS"
$ git push origin main
```

3. GitHub Authentication

Since **August 13, 2021**, GitHub has removed password-based authentication for Git operations over HTTPS. To continue using Git effectively, consider the following authentication methods.

Use Personal Access Tokens (PATs):

1. **Generate a PAT** via GitHub Personal Access Tokens.

- Select appropriate scopes (e.g., `repo` for repository access).

2. **Use the token** when prompted for a password during HTTPS Git operations.

Reference Code for Using PATs

```
# When pushing to GitHub, you'll be prompted for a username and password.
# Use your GitHub username and paste the PAT as the password.
$ git push origin main
Username: your_github_username
Password: <paste your PAT here>
```

Use SSH Authentication (Recommended):

1. **Generate an SSH key**:

   ```
   ssh-keygen -t ed25519 -C "your_email@example.com"
   ```

   - Press `Enter` to save the key in the default location.
   - Optionally, add a passphrase for extra security.

2. **Add the key to GitHub** at GitHub SSH Settings.
3. **Add SSH key to SSH agent**:

   ```
   eval "$(ssh-agent -s)"
   ssh-add ~/.ssh/id_ed25519
   ```

4. **Update Git remote to use SSH**:

   ```
   git remote set-url origin git@github.com:username/repository.git
   ```

5. **Push to GitHub using SSH**:

   ```
   git push origin main
   ```

Reference Code for SSH Setup

```
# Step 1: Generate SSH Key
$ ssh-keygen -t ed25519 -C "your_email@example.com"

# Step 2: Add SSH key to agent
$ eval "$(ssh-agent -s)"
$ ssh-add ~/.ssh/id_ed25519

# Step 3: Update Git remote URL to use SSH
$ git remote set-url origin git@github.com:username/repository.git

# Step 4: Push changes using SSH
$ git push origin main
```

4. Removing Large Files from History

To remove large files accidentally committed without Git LFS, use **BFG Repo-Cleaner** or **git filter-repo**.

Using BFG Repo-Cleaner:

1. **Clone as a bare repository**:

```
git clone --mirror https://github.com/username/repository.git
```

2. **Run BFG to delete large files**:

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

3. **Clean up and force push the cleaned history**:

```
git reflog expire --expire=now --all
git gc --prune=now --aggressive
git push --force
```

Reference Code for BFG Repo-Cleaner

```
# Step 1: Clone the repository as a bare repository
$ git clone --mirror https://github.com/username/repository.git

# Step 2: Run BFG to delete specific large files
$ java -jar bfg.jar --delete-files "*.joblib"

# Step 3: Clean up and push the cleaned history
$ git reflog expire --expire=now --all
$ git gc --prune=now --aggressive
$ git push --force
```

Using `git filter-repo`:

1. **Install `git filter-repo`**:

```
pip install git-filter-repo
```

2. **Remove large files from history**:

```
git filter-repo --path "*.joblib" --invert-paths
```

3. **Force push the cleaned history**:

```
git push --force
```

Reference Code for `git filter-repo`

```
# Step 1: Install git-filter-repo
$ pip install git-filter-repo

# Step 2: Remove specific large files from history
$ git filter-repo --path "*.joblib" --invert-paths

# Step 3: Force push the cleaned history
$ git push --force
```

Important Considerations:

- **Backup**: Always back up your repository before rewriting history, as these changes are irreversible.
- **Team Coordination**: Communicate with your team before force-pushing to avoid conflicts.

5. Summary

1. **Use Git LFS** for handling large files in your repository.
2. **Authenticate** using either **Personal Access Tokens (PATs)** or **SSH keys**.
3. **Clean history** if large files are mistakenly added without LFS.
4. **Push** changes to GitHub while ensuring that large files are properly tracked with Git LFS.

Useful Links:

- GitHub Personal Access Tokens
- GitHub SSH Setup
- Git LFS Documentation
- BFG Repo-Cleaner
-

Here are the enhanced notes focused on tackling GitHub-related tasks, particularly managing large files using **Git LFS** and dealing with authentication issues:

GitHub and Git LFS Setup for Managing Large Files

This document summarizes the key steps required for setting up Git LFS to handle large files in your GitHub repositories and overcoming common GitHub authentication issues.

1. Track Large Files with Git LFS

Git Large File Storage (Git LFS) is designed to handle large files like model files, datasets, or binary files that exceed GitHub's 100 MB limit for normal commits. Here's how to use it in your project:

Steps to Set Up Git LFS:

1. **Initialize Git LFS in your repository** (if you haven't already):

   ```
   git lfs install
   ```

2. **Track specific large file types**: Use the following command to track file types that are typically large, such as `.joblib`, `.pkl`, or `.h5` files.

   ```
   git lfs track "*.joblib"
   git lfs track "*.pkl"
   ```

3. **Verify `.gitattributes`**: Git LFS will create or modify a `.gitattributes` file in the root of your repository to ensure these files are managed by Git LFS. This file should include lines like:

```
*.joblib filter=lfs diff=lfs merge=lfs -text
*.pkl filter=lfs diff=lfs merge=lfs -text
```

4. **Add `.gitattributes`** to your Git index and commit the changes:

```
git add .gitattributes
git commit -m "Track large files using Git LFS"
```

5. **Add the large files to Git**:

```
git add "path/to/your/large/files/*.joblib"
```

6. **Commit the files**:

```
git commit -m "Add large files using Git LFS"
```

7. **Push to GitHub**:

