1. Git LFS (Large File Storage)

Git LFS is an extension to Git that helps handle large files (such as images, audio files, datasets, etc.) in repositories without affecting the performance of Git operations. Git by default isn't efficient at storing large files, which can slow down cloning, pulling, or pushing repositories. Git LFS addresses this by storing large files outside the Git repository and replacing them with lightweight pointers.

Key Points:

- **Installation**:
    - Install Git LFS using the command:

bash

Copy

```
git lfs install
```

- **Tracking Large Files**:
    - To track a file type with LFS, you can use:

bash

Copy

```
git lfs track "*.psd"
```

    - This will add the file pattern to .gitattributes, telling Git LFS to track these types of files.
- **Storing Files**: Once files are tracked, you can add and commit them just like regular Git files:

bash

Copy

```
git add largefile.psd
git commit -m "Add large file"
```

- **Push and Pull**:
    - When you push, Git LFS will upload the large files to an external server.
    - When cloning or pulling, Git LFS will download the actual files instead of the pointer.

Benefits:

- **Efficient storage**: Keeps the Git repository lightweight by storing large files outside the repo.
- **Improved performance**: Reduces overhead for common Git operations like cloning, pulling, or pushing.

---

2. Git Pull Request (PR)

A **pull request (PR)** is a feature that allows developers to propose changes to a codebase in a Git-based platform like GitHub, GitLab, or Bitbucket. It involves a request to merge one branch (usually a feature or bugfix branch) into another (often the main or develop branch).

Key Steps in a Pull Request:

1. **Fork or Branch**: Developers create a separate branch or fork from the main repository to work on new features or fixes.
2. **Commit Changes**: Developers commit changes to the branch or fork.
3. **Create a PR**: Once changes are ready, a PR is created to merge the branch into the target branch (e.g., main or develop).
4. **Review**: Team members or maintainers review the changes and discuss or suggest improvements.
5. **Merge**: Once approved, the changes are merged into the target branch.

Key Features:

- **Code review**: Enables others to review and comment on the changes.
- **Continuous Integration (CI)**: PRs can trigger automated testing and linting to ensure code quality.
- **Conflict resolution**: If there are conflicts between the PR branch and the target branch, the developer must resolve them before merging.

---

3. Git Hook

Git Hooks are custom scripts that Git runs at certain points in its execution. These hooks allow you to automate certain tasks, such as enforcing code standards, running tests, or sending notifications before or after certain Git commands.

Types of Git Hooks:

- **Pre-commit**: Runs before a commit is created. Useful for linting code or running tests.
- **Commit-msg**: Runs after the commit message is written but before the commit is finalized. Used to enforce commit message conventions.
- **Pre-push**: Runs before pushing changes to a remote repository. Useful for tests or checks before pushing.
- **Post-merge**: Runs after a merge operation, useful for notifying or triggering further processes.

Setup:

- Git hooks are found in the .git/hooks/ directory in a Git repository.

- To enable a hook, rename the corresponding script (e.g., pre-commit.sample to pre-commit), and make it executable.

Example (Pre-commit hook):

bash

Copy

```
#!/bin/sh

# Run tests before commit

npm test
```

---

4. Git Config

Git Configuration (git config) is used to set configuration options for your Git environment, including user information, editor preferences, and specific behaviors in Git. There are three levels of configuration:

1. **System-level**: Applied to all users on the system (usually located in /etc/gitconfig).
2. **Global-level**: Applied to the current user (located in ~/.gitconfig or ~/.config/git/config).
3. **Local-level**: Applied to a specific repository (located in .git/config within the repository).

Common Git Config Commands:

- Set global username and email:

bash

Copy

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

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

- Set default editor (e.g., Vim or VSCode):

bash

Copy

```
git config --global core.editor "code --wait"
```

- Set line endings (important for cross-platform compatibility):

bash

Copy

```
git config --global core.autocrlf true
```

You can view all the Git configurations using:

bash

Copy

git config --list

---

5. Git Hooks (Revisited)

As discussed earlier, **Git hooks** are automated scripts that can be used to enforce policies or integrate with external systems. Below are additional examples and details:

Example of Git Hook Workflow:

- **Pre-commit Hook**: To prevent committing code with errors:

bash

Copy

#!/bin/sh

Run eslint before commit

npm run lint

- **Post-commit Hook**: You can trigger a notification after every commit:

bash

Copy

#!/bin/sh

Send Slack notification after commit

curl -X POST -H 'Content-type: application/json' --data '{"text":"Commit successful!"}' https://hooks.slack.com/services/your/slack/webhook

6. Git Authentication with SSH Keys

Git can authenticate with remote repositories via SSH keys, providing secure and passwordless access to Git services (e.g., GitHub, GitLab, Bitbucket).

Steps for Setting Up SSH Authentication:

1. **Generate SSH Key Pair**:
    - If you don't already have an SSH key pair, you can generate one using:

bash

Copy

ssh-keygen -t rsa -b 4096 -C "your_email@example.com"

    - By default, this will create the key in ~/.ssh/id_rsa.
2. **Add SSH Key to SSH Agent**:
    - Start the SSH agent and add the key to it:

bash

Copy

```
eval "$(ssh-agent -s)"

ssh-add ~/.ssh/id_rsa
```

3. **Add SSH Key to Git Hosting Service (e.g., GitHub)**:
    - Copy the public key:

bash

Copy

```
cat ~/.ssh/id_rsa.pub
```

    - Add this key to your Git hosting service's SSH key settings (e.g., GitHub → Settings → SSH and GPG keys → New SSH key).

4. **Testing SSH Authentication**:
    - Test the connection using:

bash

Copy

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

    - If successful, you'll receive a message saying you're authenticated.

Advantages of SSH Authentication:

- **Security**: More secure than using HTTPS with passwords.
- **Convenience**: Once set up, SSH provides passwordless access, making it easier to push, pull, and clone repositories.