Git and GitHub Complete A to Z Guide

Parth Baldaniya

Contents

1 Introduction **3**

2 What is Git? **3**

3 What is GitHub? **3**

4 Setting Up Git **3**
4.1 Install Git . . . 3
4.2 Configure Name and Email . . . 3
4.3 Set Default Branch Name . . . 3
4.4 Understanding the .git Directory . . . 3

5 Basic Git Commands **4**
5.1 git init . . . 4
5.2 git clone . . . 5
5.3 git add . . . 5
5.4 git commit . . . 5
5.5 git status . . . 5
5.6 git push . . . 5
5.7 git pull . . . 5
5.8 git branch . . . 6
5.9 git checkout . . . 6
5.10 git merge . . . 6
5.11 git log . . . 6
5.12 git diff . . . 6
5.13 git remote . . . 6

6 Advanced Git Commands **7**
6.1 git stash . . . 7
6.2 git rebase . . . 7
6.3 git cherry-pick . . . 7
6.4 git reset . . . 7
6.5 git revert . . . 7
6.6 git tag . . . 7
6.7 git fetch . . . 8
6.8 git blame . . . 8

6.9 git bisect 8
6.10 git reflog 8

7 GitHub-Specific Commands and Features **8**
7.1 Link to GitHub 8
7.2 Fork a Repository 9
7.3 Pull Request 9
7.4 git clone with Submodules 9
7.5 Managing Issues 9
7.6 GitHub Actions 9

8 Common Workflow Example **9**

9 Advanced Workflow: Rebasing and Resolving Conflicts **10**

10 Tips for Beginners and Advanced Users **10**

11 Company Workflow for Interns and New Employees **10**
11.1 Onboarding 11
11.2 Branch Naming Conventions 11
11.3 Ticket System 11
11.4 Project Lifecycle 12
11.5 Pull Request and Code Review 12
11.6 Example: Contributing as an Intern 13
11.7 Tips for Success 14

12 Troubleshooting Common Issues **14**

13 Best Practices for Collaboration **14**

14 Version Control Strategies **15**

15 Effective Code Review Techniques **15**

16 Conclusion **15**

1 Introduction

This guide is your complete resource for learning Git and GitHub, from beginner to expert. Git is a version control system that tracks code changes, allowing you to manage projects efficiently. GitHub is a platform to host, share, and collaborate on code. We provide step-by-step commands, workflows, and tips, explained in simple English with examples to help you master version control and teamwork.

2 What is Git?

Git is a free tool that tracks changes in your code. It saves different versions of your project, lets you go back to old versions, and helps teams work together. Git works locally on your computer.

3 What is GitHub?

GitHub is a website where you store Git projects (called repositories). You can share code, collaborate with others, and make projects public or private. GitHub adds features like pull requests and issues to manage teamwork.

4 Setting Up Git

Before using Git, install it and configure your details.

4.1 Install Git

Download Git from git-scm.com and install it on your computer.

4.2 Configure Name and Email

Tell Git your name and email for commit records.

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

Example:

```
git config --global user.name "John Doe"
git config --global user.email "john.doe@gmail.com"
```

4.3 Set Default Branch Name

Set the default branch name to main (or another name).

```
git config --global init.defaultBranch main
```

4.4 Understanding the .git Directory

When you run git init in a folder, Git creates a hidden .git/ directory to store all the data needed to manage your repository. This directory is the heart of your Git repository,

containing configuration, history, and references. Below is a typical structure of the .git/ directory after running git init:

```
.git/
  config
  HEAD
  hooks/
    prepare-commit.msg.sample
  objects/
    info/
    pack/
  refs/
    heads/
    tags/
```

Here's what each component does:

- **config**: A text file that stores configuration settings for the repository, such as the remote repository URL or user details specific to this project. Example: It might include [remote "origin"] url = https://github.com/username/repository.git.
- **HEAD**: A file that indicates the current branch or commit you're working on (e.g., ref: refs/heads/main points to the main branch).
- **hooks/**: A directory containing scripts that run automatically during certain Git actions (e.g., before a commit). Files like prepare-commitMsg.sample are example hooks you can customize.
- **objects/**: A directory that stores all the data for your repository, including commits, files, and trees. Subdirectories like info/ and pack/ manage metadata and compressed data.
- **refs/**: A directory that stores pointers to commits, such as branches (refs/heads/) and tags (refs/tags/). For example, refs/heads/main points to the latest commit on the main branch.

