Using GitHub for development | Yessmine

GitHub Commands

why use GitHub? More users/ Documentation which could be helpful to set CI pipeline (git workflow) for the first time

▼ **Cloning a repository**

https://docs.github.com/en/repositories/creating-and-managing-repositories/cloning-a-repository

▼ **Make changes and commit**

```
git status # View the state of the repo
git add <some-file> # Stage a file
git commit -m <some-msg> # Commit a file</some-file>
```

> ⚠ Since these commands create local commits, you can repeat this process as many times as he wants without worrying about what's going on in the central repository.

▼ **Push new commits to central repository**

```
git push origin <branch_name>
```

▼ **Switching branches**

```
git checkout <brancch-name>
```

▼ **Creating a new branch and switch to it**

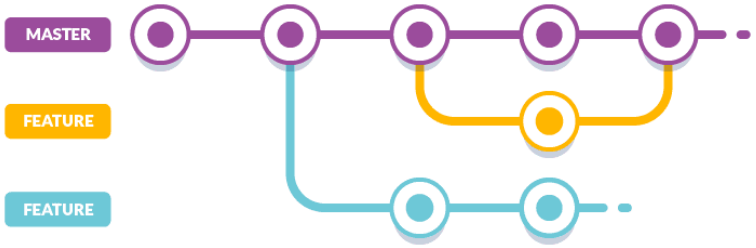
```
git branch NEW_BRANCH_NAME
git chechout NEW_BRANCH_NAME
```

or

```
git checkout -b NEW_BRANCH_NAME
```

GitHub Workflow

[ACTUAL] Feature branch workflow:



Other than the *centralized* workflow, where all changes are committed into the `main` branch without using other branches, the *feature* branch workflow requires all **feature** development to take place in a dedicated branch instead of the `main` branch. The `main` branch should never contain broken code, which is a huge advantage for continuous integration environments.

Life-cycle of a feature branch:

1. **Reset the repository's local copy of main to match the latest version**

This step ensures your local main branch is up-to-date with the remote repository before starting a new feature.

```
git checkout main
git fetch origin
git reset --hard origin/main
```

2. **Create a new-branch**

This command creates a new branch named new-feature and switches to it.

```
git checkout -b new-feature
```

3. **Update, add, commit, and push changes**

Make your changes, add them to the staging area, and commit them with a clear message.

```
git status
git add <some-file>
git commit -m "Meaningful commit message"
```

4. **Push feature branch to remote**

This command pushes your local new-feature branch to the remote repository.

```
git push -u origin new-feature
```

5. **File a pull request on GIT GUI**
    a. Navigate to your repository on the Git GUI
    b. Create a pull request (PR) asking to merge `your-feature` into `main`
6. **Merge branch to main**

After the pull request is approved and any required reviews or checks are completed:

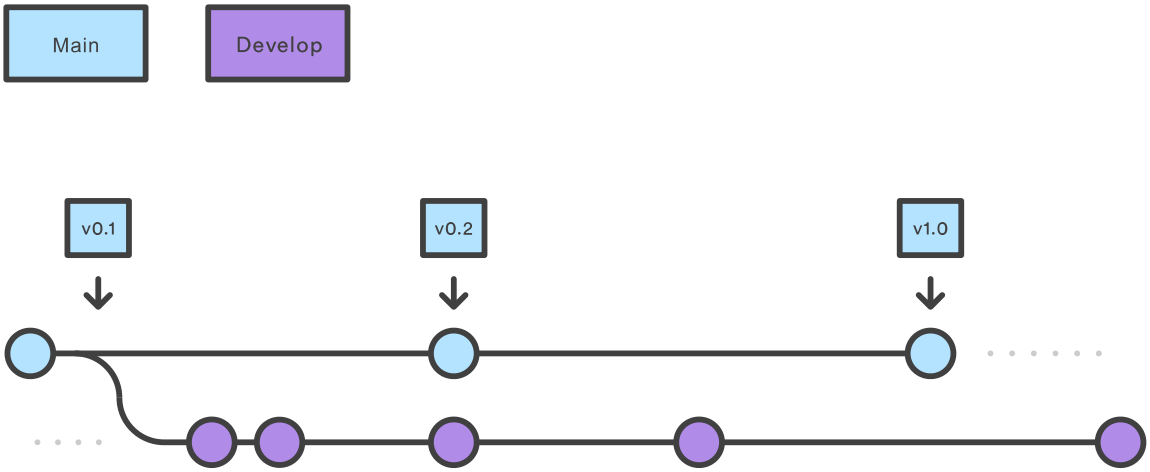
```
git checkout main
git pull
git pull origin new-feature
git push
```

7. **Delete feature branch**

Once the feature branch is merged and no longer needed:

```
git branch -d new-feature   # Deletes the local feature branc
git push origin --delete new-feature   # Deletes the remote f
```

▼ [DEPRACATED] Gitflow Workflow



Instead of a single `main` branch, this workflow uses two branches to record the history of the project. The `main` branch stores the official release history, and the `develop` branch serves as an integration branch for features.

> 🚫 This workflow is deprecated since it couldn't be maintained.

GitHub guidelines

- All commits, except for documentation updates, must be merged from a `new-feature` branch following an approved pull request. Make sure to refer to follow the **Life-cycle of a feature branch guidelines**.
- Avoid direct `git push` from the Raspberry Pi to prevent merging issues. Use `git pull` for testing purposes only.
- **Atomic Commits:** Ensure each commit reflects a single, logically distinct change. This approach facilitates easier review, rollback, and comprehension of changes over time.
- **Commit Messages:** Provide clear and descriptive messages that clarify both the purpose and rationale behind each change.