WILD CODE SCHOOL

Better Collaboration

Best of Git Flows - when to use which

COMMENT	DATE
CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
ENABLED CONFIG FILE PARSING	9 HOURS AGO
MISC BUGFIXES	5 HOURS AGO
CODE ADDITIONS/EDITS	4 HOURS AGO
MORE CODE	4 HOURS AGO
HERE HAVE CODE	4 HOURS AGO
AAAAAAAA	3 HOURS AGO
ADKFJSLKDFJSDKLFJ	3 HOURS AGO
MY HANDS ARE TYPING WORDS	2 HOURS AGO
HAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

WILD
CODE
SCHOOL

- History of Version Control Systems
- Why Git Flows?
- Git Flow, Github Flow, Feature & Development Branches
- How to choose the best Git flow for your Project

Concepts of Version Management

- ❖ Work with **others** on the **same project**
- ❖ **Return** to an **earlier version** of a file or the complete project
- ❖ **Develop features** more easily in a **team**

Concept and History of VCS

Version management software allows you to store a set of files, keeping the **chronology of all the modifications** that have been made to them.

- 90's : Creation of **CVS** (versioning system)
- Early 2000 : Apache **SVN** (Subversion)
- 2005 : Creation of **Git** by Linus Torvalds

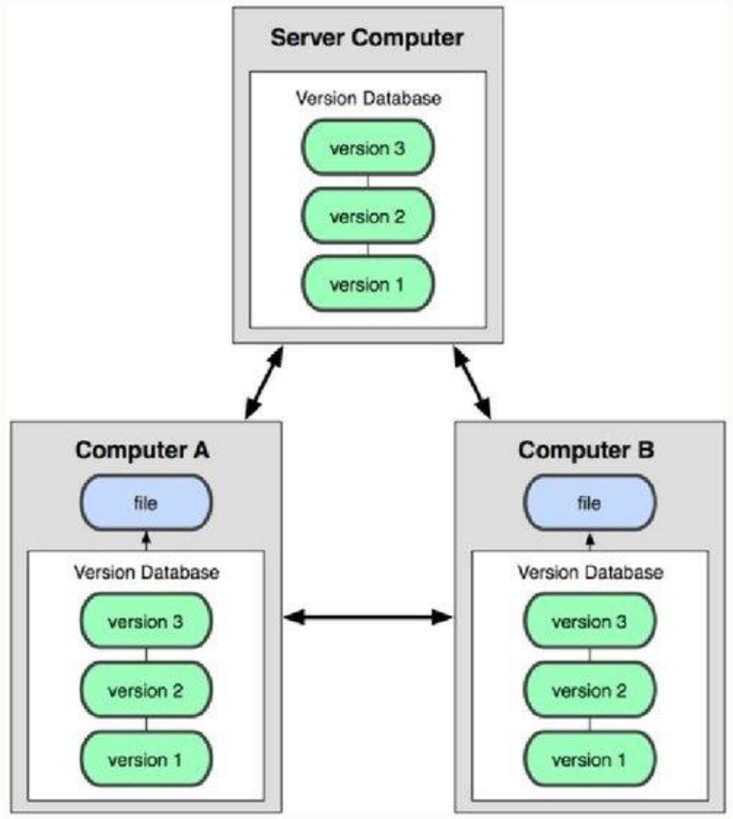
Why Git?

- **Decentralized**
  Allows you to start working right away (no server required).
- **Very fast and reliable**
  Because decentralized
- **Relatively simple, though powerful**
  branch management, creation, merge…

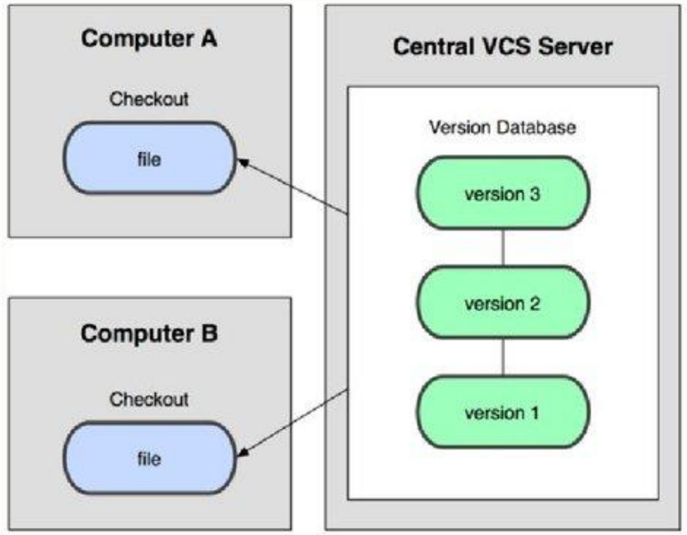
But there are other VCS managers:

- SVN (centralized)
- CVS (centralized)
- Mercurial (decentralized)
- Bazaar (decentralized)…

Centralized VS Decentralized



Decentralized



Centralized

Decentralized VCS

Advantages

- Branch system allows you to **switch from one feature to another** very easily and without interference
- Everything is **decentralized locally**, **no server connection** needed to **manage branches**
- Very effective merging algorithms
- Quick commits

Constraints

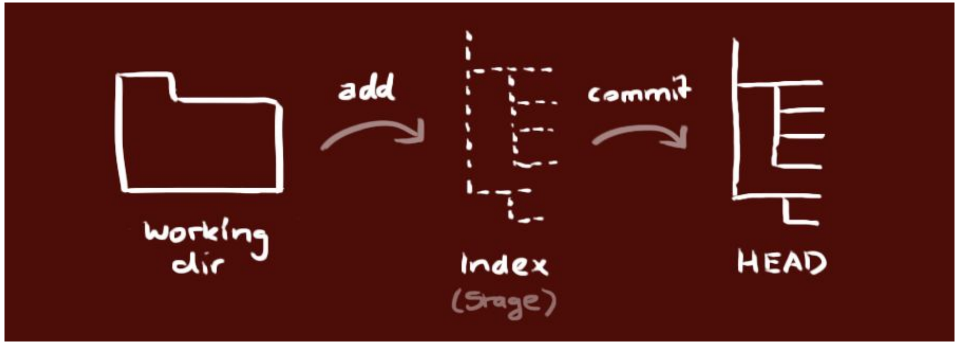
- Overhead for simple operations
- Need to agree on a workflow for each project

How does it work?

1. You make a change in your working directory
2. You **add** your files to the index (snapshots of your files)
3. When your project is in a **clean state** (new feature or full functional part), you make a **commit** that will save your project in the state it is in your index thanks to an *id* and a *comment*.

Regarding your files, Git manages 3 main "trees":

1. Working directory
2. Index (stage)
3. HEAD



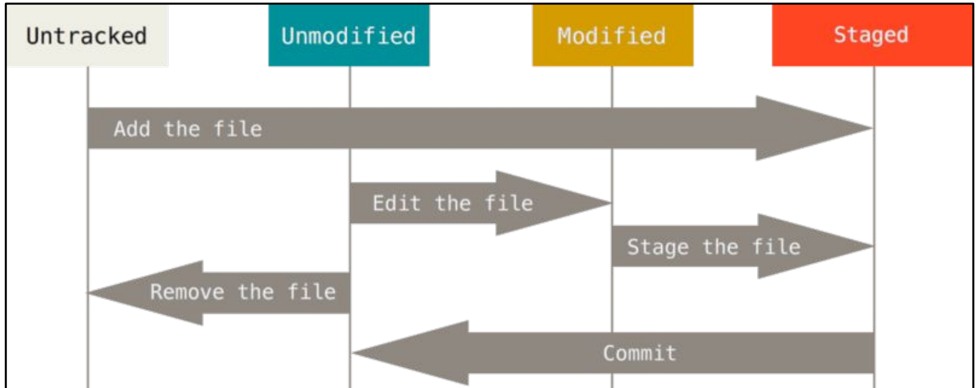
How does it work?

Files in the working directory can only have **2 states**:

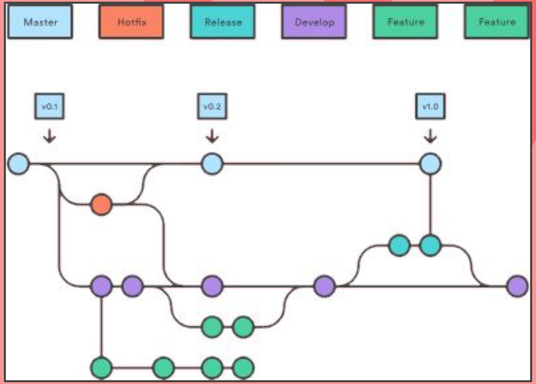
1. **Tracked**
2. **Untracked**

Tracked files can have 3 states:

1. **Modified** (modified but not added to the index)
2. **Unmodified** (committed)
3. **Staged** (modified and added to the index)



Git Workflows



What are Git Workflows?