Understanding the .git/ directory helps you see how Git tracks changes and manages your project. Avoid modifying these files manually unless you're an advanced user, as it can corrupt the repository.

5 Basic Git Commands

These are the most common Git commands for beginners.

5.1 git init

Purpose: Starts a new Git repository in your folder.

```
cd my_project
git init
```

This creates a .git folder to track changes.

5.2 git clone

Purpose: Copies a repository from GitHub to your computer.

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

This downloads the repository to a folder named repository.

5.3 git add

Purpose: Stages files for the next commit.

```
git add index.html
git add .
```

git add . stages all changed files.

5.4 git commit

Purpose: Saves staged changes with a message.

```
git commit -m "Added homepage"
```

Use -m to add a message describing the changes.

5.5 git status

Purpose: Shows the current state of your repository.

```
git status
```

Output:

```
On branch main
Changes not staged for commit:
  modified: index.html
```

5.6 git push

Purpose: Uploads commits to a remote repository (e.g., GitHub).

```
git push origin main
```

Pushes changes to the main branch on GitHub.

5.7 git pull

Purpose: Downloads and merges changes from a remote repository.

```
git pull origin main
```

Updates your local repository with GitHub changes.

5.8 git branch

Purpose: Lists or creates branches. A branch is a separate version of your project.

```
git branch
git branch new-feature
```

First command lists branches; second creates new-feature.

5.9 git checkout

Purpose: Switches branches or restores files.

```
git checkout new-feature
```

Switches to the new-feature branch.

5.10 git merge

Purpose: Combines changes from one branch into another.

```
git checkout main
git merge new-feature
```

Merges new-feature into main.

5.11 git log

Purpose: Shows commit history.

```
git log
```

Output shows commit IDs, authors, dates, and messages.

5.12 git diff

Purpose: Shows differences between changes.

```
git diff
```

Displays changes in unstaged files.

5.13 git remote

Purpose: Manages connections to remote repositories.

```
git remote add origin https://github.com/username/repository.git
git remote -v
```

First command links to GitHub; second lists remotes.

6 Advanced Git Commands

These commands are for more experienced users but explained simply.

6.1 git stash

Purpose: Temporarily saves changes that aren't ready to commit.

```
git stash
git stash pop
```

git stash saves changes; git stash pop restores them.

6.2 git rebase

Purpose: Moves or combines commits to create a cleaner history.

```
git checkout feature
git rebase main
```

Applies feature commits on top of main. Note: Be careful; rebase rewrites history.

6.3 git cherry-pick

Purpose: Copies a specific commit to your current branch.

```
git cherry-pick abc123
```

Applies the commit with ID abc123 to your branch.

6.4 git reset

Purpose: Undoes commits or unstages files.

```
git reset HEAD~1
git reset --hard HEAD~1
```

First command undoes the last commit but keeps changes; –hard discards changes.

6.5 git revert

Purpose: Creates a new commit that undoes a previous commit.

```
git revert abc123
```

Undoes the commit with ID abc123 without changing history.

6.6 git tag

Purpose: Marks a specific commit (e.g., for a release).

```
git tag v1.0
```

```
git push origin v1.0
```

Creates and pushes a tag v1.0.

6.7 git fetch

Purpose: Downloads remote changes without merging.

```
git fetch origin
```

Gets updates from GitHub but doesn't apply them.

6.8 git blame

Purpose: Shows who changed each line in a file.

```
git blame index.html
```

Lists lines with commit IDs, authors, and dates.

