Intro to Git Version Control for VLSI Front End Engineers

Version control is an essential practice in modern software and hardware development, enabling teams to manage changes, collaborate effectively, and maintain a history of project evolution. For VLSI front-end engineers, utilizing Git and GitHub can significantly enhance productivity and streamline workflows. This document aims to provide a comprehensive overview of Git version control tailored specifically for VLSI front-end engineers, covering its benefits, best practices, and integration with existing design tools.

Understanding Version Control

What is Version Control?

Version control systems (VCS, not to be confused with Synopsys VCS) are tools that help manage changes to files over time. They allow multiple developers/users to work on the same project simultaneously without overwriting each other's contributions. Git is a Distributed Version Control System (more on this later) that tracks changes in source code during software development but is also applicable to hardware design files, such as HDLs (Hardware Description Language) files used in VLSI Design & Verification.

Why is Version Control Essential?

- **History and Accountability**: With version control, we can view the entire history of changes made to a project, including who made each change and when. This accountability fosters transparency and facilitates collaboration.

- **Collaboration and Teamwork**: Version control enables multiple developers to work on the same project simultaneously without the fear of overwriting each other's work. It allows for seamless collaboration by managing conflicting changes and merging them intelligently.

- **Revert and Rollback**: Mistakes happen (more often than you'd think) but with version control, we can easily revert to a previous state of project if something goes wrong.

- **Branching and Parallel Development**: Version Control systems like Git allows us to create branches, separate lines of development that diverge from the main codebase.

Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification

Types of Version Control Systems

- **Centralized Version Control Systems (CVCS)**:
  These systems have a single central server that stores all versions of files. Developers check out files (sync up) from this Central Repository and commit changes back to it. Examples include Subversion and Perforce.

- **Distributed Version Control Systems (DVCS)**:
  In DVCS like Git, every developer/user has a complete copy of the repository, including its history. This allows for offline work and provides redundancy since any local repository can serve as a backup.

Getting Started with Git

What is Git?

Git is the most widely used Distributed Version Control System software development tool in the industry. It's open-source, distributed and incredibly powerful. It enables offline project work, and when ready, it syncs modifications. It must be installed before we use it.

Installation

1. Open your web browser
2. Navigate to the official Git Website
3. Download the Git installer as per your OS
4. Run the installer and follow on-screen instructions

Once Git is installed, configure your username and email address (to keep track of changes made when in a team):

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

git config –global user.email "youremail@mail.com"
```

Git On Local Machine

Once Git is installed, let's create a new Git repository (on local machine) and track some changes:

1. Open Git Bash Terminal

2. Navigate to your project directory using **cd** command. For example, if your project is on your desktop named Git_test, type:

Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification

```
cd ~/Desktop/Git_test
```

Note: Create a new folder on Desktop named Git_test first or use mkdir command in Git Bash terminal to create a new directory.

3. Initialize a new Git repository in your project directory using the **git init** command:

```
git init
```

This created a hidden .git directory where Git data is stored.

4. Create new files (e.g., design.v, testbench.sv, top.sv etc) in the same directory

5. Open the files in text editor and type-in/add content

6. Add the new file to Git staging area (staging area is like a holding area for changes) using the Git **add** command:

```
git add “file_name”
```

7. Commit your changes to the repository using the **git commit** command:

```
git commit -m “Comments/Messages”
```

8. Check the status of git commit using the **git status** command:

```
git status
```

9. To remove a file, use **git rm** command:

```
git rm –cached “file_name”
```

It's important to note that after removing a file from the repository, you must commit the changes. If you don't commit, the deleted file will still remain in the repository. Follow step 7 to commit it.

10. To Branching from master tree, use **git branch** command:

Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification

```
git branch “name_of_branch”
```

This creates a new branch for working on a specific feature without affecting the main branch (master)

11. Make changes on the new branch (e.g. add new files, content etc).

12. Switch back to the main branch or vice-versa using the **git checkout** command:

```
git checkout master
```

13. Merge the branch and master using the **git merge** command:

```
git merge branch_name
```

Note: Before merging the branch and the master, checkout (switch) to master.

Git On Remote Repository

When collaborating with others, you'll need to push your changes to a remote repository, usually hosted on platforms like GitHub or GitLab, and pull changes from the remote repository to keep your local repository up-to-date. In this guide, we will consider GitHub as our remote repository platform.

1. Set up a repository on GitHub and copy its link.

2. Create a remote repository using the **git remote** command:

```
git remote add <name_of_repo>  <Link_of_GitHub_Repo>
```

Note: You will be prompted to log in to your GitHub account and grant Git permission to access your profile in order to make changes.

3. Push your local commits to the remote repository using **git push** command:

```
git push -u <name_of_repo> master/branch
```

Note: We can have branches in the remote repositories just like we have branches on local machine and the changes can be committed to master or branch.

Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification

4. Pull changes from the repository using **git pull** command:

```
git pull -u <name_of_repo> master/branch
```

This command fetches changes from the remote branch and attempts to merge them into your local branch.

Prajwal N
Design Verification Engineer
github.com/Prajwal-N-6/RTL-Verification