GitHub Branches & Pull Requests

Using the GitHub CLI

1. Cloning the Repository

1. Open a terminal.
2. Run the following command to clone the repository:

```
git clone <repository-url>
cd <repository-name>
```

2. Creating a New Branch

1. Create and switch to a new branch:

```
git switch -c feature-branch-name
```

3. Making Changes and Committing

1. Edit your files using any text editor.
2. Stage the changes:

```
git add .
```

3. Commit the changes with a meaningful message:

```
git commit -m "Brief description of changes"
```

4. Pushing Your Branch

1. Push the branch to GitHub:

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

2. If this is the first time pushing the branch, you may need to set the upstream:

```
git push -u origin feature-branch-name
```

- This links your local branch with the remote branch so future `git push` and `git pull` commands work without needing to specify the branch explicitly.
- After setting the upstream, you can simply use:

```
git push
```

for subsequent pushes.

5. Opening a Pull Request (PR)

1. Go to the repository on GitHub.
2. Click **Pull Requests > New Pull Request**.
3. Select the feature branch as the **source** and `main` as the **target**.
4. Add a clear **title** and **description** explaining what the changes accomplish.
5. Assign a **reviewer** to provide feedback:
   - Click the **Reviewers** section and select the appropriate team member.
   - They will be notified to review the PR and leave comments.
6. Set an **assignee** (yourself or another person responsible for handling the PR):
   - This helps indicate who is responsible for finalizing the PR.
7. Add **tags/labels** (optional but useful):
   - Labels such as `bug`, `feature`, or `enhancement` help categorize the PR.
   - These can be set from the **Labels** section.
8. Click **Create Pull Request** to submit it for review.

6. Reviewing & Approving Code

1. Your partner reviews the PR and adds comments.
2. Address feedback by making additional commits.
3. Once approved, merge the PR:

```
git switch main
git pull origin main
git merge feature-branch-name
```

4. Delete the branch after merging:

```
git branch -d feature-branch-name
git push origin --delete feature-branch-name
```

7. Keeping Your Branch Up to Date

1. Ensure you're on the `main` branch:

```
git switch main
```

2. Pull the latest updates:

```
git pull origin main
```

3. Merge the latest changes into your feature branch:

```
git switch feature-branch-name
git merge main
```

Using IntelliJ IDEA's Built-in Git Features

1. Cloning the Repository

1. Open IntelliJ IDEA and go to **File > New > Project from Version Control**.
2. Select **Git** and paste the repository URL.
3. Choose a local folder and click **Clone**.

2. Creating a New Branch

1. Open the **Git** tool window in IntelliJ IDEA.
2. Click on the branch dropdown in the bottom-right corner.
3. Select **New Branch**, name it descriptively, and create it.

3. Making Changes and Committing

1. Edit your code in IntelliJ IDEA.
2. Open the **Version Control** tool window (Alt+9).
3. Stage the changes by clicking **+** next to modified files.
4. Enter a meaningful commit message.
5. Click **Commit and Push**.

4. Pushing Your Branch

1. Click **VCS > Git > Push** or use **Ctrl+Shift+K**.
2. Select the correct branch and push.

5. Opening a Pull Request (PR)

1. In IntelliJ, click **Git > GitHub > Create Pull Request**.
2. Select the feature branch and `main` as the target.
3. Add a title and description, then submit the PR.
4. Assign a **reviewer** for feedback.

5. Assign an **assignee** to indicate who is responsible.
6. Add **tags/labels** for better categorization.

6. Reviewing & Approving Code

1. Open the PR in GitHub.
2. Review changes and add comments.
3. Address feedback with new commits.
4. Once approved, merge the PR in GitHub.

7. Keeping Your Branch Up to Date

1. Open **Git > Branches**, switch to `main`.
2. Click **Pull** to get the latest updates.
3. Switch back to your feature branch.
4. Click **Merge into Current** to integrate the latest changes.

Best Practices

- Use clear commit messages.
- Keep branches short-lived (merge ASAP).
- Review each other's code before merging.
- Delete branches after merging to keep the repo clean.