6.9 git bisect

Purpose: Finds the commit that caused a bug by binary search.

```
git bisect start
git bisect bad
git bisect good abc123
```

Start bisect, mark current commit as bad, and a known good commit.

6.10 git reflog

Purpose: Shows a log of all actions (useful to recover lost commits).

```
git reflog
```

Lists all actions like commits, resets, and checkouts.

7 GitHub-Specific Commands and Features

These commands and actions connect Git to GitHub or use GitHub features.

7.1 Link to GitHub

Link your local repository to GitHub:

```
git remote add origin https://github.com/username/repository.git
git push -u origin main
```

-u sets origin main as the default push target.

7.2 Fork a Repository

Forking copies someone's GitHub repository to your account. On GitHub, go to the repository and click 'Fork'.

7.3 Pull Request

A pull request asks to merge your changes into the original repository:

1. Push your branch: `git push origin feature`

2. On GitHub, go to 'Pull Requests' and click 'New Pull Request'.

3. Select your branch and submit.

7.4 git clone with Submodules

Purpose: Clones a repository with its submodules (nested repositories).

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

7.5 Managing Issues

Issues track bugs or tasks on GitHub. Create them on the repository's 'Issues' tab. Link commits to issues using #issue-number in commit messages:

```
git commit -m "Fix bug #42"
```

7.6 GitHub Actions

GitHub Actions automates tasks (e.g., testing code). Create a .github/workflows folder with a YAML file to define workflows. Example:

```
name: CI
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Run tests
        run: echo "Running tests"
```

8 Common Workflow Example

Here's a step-by-step example of working on a project:

1. Clone repository: `git clone https://github.com/username/repository.git`

2. Create branch: `git branch feature`

3. Switch branch: `git checkout feature`
4. Edit files.
5. Stage changes: `git add .`
6. Commit: `git commit -m "Added feature"`
7. Push: `git push origin feature`
8. Create a pull request on GitHub.
9. Merge branch: `git checkout main; git merge feature`
10. Delete branch: `git branch -d feature`

9 Advanced Workflow: Rebasing and Resolving Conflicts

If you're working with a team, you may need to rebase and resolve conflicts:

1. Pull latest changes: `git fetch origin; git rebase origin/main`
2. If conflicts occur, Git pauses. Edit conflicting files, then:
3. Stage resolved files: `git add .`
4. Continue rebase: `git rebase -continue`
5. Push changes: `git push -force`

Note: Use –force carefully, as it overwrites remote history.

10 Tips for Beginners and Advanced Users

- **Beginners**: Write clear commit messages. Use git status often. Pull before pushing to avoid conflicts.
- **Advanced**: Use git reflog to recover lost work. Tag releases with git tag. Automate with GitHub Actions.
- Practice on a test repository to learn advanced commands.
- Backup your repository before using rebase or reset.

11 Company Workflow for Interns and New Employees

This section helps interns and new employees understand how to contribute to a software project at a company using Git, GitHub, and various project management tools like Jira, Microsoft Azure DevOps, Trello, or others. It covers the workflow from joining a project to completing a task, including branch naming, ticket systems, code reviews, and the

project lifecycle.

11.1 Onboarding

When you join the company, you'll:

1. **Set Up Git**: Install Git and configure your name and email (see Section 4). Your manager will provide access to the company's GitHub repositories.

2. **Get Tools**: Install tools like VS Code and set up project-specific software (e.g., Node.js, Python). You'll receive a setup guide.

3. **Learn the Workflow**: The company uses a Feature Branch Workflow. All changes are made in feature branches, reviewed via pull requests, and merged into the main branch.

4. **Join the Ticket System**: The company uses a project management tool to track tasks, such as Jira, Azure DevOps, Trello, or similar tools. You'll get an account to view and work on assigned tasks. These tools help organize work and integrate with GitHub to track progress.

