day-3

Git Branches and Pull Requests: A Comprehensive Guide

Overview of Today

- [ ] Problems that Git branches solves
- [ ] Git Branches, What are they, how to use them, when to use
- [ ] Pull Requests, What are they, how to use, when to use
- [ ] Problems that Pull Requests solves
- [ ] A Full Tutorial with an example repository

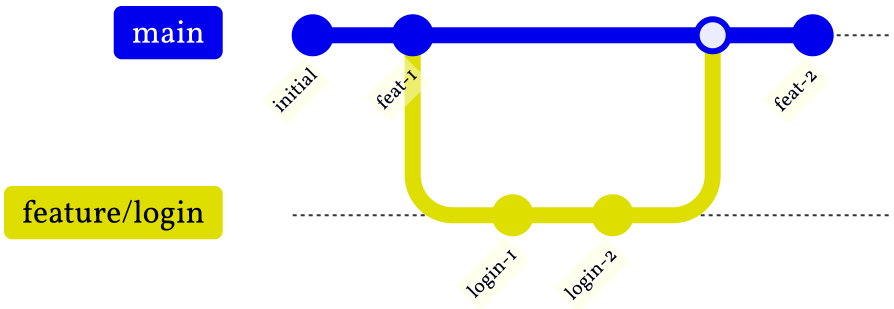
Resources that'll help you

- Git Branching Tutorial An interactive way to learn Git branching, here you can master all edge cases that we won't have time to cover here.
- GitHub Pull Request Guide Official GitHub documentation
- Pull Requests in 100 Seconds Pull-requests tutorial in 100 seconds

---

Understanding Git Branches

- Git branches are fundamental to modern software development workflows.
- They allow developers to work on features, fixes, and experiments without affecting the main codebase.
- Think of branches as parallel universes where different versions of your code can exist simultaneously.



Problems Git Branches Solve

1. Multiple developers can work on different features simultaneously without interfering with each other
2. Experimental features can be developed without risking the stability of the main code
3. Different versions of the software can be maintained concurrently
4. Changes can be organized and reviewed more effectively
5. Roll backs and version control become more manageable

Working with Branches

Here are the essential Git branch commands you'll use regularly:

```
# Create and switch to a new branch
git checkout -b feature-name

# Switch between existing branches
git checkout branch-name

# List all branches
git branch

# Delete a branch
git branch -d branch-name

# Merge a branch into current branch
git merge branch-name
```

Branch Naming Conventions

A good branch name should be:

1. Descriptive of the work being done
2. Use hyphens (-) to separate words

3. Start with a category prefix

Examples:

```
+ feature/user-authentication
+ bugfix/login-error
+ hotfix/security-patch
+ docs/api-documentation

- user auth
- branch331
```

Pull Requests

Pull Requests (PRs) are a way to propose changes to a codebase. They provide a platform for code review, discussion, and quality control before changes are merged into the main codebase.

Problems Pull Requests Solve

1. Code Quality: Changes can be reviewed before being merged
2. Knowledge Sharing: Team members can learn from each other's code
3. Documentation: Changes are documented and discussed in one place
4. Project Management: Work can be tracked and managed effectively

> **Note**
>
> An Example PR workflow Diagram has been inserted at the bottom of this File

Creating a Pull Request

1. Push your branch to the remote repository:

```
git push origin feature-name
```

2. Go to your repository on GitHub
3. Click "Compare & pull request"
4. Fill in the PR description following the template
5. Request reviewers

6. Submit the PR

Pull Request Best Practices

1. Keep PRs small and focused
2. Write clear descriptions
3. Respond to feedback promptly

Tutorial: Working with Branches and Pull Requests

> **Note**
>
> This tutorial is recorded within the Day-3 session, link in Discord

In this tutorial, we will apply what we learned from branching & pull requests by doing some updates to the IT Club website to add the Day 3 Content, You can apply it with any other repo on Github, whether yours or not.
Note: The tutorial will focus on collaborating on existing remote repos.

Any typical Contribution workflow usually has these Steps:

1. Forking
2. Cloning (if you want to open the repo locally)
3. Branching
4. Doing the Changes
5. Staging
6. Committing
7. Pushing
8. Opening a Pull-request
9. Merging
10. Syncing - *Optional*

1. Forking

Done on the Browser, you create a copy of the Repository with the same name in your Account, Example: `GJU-IT-Club/website`, after forking it, you can have a `Your-Username/website` copy.

2. Clone that fork

- Go to your forked Repository (e.g. `Your-Username/website`)
- then to the green button, make sure you're in the *HTTPS* Tab, click the copy button, and you should have an address like this: `https://github.com/Your-Username/website.git`
- In your Terminal, and Navigate to address that you want to create using the `cd` command

```
# Use this to go a folder in Desktop\my-folder
cd Desktop\my-folder
cd .. # Use this to go back one directory
```

```
- Now use the `git clone` command to create a copy of your forked repository
on your local machine.
    ```sh
    git clone https://github.com/Your-Username/website.git
```

3. Branching

by default you'll be on the default branch, `main` in our case.

