Teach-me session: Git Workflows

Maria L. Pacheco

Purdue Natural Language Processing Group

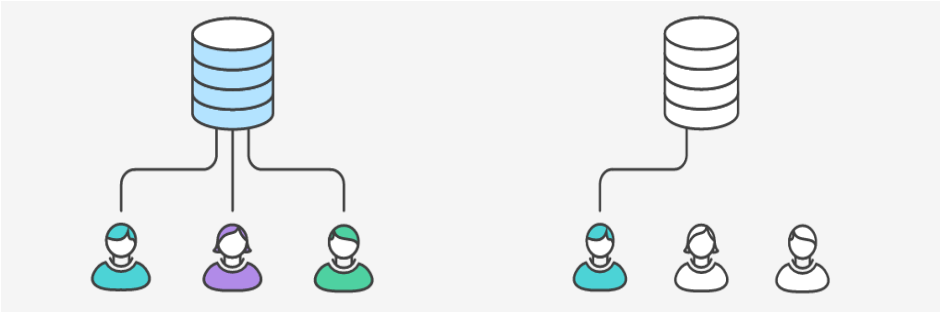
pachecog@purdue.edu

September 1, 2016

Content and images adapted from Atlassian Git Tutorials

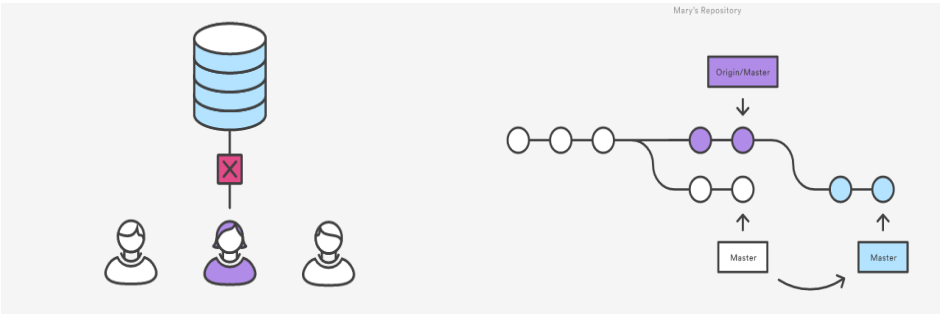
Overview

Centralized Workflow



- Cosmo, Kristen and I-Ta clone the central repository: `master`
- They all work on different features in their local repository
- Once his feature is ready, Cosmo pushes it to the remote repository

Conflicts in Centralized Workflow

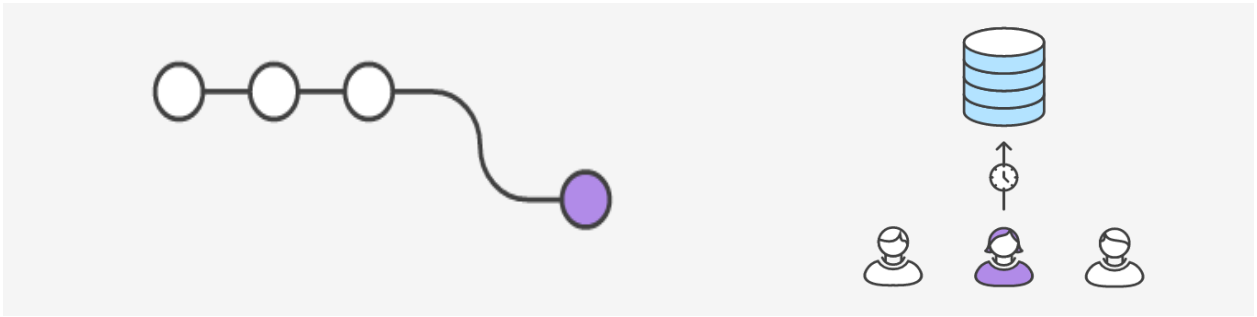


- When Kristen is finished with her feature, she can't directly push her changes to the remote repository
- Kristen needs to pull Cosmo's changes to her local repository first
- Git will try to automatically merge the changes
- If they worked on related code, a conflict might occur. Kristen will need to fix the conflict manually before pushing to the remote repository.

Advantages/Disadvantages of the Centralized Workflow

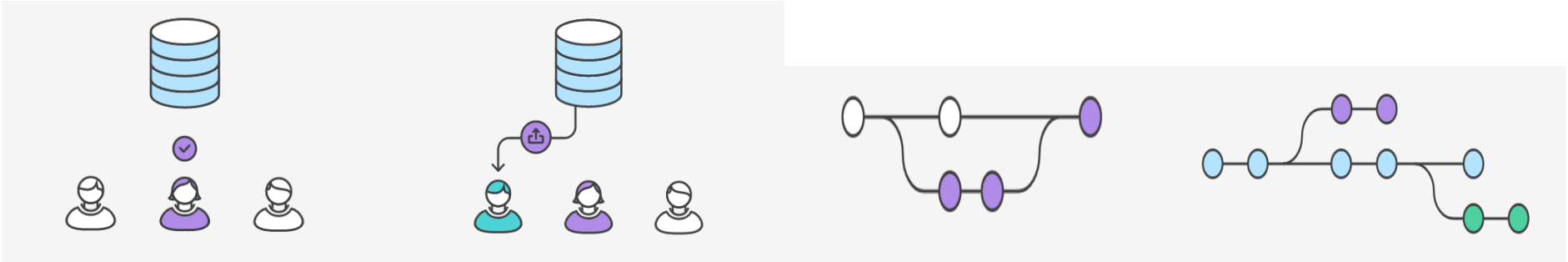
- Very simple workflow, similar to SVN style.
- As the project grows, it becomes hard to initiate discussions based on meaningful features and changes
- We can't keep a stable version if we push work in-progress to `master`
- Conflicts grow with the number of developers involved
- What are the alternatives?