11.2 Branch Naming Conventions

The company uses consistent branch names to track work. The format is:

```
<type>/<username>/<ticket-id>-<short-description>
```

- **type**: feature (new functionality), bugfix (fixing bugs), hotfix (urgent production fixes), or release (preparing a release).

- **username**: Your GitHub username or company ID.

- **ticket-id**: The task identifier from the project management tool (e.g., PROJ-123 for Jira, 123 for Azure DevOps, or a Trello card ID like abc123).

- **short-description**: A brief description (e.g., add-login-page).

Examples:

```
feature/johndoe/PROJ-123-add-login-page % Jira
feature/johndoe/123-add-login-page % Azure DevOps
feature/johndoe/abc123-add-login-page % Trello
```

The ticket ID links your code to the task in the project management tool.

11.3 Ticket System

Tasks are tracked in a project management tool, which organizes work into tickets (also called issues, work items, or cards). Common tools include:

- **Jira**: Tasks are called 'issues' (e.g., PROJ-123). You'll update issue statuses (e.g., 'To Do' to 'In Progress') and link commits using the issue ID.

- **Azure DevOps**: Tasks are 'work items' (e.g., 123). You'll use Azure Boards to track progress and link work items to GitHub commits or pull requests.
- **Trello**: Tasks are 'cards' (e.g., abc123). You'll move cards across lists (e.g., 'To Do' to 'Doing') and link them to GitHub issues or commits.
- **Others**: Tools like Asana or ClickUp use similar concepts, with tasks or cards that integrate with GitHub.

How to use the ticket system:

- **Find a Ticket**: Your manager assigns you a task. Check the tool for details like requirements and deadlines.
- **Update Status**: Mark the task as 'In Progress' when you start. Add comments for questions or updates.
- **Link to Code**: Include the ticket ID in your branch name and commit messages (e.g., `git commit -m "PROJ-123: Added login page"` for Jira, or `123: Added login page` for Azure DevOps). This connects your work to the task.

11.4 Project Lifecycle

A project at the company follows these stages:

1. **Kickoff**: The team discusses project goals and creates tasks in the project management tool.
2. **Development**: Developers create feature branches, write code, and commit changes.
3. **Code Review**: Changes are submitted via pull requests and reviewed by team members.
4. **Testing**: Automated tests (via GitHub Actions) and manual testing ensure quality.
5. **Deployment**: Approved changes are merged into main and deployed to production.
6. **Release**: The team tags the release (e.g., `git tag v1.0`) and updates documentation.

11.5 Pull Request and Code Review

To contribute code:

1. **Push Your Branch**: After committing changes, push your branch to GitHub:

   ```
   git push origin feature/johndoe/PROJ-123-add-login-page
   ```

2. **Create a Pull Request**:
   - Go to the repository on GitHub.

- Click 'Pull Requests' > 'New Pull Request.'
- Select your branch and target main.
- Add a title (e.g., PROJ-123: Add Login Page) and description with task details.
- Assign reviewers (e.g., your mentor or senior developer).

3. **Address Feedback**: Reviewers may request changes. Update your branch:

```
git commit -m "PROJ-123: Fixed review comments"
git push
```

4. **Merge**: Once approved, a senior developer merges the pull request into main.

The company requires at least one reviewer approval and passing tests before merging.

11.6 Example: Contributing as an Intern

Suppose you're an intern tasked with adding a login page (task ID PROJ-123 in Jira, 123 in Azure DevOps, or abc123 in Trello):

1. **Set Up**: Clone the repository:

```
git clone https://github.com/company/project.git
```

2. **Create Branch**: Name it per convention, using the task ID from your tool:

```
git checkout -b feature/johndoe/PROJ-123-add-login-page
```

3. **Work on Task**: Write code for the login page. Stage and commit:

```
git add .
git commit -m "PROJ-123: Added login page"
```

   Use the appropriate task ID (e.g., 123 for Azure DevOps, abc123 for Trello).

