Lunch & Learn: Exploring Git branching models and Git Flow alternatives

Git is a free, open source, distributed version control system for coordinating changes to a codebase over time across potentially thousands of authors. Git manages distinct contributions through the use of *commits*, which are saved states of a codebase, and branching, which allows for multiple divergent versions of that codebase to exist simultaneously.

Git Flow is a branching strategy first introduced in 2010 by [this blog post](https://nvie.com/posts/a-successful-git-branching-model/), which coordinates developer contributions through the use of three transient branch types – features, releases, and hotfixes – and two permanent branches – *develop*, representing an agreed upon, shared base state from which developers create new features, and *master*, representing the state of an application’s most recent production release. Shortly after its introduction, Git Flow was widely adopted in enterprise software development, and remains one of the most popular branching models in the world today.

In this presentation, I introduce three alternatives to Git Flow, and argue that we should move away from Git Flow towards a simpler branching model. I begin with a low-level look at how Git works, followed by an overview of Git Flow. I then show that Git Flow introduces excess complications into its branching model that both postpone the early resolution of conflicts and increase the likelihood of human error, with the worst problems being caused by the need to coordinate the states of two eternal branches. From there, I show how three different models – GitHub flow, Release Flow, and Trunk-based development – avoid these problems, leading to less time devoted to bug fixes and more time for feature development.