A Git workflow is a **recipe** or **recommendation** for how to **use Git** to accomplish work in a **consistent and productive manner**.

Git offers a lot of **flexibility** in how users manage changes. Given Git's **focus on flexibility**, there is **no standardized process** on how to interact with Git.

When working with a team on a Git-managed project, it's important to make sure **the team is all in agreement** on how the flow of changes will be applied.

To ensure the team is on the same page, an **agreed-upon Git workflow should be developed** or selected.

https://www.atlassian.com/git/tutorials/comparing-workflows

What are Criteria for Git Flows? (1st Part)

- When evaluating a workflow for your team, it's most important that you consider **your team's culture.**
- Does this workflow **scale with team size**? Does it have to?
- Is it easy to **undo mistakes and errors** with this workflow?
- Does this workflow impose any new **unnecessary cognitive overhead** to the team?

https://www.atlassian.com/git/tutorials/comparing-workflows

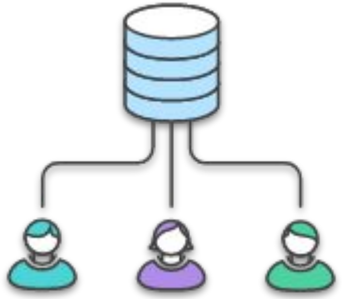
Centralized VCS (SVN)

The Centralized Workflow is a great Git workflow for **teams transitioning from SVN**.

Like Subversion, the Centralized Workflow uses a **central repository** to serve as the **single point-of-entry** for all changes to the project. Instead of **trunk**, the default development branch is called **main** and **all changes are committed into this branch**.

This workflow doesn't require any other branches besides main.

The Centralized Workflow is **great for small teams**. The conflict resolution process detailed above **can form a bottleneck as your team scales in size**.



Feature Branches

The core idea behind the Feature Branch Workflow is that **all feature development should take place in a dedicated branch** instead of the main branch.

Encapsulating feature development also makes it possible to **leverage pull requests**, which are a way to **initiate discussions around a branch**.

Feature Branching is a logical extension of Centralized Workflow. The core idea behind the Feature Branch Workflow is that all feature development should take place in a dedicated branch instead of the main branch. This encapsulation makes it easy for **multiple developers** to work on a particular **feature without disturbing the main codebase**.

It also means the main branch **should never contain broken code**, which is a huge advantage for continuous integration environments.

https://www.atlassian.com/git/tutorials/comparing-workflows

Real Collaboration with Feature Branches

Encapsulating feature development also makes it possible to leverage pull requests, which are a way to **initiate discussions around a branch**.