4. **Push Changes**: Push to GitHub:

```
git push origin feature/johndoe/PROJ-123-add-login-page
```

5. **Open Pull Request**: On GitHub, create a pull request with title PROJ-123: Add Login Page and link to the task in your project management tool (e.g., paste the Jira issue URL, Azure DevOps work item link, or Trello card URL).

6. **Handle Review**: Your mentor requests changes. Update the code, commit, and push again.

7. **Merge and Cleanup**: After approval, the pull request is merged. Delete the branch:

   ```
   git branch -d feature/johndoe/PROJ-123-add-login-page
   ```

8. **Update Task**: Mark the task as 'Done' in the project management tool (e.g., move the Jira issue to 'Done,' update the Azure DevOps work item, or move the Trello card to a 'Done' list).

11.7 Tips for Success

- Learn the specific project management tool used by your company (e.g., Jira, Azure DevOps, Trello). Check its documentation or ask your mentor for guidance.
- Communicate with your team via Slack, Microsoft Teams, or email if you're stuck.
- Check `git status` and `git log` to avoid mistakes.
- Keep pull requests small for easier reviews.
- Always pull the latest main before starting a new branch:

  ```
  git checkout main
  git pull origin main
  ```

12 Troubleshooting Common Issues

- **Merge conflicts**: Edit conflicting files, mark resolved with `git add`, and commit.
- **Push rejected**: Run `git pull` first or use `git push -force` (with caution).
- **Lost commits**: Use `git reflog` to find and restore commits.

13 Best Practices for Collaboration

Effective collaboration ensures smooth teamwork and high-quality code. These practices help teams work efficiently using Git and GitHub.

- **Communicate Early**: Discuss tasks in GitHub Issues or project management tools (e.g., Jira, Trello) to clarify requirements before coding.
- **Write Clear Commit Messages**: Include ticket IDs and describe changes clearly. Example: `git commit -m "PROJ-123: Add login validation"`.
- **Keep Pull Requests Small**: Focus each pull request on a single task to simplify reviews. Push your branch: `git push origin feature/johndoe/PROJ-123-add-login-page`.
- **Sync Regularly**: Pull updates from main to avoid conflicts: `git pull origin main`.

- **Use GitHub Actions**: Automate testing to catch errors early. Example workflow in `.github/workflows/ci.yml`.

14 Version Control Strategies

Choosing the right version control strategy improves project organization and reduces conflicts. Here are common strategies for Git and GitHub.

- **Feature Branch Workflow**: Create a branch for each feature (e.g., `feature/johndoe/PROJ-123-ad` Merge via pull requests.
- **Gitflow**: Use `main` for production, `develop` for integration, and branches like `feature/`, `release/`, or `hotfix/` for specific tasks.
- **Trunk-Based Development**: Commit directly to `main` or short-lived branches for rapid development. Ensure robust testing with `git push origin main`.
- **Tagging Releases**: Mark stable versions with tags: `git tag v1.0; git push origin v1.0`.
- **Resolve Conflicts Promptly**: Use `git fetch origin; git rebase origin/main` to stay updated and handle conflicts early.

15 Effective Code Review Techniques

Code reviews improve code quality and foster team learning. These techniques make reviews productive and collaborative.

- **Provide Specific Feedback**: Suggest clear improvements in pull request comments, referencing code lines and ticket requirements (e.g., PROJ-123).
- **Test Changes Locally**: Clone the branch (`git clone -branch feature/johndoe/PROJ-123-add-` and test before approving.
- **Check for Edge Cases**: Ensure code handles errors and unusual inputs. Verify against task requirements.
- **Be Respectful**: Offer constructive criticism and explain why changes are needed to maintain a positive team environment.
- **Automate Checks**: Use GitHub Actions to run linters and tests, reducing manual review effort. Example: `run: npm test` in workflow.

16 Conclusion

Git and GitHub are essential for managing and sharing code. Start with basic commands, then explore advanced ones as you gain confidence. For more details, visit git-scm.com or docs.github.com.