Feature-Branch Workflow



- Di has to work in a new feature for our project, so she clones and pulls the remote repository
- She takes out a branch from her local repository to work on her feature
- After working all Monday she is done with an important part of her feature.
- Pranjal is collaborating with Di and needs to take a look, so Di pushes her work to a remote branch accessible to him.

Merging in Feature-Branch Workflow

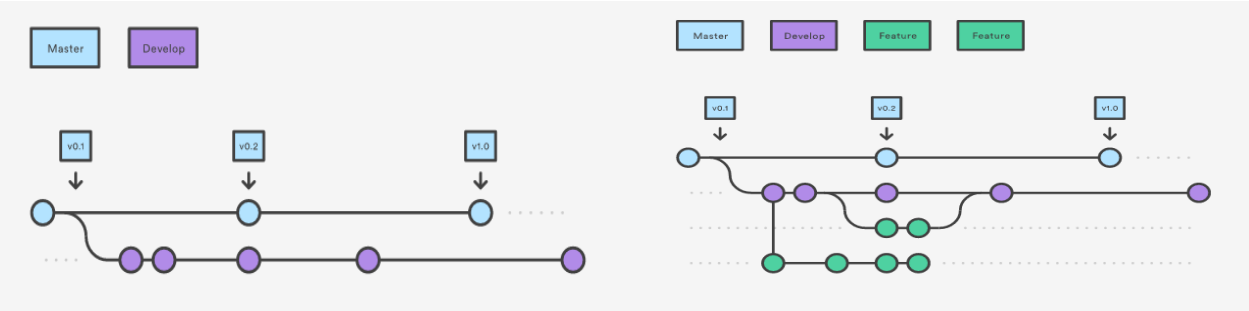


- Di is done with her feature, she first needs to make sure the remote branch has her latest changes by pushing her local changes.
- She now is ready to merge her feature into `master`
- Chang is in charge of maintaining the master branch, so she will file a pull-request to do this merge and he will take a look at it.
- They communicate back and forth to do some changes. Once everything is ready, either of them performs the merge.
- In the mean time, Mahak was doing the exact same thing in her own feature branch

Advantages/Disadvantages of the Feature-Branch Workflow

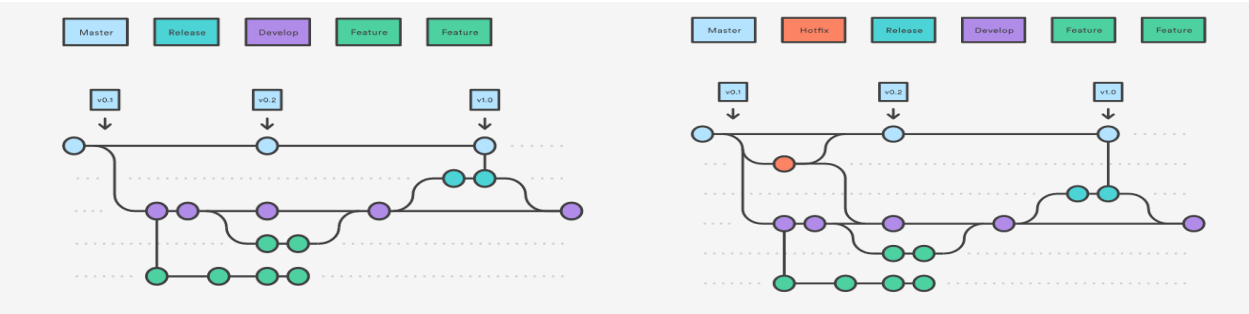
- Everybody can work independently by isolating the work in different branches
- It is trivial to share work when necessary
- Integration work still has to be tested directly on `master`
- Can we find something better?

Gitflow Workflow



- Very similar to Feature-Branch workflow but it changes the branch structure
- `master` records the history of the project, `develop` serves as an integration branch for features

Gitflow Releases and Hotfixes



- Once a version is ready for release, a release branch is created.
- **No new features should be added here**. Only bug fixes, documentation generation, and other release-oriented tasks
- **Hotfixes** are quick patches over bugs found in a release. These are the only branches (other than `develop`) that should be taken directly from `master`

My proposal for PurdueNLP

- A simplified version of Gitflow for PurdueNLP projects
- Using the master-develop branch structure
- Taking feature branches from develop
- We don't need to use pull-requests unless we are working on a software release
- In case of a software release, we adopt the full Gitflow model and pull-requests.
- Open sourcing for other groups to extend our project? **Forking Workflow** (Discussion for another day...)
- What do you guys think?