Git and GitHub Workflows

Information to aid your VCS and branching decisions

a discussion with Matthew McCullough
software development
decisions...
decisions…
Platform decisions…
Language decisions…
IDE decisions…
Process decisions…
but the **initial** decisions are...
VCS decisions
basis for other decisions
centralized
de-centralized
GitHub
Team decisions
how many team members?
team member?
1 to 5 team members?
A repository with a long, sordid commit history with elements that can pass a build and fail a Maven build script.

— Edit

3 contributors
5 to 15 team members?
15 to 50 team members?
50 to 150 team members?
150+ team members?
Git Source Code Mirror - This is a publish-only repository and all pull requests are ignored. Please follow Documentation/SubmittingPatches procedure for any of your improvements.

37,387 commits
5 branches
522 releases
813 contributors
Ruby on Rails http://rubyonrails.org

- 46,588 commits
- 37 branches
- 226 releases
- 2,397 contributors
Linux kernel source tree

468,236 commits
1 branch
383 releases
4,082 contributors

branch: master
linux / +
optimization decisions
optimized for fast iterations?

GitHub Flow
master
optimized for release maintenance?

Git Flow
Time

feature branches

develop

release branches

hotfixes

master

Feature for future release

Major feature for next release

Incorporate bug fix in develop

Severe bug fixed for production: hotfix 0.2

From this point on, “next release” means the release after 1.0

Tag 1.0

Tag 0.1

Tag 0.2

Incorporate bug fix in develop

Only bug fixes!

Start of release branch for 1.0

Author: Vincent Driessen

Original blog post: http://nvie.com/posts/a-succesful-git-branching-model

License: Creative Commons BY-SA
From this point on, “next release” means the release after 1.0

Only bugfixes!

Bugfixes from rel. branch may be continuously merged back into develop

Incorporate bugfix in develop

Only bugfixes!

Start of release branch for 1.0

Author: Vincent Driessen

Original blog post: http://nvie.com/posts/a-succesful-git-branching-model

License: Creative Commons BY-SA
command line tooling
optimized for developer on-boarding

atomic flow
optimized for clarity & maintenance

git.git gitworkflows
NAME

gitworkflows - An overview of recommended workflows with git

SYNOPSIS

git *

DESCRIPTION

This document attempts to write down and motivate some of the workflow elements used for `git` itself. Many ideas apply in general, though the full workflow is rarely required for smaller projects with fewer people involved.

We formulate a set of `rules` for quick reference, while the prose tries to motivate each of them. Do not always take them literally; you should value good reasons for your actions higher than manpages such as this one.

SEPARATE CHANGES

As a general rule, you should try to split your changes into small logical steps, and commit each of them. They should be consistent, working independently of any later commits, pass the test suite, etc. This makes the review process much easier, and the history much more useful for later inspection and analysis, for example with `git-blame(1)` and `git-bisect(1)`. 
command decisions
$ git merge master <topic>
$ git revert <SHA>
$ git commit --amend
$ git pull -r
$ git rebase master
$ git rebase -i <SHA>
$ git rebase --autosquash -i <SHA>
$ git merge --squash <topic>
$ git bisect start
$ git bisect good
$ git bisect bad
$ git bisect run
$ git bisect log
$ git reset --hard
$ git reset --soft
$ git reset --mixed
So what workflow should I chose Matthew?
mix and match
the simplest thing that works