This lecture gives an introduction about branches in git.

Why are branches needed? How are branches used?

In a large project, many people work on different pieces. It is undesirable that everyone modifies the share repository. Instead, each person should work on a private space until it is ready to share. This is called branch in git.

How can branches be used?

Typically, a project has a master branch. This should be the stable branch. This branch may be used for releasing the software to customers or for long-term support.

New features should be added as feature branches. Feature branches are likely to be unstable because new features are added and tested. These feature branches should be short-term and merged into the master branches soon. Sometimes, a branch is created to fix bugs. These branches are also short-term. The merge process is called pull request.

This figure shows a possible scenario of branch history. At the top is the master branch. Each dot means a commit. As you can see the master branch is not touched often because the master branch should be stable.

Sometimes, a serious problem may be discovered in the master branch. Maybe it is a security problem. In this case, the problem needs to be fixed as soon as possible. This is called hot fix. Hot fix should occur rarely. After the fix, it is merged into the master branch right away.

The more common scenario is a development branch. This is the branch where people contribute their work. This development branch may have multiple feature branches. Most feature branches should be merged into the development branch quickly. It is possible to create new branches for new features. The development branch is not stable because new features are added frequently.

After the development branch reaches a stage where it is ready to release, the development branch is merged with the master branch and this is the stable version.

Howe should git branches be used? Here are some principles.

First, for each commit, use an informative message. The commit messages can provide the trail of the changes.

Before creating any branch, the branch must have a clearly defined purpose. Each branch should solve one problem. Maybe adding one new feature. Maybe fixing one bug. After this branch’s purpose is fulfilled, merge the branch and delete the branch.

A branch should pull from the main branch frequently. The reason is the new features created by other people should be reflected in the feature branch. Depending on the organization, if development branch is used, then software developers should pull from the development branch.

As mentioned earlier, a feature branch should not live for too long. A feature branch should be merged into the development branch after the feature has been implemented and fully tested.

Also, it is important to have experienced members to approve pull requests before features are merged into the main branch. This can ensure that the features merged into the main branches are high quality.

How long should a branch live? There is no fixed rule. A general rule of thumb is to merge and delete a feature branch within two weeks. If a branch is not merged within two weeks, the feature may be too complex and should be divided into smaller units.

In contrast, here are some common mistakes. If commits have no informative messages, soon the repository will be filled by commits without any clear purpose and people will be unable to understand why these commits occur.

Another common mistake is creating new branches without any specific reason. This problem is particularly common among students. At the beginning a semester, a group of students start working on a software project. Each student creates a branch but nobody knows what each branch is for. The problem is that each student will keep a personal branch for too long. If a branch lives too long, the differences among branches become too large and merging them becomes very difficult.

Among all common problems, the next one is the most serious. Many people are afraid of merging branches. The reason is that they are afraid their work has bugs. Or their work may have conflicts. As a result, these people do not merge.

If they do not merge, the differences among branches grow and conflicts will become more likely.

As time goes by, merging becomes more and more difficult. At the end of the semester, the students do not merge their work and the project has no progress. All work by the students are lost because nobody knows how to merge the branches.

A related problem is to trying to solve several problems with only one branch. This is similar to the earlier problem. When one branch tries to do many things, this branch will take long time and the differences between this branch and the other branches can become too large. Merging becomes impossible.

Do not work on multiple branches simultaneously. You will lose focus and get confused.

Another common problem is that a branch is merged without proper inspection. Buggy code gets into the master branch and stays in the branch.

When a team assigns an inexperienced member to approve pull requests, the team is going to have many problems soon.