**Questions about code versioning:**

**Why is branching with Mercurial or git easier than with SVN?**

<https://mentormate.com/blog/differences-git-svn/>

**What are the pros and cons of Distributed Version Control Systems like Git over Centralized ones like SVN?**

*28*

*I don't have a lot of experience with git, but:*

*Pros:*

*It's really quick*

*Local commits rock*

*Quick to start a new repository (no configuration etc)*

*github is easy to use*

*(I haven't really "needed" the distributed side of things yet, beyond being able to have a local repository and push to a public one.)*

*Cons:*

*Windows support is still lagging behind, I believe - and you can't just use it from a normal command prompt*

*Lack of IDE and Explorer integration*

*It took me a while to find a good introductory text along the lines of the redbean book.*

*The fact that "adding" a changed file only adds the contents at that point of time (so it can show up as staged for commit and still have modifications which require another git add) took a while to grasp*

**Could you describe GitHub Flow and GitFlow workflows?**

[*https://stackoverflow.com/questions/18188492/what-are-the-pros-and-cons-of-git-flow-vs-github-flow*](https://stackoverflow.com/questions/18188492/what-are-the-pros-and-cons-of-git-flow-vs-github-flow)

*As discussed in GitMinutes episode 17, by Nicholas Zakas in his article on "GitHub workflows inside of a company":*

*Git-flow is a process for managing changes in Git that was created by Vincent Driessen and accompanied by some Git extensions for managing that flow.*

*The general idea behind git-flow is to have several separate branches that always exist, each for a different purpose: master, develop, feature, release, and hotfix.*

*The process of feature or bug development flows from one branch into another before it’s finally released.*

*Some of the respondents indicated that they use git-flow in general.*

*Some started out with git-flow and moved away from it.*

*The primary reason for moving away is that the git-flow process is hard to deal with in a continuous (or near-continuous) deployment model.*

*The general feeling is that git-flow works well for products in a more traditional release model, where releases are done once every few weeks, but that this process breaks down considerably when you’re releasing once a day or more.*

*In short:*

*Start with a model as simple as possible (like GitHub flow tends to be), and move towards a more complex model if you need to.*

*You can see an interesting illustration of a simple workflow, based on GitHub-Flow at:*

*"A simple git branching model", with the main elements being:*

*master must always be deployable.*

*all changes made through feature branches (pull-request + merge)*

*rebase to avoid/resolve conflicts; merge in to master*

**What's a rebase?**

*In Git, there are two main ways to integrate changes from one branch into another: the merge and the rebase. In this section you’ll learn what rebasing is, how to do it, why it’s a pretty amazing tool, and in what cases you won’t want to use it.*

**Why merges are easier with Mercurial and git than with SVN and CVS?**

[*https://stackoverflow.com/questions/43995/why-is-branching-and-merging-easier-in-mercurial-than-in-subversion*](https://stackoverflow.com/questions/43995/why-is-branching-and-merging-easier-in-mercurial-than-in-subversion)