- Use `git branch` to on which branch you are.
- Now create a branch to the feature you're creating, and switch to it.

```
git checkout -b content/day-3
```

```


#### 4. Apply the Changes
Congrats, you're now on an alternate Universe, apply your Changes.

#### 5. 6. 7. Update your remote forked Repo
Now it's time to stage the files, commit them, then push them to remote.
```sh
git add .
git commit -m 'update: add day 3 Content'
git push origin content/day-3
```

8. Pull-request Time

To create a pull request:
11. Navigate to the original repository (e.g., `GJU-IT-Club/website`) on GitHub
12. Click on the "Pull requests" tab

13. Click the green "*New pull request*" button
14. Click "compare across forks"
15. Set the base repository to the original repo's main branch (e.g., `GJU-IT-Club/website:main`)
16. Set the head repository to your forked repository's branch (e.g., `Your-Username/website:content/day-3`)
17. Review the changes
18. Click "Create pull request"
19. Add a descriptive title and comment explaining your changes
20. Click "Create pull request" again

Example pull request description:

```
## Day 3 Content Update

This pull request adds content from Day 3 to the IT Club website.

### Changes
- Added new section for Day 3 materials
- Updated relevant documentation
- Ensured formatting consistency

### Checklist
- [x] Followed project contribution guidelines
- [x] Tested changes locally
- [x] Reviewed for potential conflicts
```

9. Merging

Now you wait until your changes are merged into `main`, afterwards you can start cleaning

10. Syncing (Optional)

After your pull request is merged, sync your local and forked repositories:

1. Switch to main branch locally:

```
git checkout main
```

1. Add original repository as upstream (upstream means the original repo):

```
git remote add upstream https://github.com/Original-Owner/repository.git
```

2. Fetch & Download upstream changes:

```
git fetch upstream
```

3. Merge upstream changes into your local main:

```
git merge upstream/main
```

4. Push updated main to your forked repository:

```
git push origin main
```

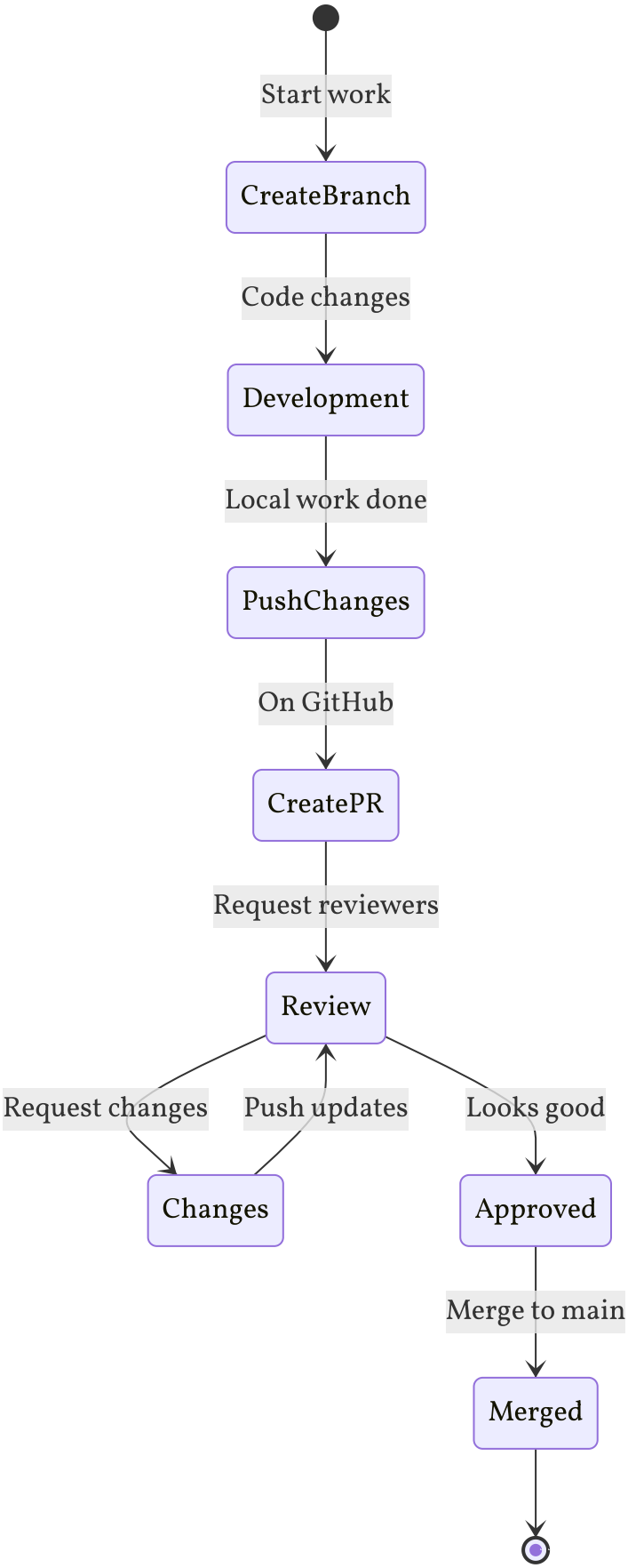
> **Tips**
>
> - Always keep your fork and local repositories synchronized with the upstream repository
> - Use meaningful branch and commit names
> - Write clear pull request descriptions
> - Review changes carefully before merging

Tasks

- [ ] Create a new repository and practice branching
- [ ] Submit your first pull request to an open source project ~ hint: try our website

---

Example PR Flow


Start work
CreateBranch
Code changes
Development
Local work done
PushChanges
On GitHub
CreatePR
Request reviewers
Review
Request changes
Push updates
Looks good
Changes
Approved
Merge to main
Merged