```
git push origin main
```

   This command will push the changes to your GitHub repository, and the large files will be uploaded using Git LFS.

---

2. Overcoming GitHub's 100 MB File Limit

If you attempt to push files over 100 MB without configuring Git LFS, GitHub will reject the push with an error. By setting up Git LFS (as described above), you ensure that these large files are stored outside the regular Git repository and managed separately by GitHub's LFS infrastructure.

- GitHub's **file size limit** for regular files is **100 MB**.
- **Git LFS** allows you to store files up to **2 GB** each (note that GitHub has storage and bandwidth limits for Git LFS depending on your account plan).

---

3. Handling GitHub Authentication Issues

As of **August 13, 2021**, GitHub removed password-based authentication for Git operations over HTTPS. Here's how to set up modern authentication methods:

Use Personal Access Tokens (PATs)

1. **Generate a Personal Access Token**:
   - Go to GitHub Personal Access Tokens.
   - Click **Generate new token** and select the required permissions (`repo` scope for repository access).
   - **Copy the token** as you won't be able to see it again.
2. **Use the Token for HTTPS Authentication**: The next time Git prompts for your username and password, use:
   - **Username**: Your GitHub username.
   - **Password**: Paste the generated **Personal Access Token** (PAT).

Example:

```
git push origin main
Username: your-github-username
Password: <paste your PAT>
```

Set Up SSH Authentication (Recommended)

1. **Generate an SSH Key**:

   ```
   ssh-keygen -t ed25519 -C "your_email@example.com"
   ```

   - When prompted, press `Enter` to save the key in the default location (`~/.ssh/id_ed25519`).
   - Optionally, enter a passphrase for added security.

2. **Add the SSH Key to GitHub**:

   - Copy the SSH public key to your clipboard:

     ```
     cat ~/.ssh/id_ed25519.pub
     ```

   - Go to GitHub SSH Settings and add the SSH key.

3. **Update Your Git Remote to Use SSH**:

   ```
   git remote set-url origin git@github.com:username/repository.git
   ```

4. **Push to GitHub using SSH**: Now, your authentication will happen via SSH, and you won't be asked for your username and password:

   ```
   git push origin main
   ```

---

4. Force Push After Rewriting History (e.g., Using BFG or Filter-Repo)

If you need to rewrite your Git history to remove large files that were accidentally committed without using Git LFS, you can use tools like **BFG Repo-Cleaner** or **git filter-repo**.

Remove Large Files from History Using BFG Repo-Cleaner:

1. **Install BFG Repo-Cleaner**: Download the BFG jar file from the BFG Repo-Cleaner website.

2. **Clone the repository as a bare repository**:

   ```
   git clone --mirror https://github.com/username/repository.git
   ```

3. **Run BFG to remove large files**:

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

4. **Clean and force-push the changes**:

```
git reflog expire --expire=now --all
git gc --prune=now --aggressive
git push --force
```

Remove Large Files Using `git filter-repo`:

1. **Install `git filter-repo`:**

```
sudo apt install git-filter-repo
```

2. **Remove large files from the history**:

```
git filter-repo --path "*.joblib" --invert-paths
```

3. **Force-push the cleaned history**:

```
git push --force
```

---

5. Summary of GitHub Workflow with Large Files

1. **Track large files** using Git LFS.
2. **Use Personal Access Tokens (PATs)** or **SSH keys** for authentication.
3. **Remove large files from history** if they were committed without LFS.
4. **Push changes** to GitHub, ensuring that large files are handled by Git LFS and that authentication is properly configured.

Certainly! Here's a markdown-formatted reference guide summarizing the main steps we've taken in this process:

Git Repository Cleanup and Configuration Guide

Handling Large Files

1. Remove large files from Git history:

```
git filter-branch --force --index-filter \
"git rm --cached --ignore-unmatch path/to/large/file.ext" \
--prune-empty --tag-name-filter cat -- --all
```

2. Use BFG Repo-Cleaner (alternative method):

```
java -jar bfg.jar --delete-files file_to_delete.ext .
```

3. Clean up refs and optimize the repo:

```
git for-each-ref --format='delete %(refname)' refs/original | git update-ref --stdin
git reflog expire --expire=now --all
git gc --prune=now
```

Updating .gitignore

Add patterns to ignore large files:

```
echo "*.joblib" >> .gitignore
echo "*.pkl" >> .gitignore
git add .gitignore
git commit -m "Update .gitignore to exclude large model files"
```

Force Push Changes

After cleaning up history:

```
git push origin branch_name --force
```

Configuring Git User Information

1. Set username:

   ```
   git config --global user.name "Your Name"
   ```

2. Set email:

   ```
   git config --global user.email "youremail@example.com"
   ```

3. Verify settings:

   ```
   git config --list
   ```

Handling SSH Keys

1. Generate new SSH key:

   ```
   ssh-keygen -t ed25519 -C "your_email@example.com"
   ```

2. Start SSH agent:

   ```
   eval "$(ssh-agent -s)"  # For Unix-based systems
   Start-Service ssh-agent  # For Windows PowerShell
   ```

3. Add SSH key to agent:

   ```
   ssh-add ~/.ssh/id_ed25519  # Path may vary
   ```

4. Add public key to GitHub account (Settings > SSH and GPG keys)

5. Test SSH connection:

   ```
   ssh -T git@github.com
   ```

Committing and Pushing Changes

1. Stage changes:

   ```
   git add .
   ```

2. Commit changes:

   ```
   git commit -m "Commit message"
   ```

3. Pull latest changes:

   ```
   git pull origin branch_name
   ```

4. Push changes:

   ```
   git push origin branch_name
   ```

Troubleshooting

- If encountering issues with large files, consider using Git LFS or storing large files outside the repository.
- Always backup your repository before making significant changes to history.
- Communicate with team members when force pushing or rewriting history.