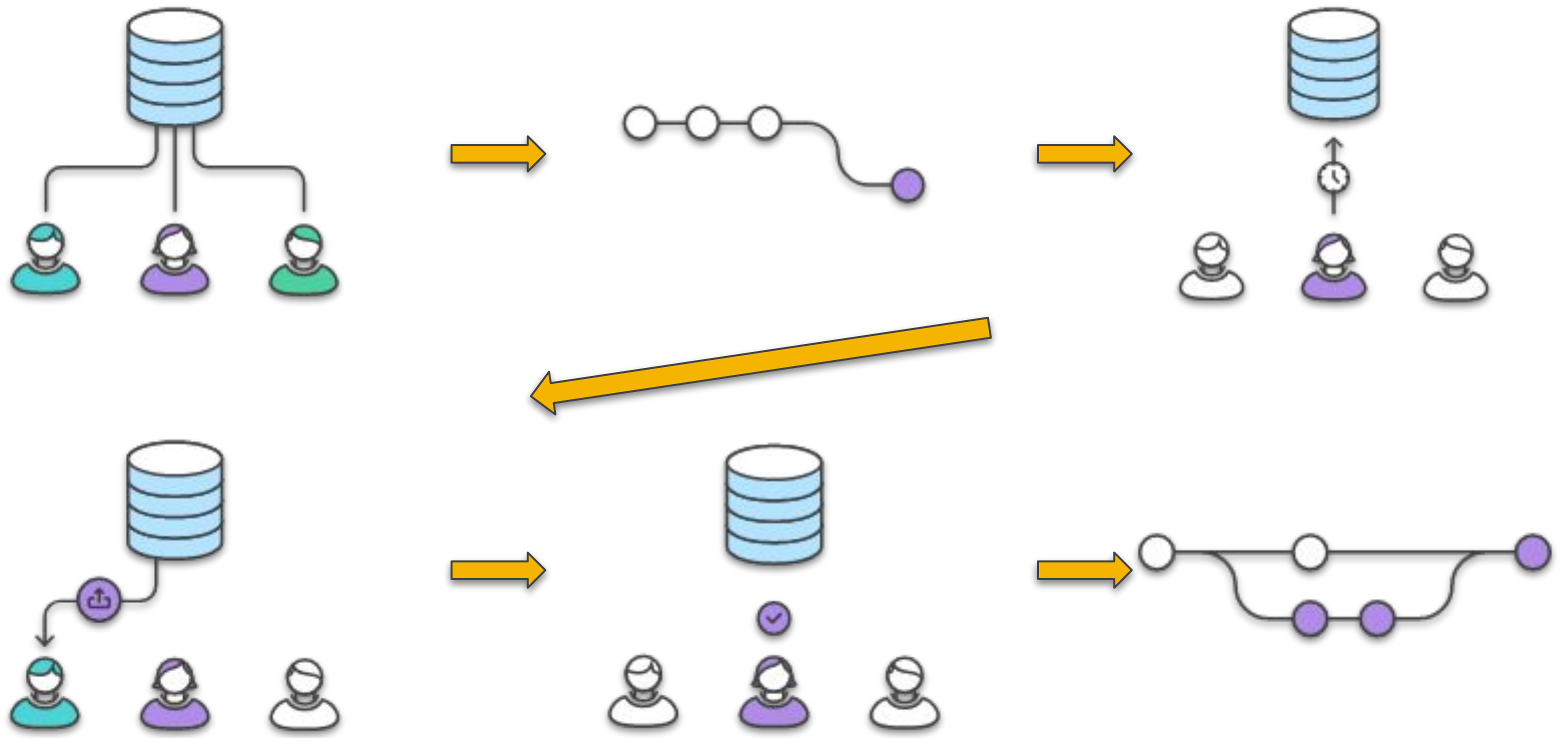
They give other developers the opportunity to **sign off** on a feature **before it gets integrated** into the official project.

Or, if you get stuck in the middle of a feature, you can **open a pull request asking for suggestions** from your colleagues. The point is, pull requests make it incredibly **easy for your team to comment on each other's work**.

- focused on branching patterns
- can be leveraged by other repo oriented workflows
- promotes collaboration with team members through pull requests and merge reviews

