# Tech_A git-flow scenario

## Our git flow topic is to make a program about ATM system using Python programming language.

1. **Main Branch**: We begin with the main branch, which serves as the current state of the production code. This branch acts as a protection branch and stores the original source code.
2. **Develop Branch**: We create a develop branch based on the main branch. The develop branch serves as a base for branching out feature branches.
3. **Feature Branches**: According to our Git Flow, we create four feature branches from the develop branch. Each of these branches writes code simultaneously. The feature branches are as follows:
   def (sign_in/sign_out, deposit, withdraw, display).

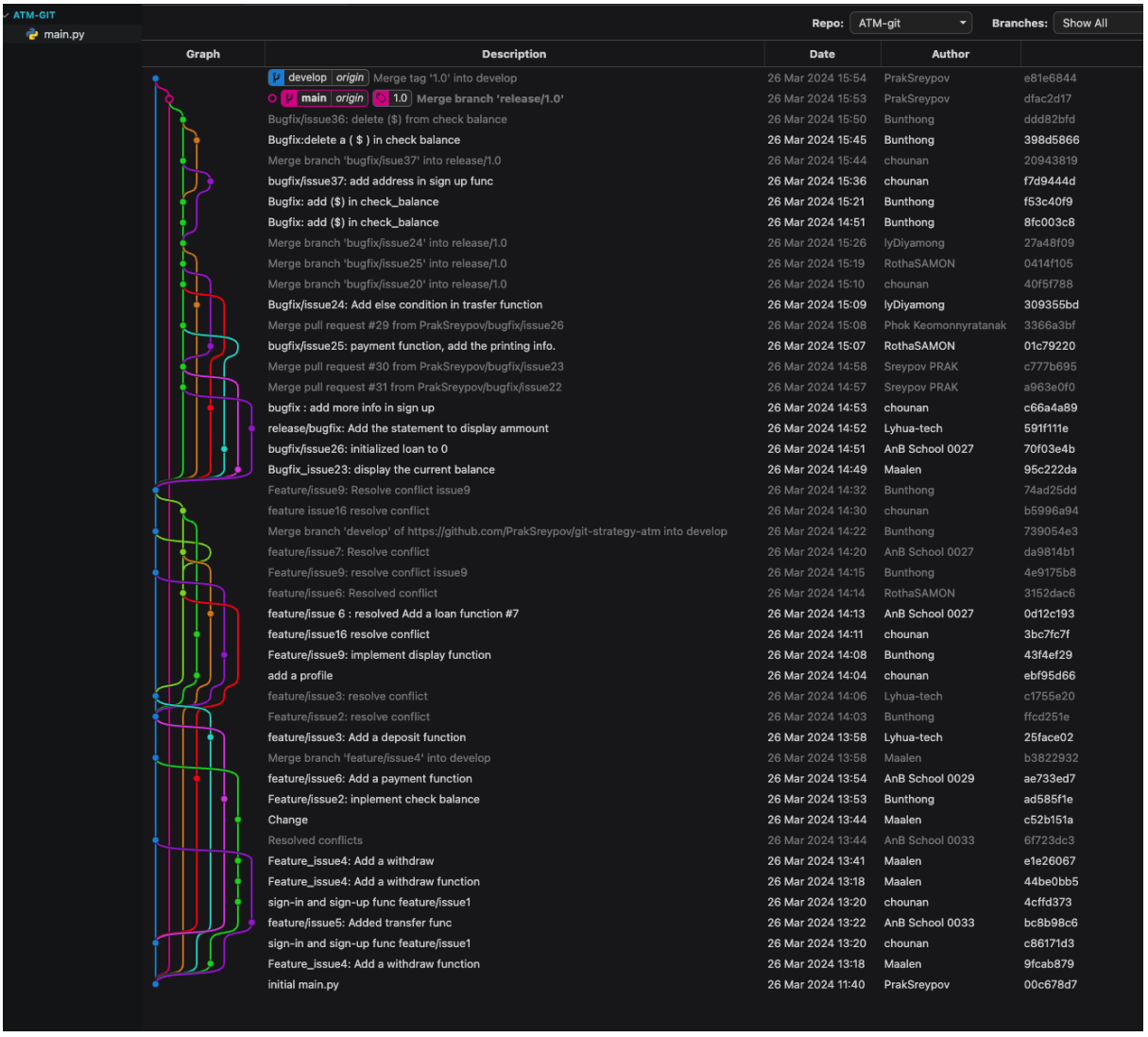
- **Feature 1**: The first team member creates a feature branch named "issue1" and adds sign_in/sign_out function based on the issues assigned by the project leader on GitHub. After adding the functions, the changes are staged, committed to the local repository, and then published to the remote repository. A pull request (PR) is created, waiting for approval. After approval, the branch is merged into the develop branch and pushed to the remote repository.
- **Feature 2**: The second team member creates a feature branch named "issue2" and adds deposit function based on the issues assigned by the project leader on GitHub. After adding the functions, the changes are staged, committed to the local repository, and then rebase from remote repository to get the latest changes that merged from feature1. After rebasing repeat the same steps as Feature 1.
- **Feature 3**: The third team member creates a feature branch named "issue3" and adds withdraw function based on the issues assigned by the project leader on GitHub. After adding the functions, the changes are staged, committed to the local repository, and then rebase from remote repository to get the latest changes that merged from feature1 and feature2. After rebasing repeat the same steps.
- **Feature 4**: The fourth team member creates a feature branch named "issue4" and adds display function based on the issues assigned by the

project leader on GitHub. After adding the functions, the changes are staged, committed to the local repository, and then rebase from remote repository to get the latest changes that merged from feature1, feature2 and feature3. After rebasing repeat the same steps.

4. **Release branch** : After merging all the features into the develop branch, it's time to make to another branch called Release branch that has to release the first version of system. Team leader created a release branch and published it into the remote repository, team member tracked the release branch and create another branch called bugfix to fix some bugs before releasing the first version.
   - After that team members start solving bugs in the bugfix branches, then published it into the remote repository. A pull request (PR) is created, waiting for approval. Upon approval, the branch is merged into the release branch and team member pushed it into the remote repository.
   - Notice: Team members need to rebase to get the latest changes from the previous bugfix.
   - When all bugs have solved, Team leader pulls the changes from remote repository to local, then finish the release branch and push all the changes including tag to the remote repository.
5. **Hotfix branch**: In this scenario, when our system encounters critical issues that require immediate resolution, a Hotfix branch must be created based on the Main branch. Subsequently, the identified errors are addressed and fixed within this hotfix branch.
   - The team leader assigns the issues to a team member to begin working on.
   - The team member creates a hotfix branch based on the main branch and addresses the identified issues within this branch.
   - After fixing the issues, the modified code is added to the staging area, committed to the local repository, and then published.
   - A pull request (PR) is created for the hotfix branch.
   - Once the PR is reviewed and approved, the team member finalizes the hotfix branch by pushing all changes, including tags, to the remote repository. Finally, we got new tag or new version of the program.

To sum up, by implement the scenario of creating ATM system using python programming language to simulate our Git Flow strategy, we can gain a better understanding of how branching and merging work in practice, which can then be applied to real-world projects managed using Git.

# Git Graph




*This is the history of our scenario implementation for simulate our git flow strategy.*