GitHub

Created	@April 16, 2025 6:33 PM
Category	boost

🌿 GitHub Tutorial – Quick Reference Notes

🔷 Step 1: Clone a Repository

To copy a remote repository to your local machine:

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

> This creates a local folder containing the full project history.

🔷 Step 2: Check the Status

Before and after making changes, it's wise to check the current state of your repository:

```
git status
```

> Shows tracked/untracked files, staged changes, and current branch.

🔷 Step 3: Stage the Changes

To add all modified and new files to the staging area:

```
git add .
```

> . adds everything; alternatively, you can add specific files like git add file.txt.

🔷 Step 4: Commit Your Changes

To commit with a descriptive message:

```
git commit -m "Your message here"
```

> Commits should tell the story of what changed and why.

🔷 Step 5: Push to the Remote Branch

To upload your local commits to GitHub:

```
git push origin main
```

- `origin` – This is the name Git gives to the default remote repository when you clone one. It's a pointer, a shorthand alias for a URL like `https://github.com/yourname/project.git`.

- `main` – This refers to the branch you're pushing. It used to be called `master` in many setups, but now `main` is the new norm—the central timeline, the riverbed of truth.

What is "origin"?

1. Definition

- `origin` is the **default name** Git gives to the **remote repository** from which you cloned your local repo.
- It acts as a **shortcut** to refer to the original repository (usually hosted on GitHub, GitLab, etc.).

2. Key Characteristics

✅ **Default Remote Name** – Assigned automatically when you run `git clone`.

✅ **Points to Source Repo** – Refers to the URL of the original repository.

✅ **Used in Push/Pull** – Helps sync changes between local and remote repos.

3. Common Commands

Command	Description
`git clone <url>`	Clones a repo and sets `origin` to the source URL.
`git remote -v`	Lists all remotes (shows `origin`'s fetch/push URLs).
`git push origin main`	Pushes local `main` branch to `origin`.
`git pull origin main`	Pulls latest changes from `origin/main`.
`git remote rename origin <new-name>`	Renames `origin` to something else.
`git remote add <name> <url>`	Adds a new remote (e.g., `upstream` for forks).

4. Why is it Called "origin"?

- **Git Convention** – Not GitHub-specific; Git uses `origin` as the default remote name.

- **Represents the Source** – The repository from which the clone was made.

5. Modifying "origin"

- **Change the URL of `origin`:**

```
git remote set-url origin <new-url>
```

- **Rename `origin`:**

```
git remote rename origin <new-name>
```

- **Add a New Remote (e.g., `upstream` for forks):**

```
git remote add upstream <original-repo-url>
```

What are Branches?

1. Definition

A **branch** in Git/GitHub is a **parallel version** of a repository that allows you to work on new features, bug fixes, or experiments **without affecting the main codebase** (usually `main` or `master`).

Why Use Branches?

✔ **Isolate changes** – Work independently without breaking the main project.

✔ **Collaborate efficiently** – Multiple developers can work on different features simultaneously.

✔ **Test safely** – Experiment without risking the stable version.

---

2. Key Concepts

A. Default Branch

- Usually named `main` (or `master` in older repos).
- Represents the **production-ready/stable version** of the project.

B. Feature Branch

- A temporary branch for adding a new feature (e.g., `feature/login-page`).
- Merged back into `main` when complete.

C. Release Branch

- Used for preparing a new version (e.g., `release/v1.0`).
- Helps stabilize code before deployment.

D. Hotfix Branch

- Fixes critical bugs in production (e.g., `hotfix/broken-login`).
- Merged into both `main` and development branches.

3. Common Branch Commands

Command	Description
`git branch`	Lists all local branches.
`git branch <name>`	Creates a new branch.
`git checkout <name>`	Switches to a branch.
`git checkout -b <name>`	Creates & switches to a new branch.
`git merge <branch>`	Merges a branch into the current one.
`git branch -d <name>`	Deletes a local branch.
`git push origin <branch>`	Pushes a branch to remote (GitHub).
`git fetch --all`	Updates all remote branches.

4. How Branches Work in GitHub

A. Creating a Branch

1. **Locally:**

```
git checkout -b feature/new-button
```

2. **On GitHub:**
   - Via the **branch dropdown** → "New branch".
   - Or when creating a **Pull Request (PR)**.

B. Syncing Branches

- **Push a new branch to GitHub:**

```
git push origin feature/new-button
```

- **Pull latest changes from a branch:**

```
git pull origin main
```

C. Merging Branches (via Pull Request)

1. Commit & push changes to your branch.
2. Open a **Pull Request (PR)** on GitHub.
3. Reviewers approve → Merge into `main`.

5. Best Practices

✅ **Name branches clearly** – E.g., `fix/header-bug`, `feature/dark-mode`.

✅ **Keep branches short-lived** – Merge quickly to avoid conflicts.

✅ **Delete old branches** – After merging, clean up local & remote branches.

✅ **Always pull latest** `main` – Before creating a new branch to avoid conflicts.

6. Example Workflow

1. Start from `main`:

```
git checkout main
git pull origin main  # Get latest updates
```

2. Create a new feature branch:

```
git checkout -b feature/search-bar
```

3. Make changes, commit, and push:

```
git add .
git commit -m "Add search bar UI"
git push origin feature/search-bar
```

4. Open a **Pull Request (PR)** on GitHub → Merge into `main`.

7. Visualizing Branches

```
main
├── feature/login-page
├── hotfix/broken-link
└── release/v2.0
```

🧹 To delete a branch from the remote repository

```
git push origin --delete <branch-name>
```

Example:

```
git push origin --delete test
```

Explanation:

- This command tells the **remote repository** (usually GitHub) to permanently delete the specified branch.

- It's a **remote operation** — the branch is removed from the remote, not from your local machine.
- Safe to use after merging a branch or cleaning up unneeded remote branches.

🔭 To view all branches on the remote repository

```
git ls-remote --heads origin
```

Explanation:

- Directly queries the **remote** (origin) to list all current branch references.
- It does **not** depend on your local branch list or fetch history.
- Useful when:
  - You've deleted local branches and want to see what still lives remotely.
  - You're troubleshooting or auditing your remote branches.

Undoing Changes in GitHub

GitHub provides several ways to undo changes at different stages of your workflow. Here's a comprehensive guide:

1. Undoing Uncommitted Changes

Discard local changes in a file:

```
git checkout -- <filename>
```

Discard all local changes (unstaged):

```
git restore .
# or
git checkout -- .
```

2. Undoing Staged Changes (Before Commit)

Unstage a file:

```
git reset HEAD <filename>
```

Unstage all files:

```
git reset
```

3. Undoing Commits

Amend the most recent commit (changes message or adds forgotten files):

```
git commit --amend
# Then force push if already pushed to remote:
git push origin <branch> --force
```

Undo last commit but keep changes staged:

```
git reset --soft HEAD~1
```

Undo last commit and unstage changes:

```
git reset HEAD~1
```

Undo last commit and discard changes completely:

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

4. Reverting Published Commits

Create a new commit that undoes changes from a previous commit:

```
git revert <commit-hash>
# Then push normally
git push origin <branch>
```

5. Undoing Multiple Commits

Interactive rebase (to squash, edit, or drop commits):

```
git rebase -i HEAD~n  # where n is number of commits to review
```

6. Undoing a Merge

Undo a merge that hasn't been pushed:

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

Undo a merge that has been pushed:

```
git revert -m 1 <merge-commit-hash>
```

7. GitHub-Specific Undo Options

Using GitHub Desktop:

- Right-click on commit in history → "Revert this commit"
- Undo button for uncommitted changes

On GitHub.com:

- Use the "Revert" button on pull requests or individual commits

- Delete or edit commits directly in the web editor (for small changes)

Important Notes:

- Use `-force` push with caution as it rewrites history
- Coordinate with team when undoing pushed changes
- Consider creating a backup branch before major undo operations
- `git reflog` can help recover lost commits if you make a mistake

Remember that undoing changes becomes more complicated after they've been pushed to a shared repository, so it's best to catch mistakes early.

Git Reset command example

https://www.freecodecamp.org/news/git-reset-hard-how-to-reset-to-head-in-git/