https://www.atlassian.com/git/tutorials/comparing-workflows

Real Collaboration with Feature Branches



https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow

Gitflow

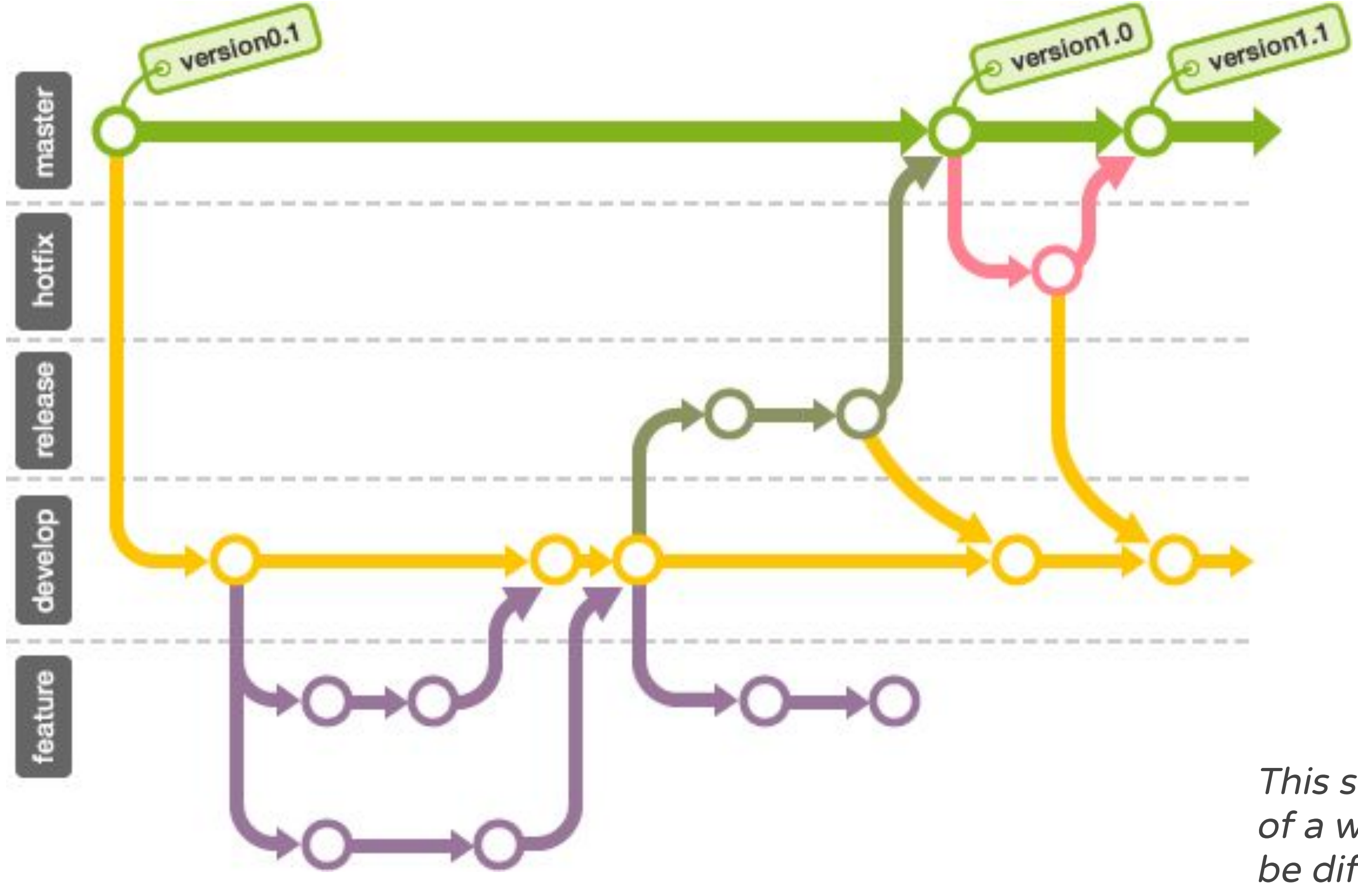
The individual Gitflow workflow dictates what **kind of branches to set up** and **how to merge them** together.

Some key takeaways to know about Gitflow are:

- The workflow is great for a **release-based software workflow**.
- Gitflow offers a **dedicated channel for hotfixes to production**.

Typical Branches for Gitflow: *master, develop, features-**, *hotfix, release*

Sample Workflow (Git Flow)



This schema is just one sample of a workflow, branches can be different in other projects.

Gitflow

The overall workflow of Gitflow:

- ➢ A **develop** branch is created from **main**
- ➢ A **release** branch is created from **develop**
- ➢ **Feature branches** are created from **develop**
- ➢ When a feature is **complete** it is **merged into the develop branch**
- ➢ When the **release branch is done** it is **merged into develop and main**
- ➢ If an issue in **main** is detected a **hotfix branch** is created from main
- ➢ Once the **hotfix is complete** it is merged to **both develop and main**

Gitflow: Branches & Branching

How to follow a Gitflow:

- ❖ By convention the main branch of a project is called main *(previously master)*. This branch **should always be fully functional.**
- ❖ Two kind of branches: branches that are intended to **always exist** (e.g. dev, release, prod ...) and temporary which are **going to disappear** (e.g. bugfix, temporary features, user-story ...)
- ❖ Branches are created for each **change to the code base.**
  - ➢ For any modification of the project, the **addition** of a **new feature** for example, a **new branch is created** that will be re-integrated later (merge or rebase) according to the workflow of the company.
- ❖ **No changes directly to main.** There should always be a (new) branch for the changes.

Gitflow Summary

- Based on **Feature Branch Workflow**
- Assigns very **specific roles to different branches** and defines **how and when they should interact**
- Feature Branch Workflow with **development** branch and individual branches for **preparing, maintaining, and recording releases**
- Supports **pull requests**, **isolated experiments**, and **efficient collaboration**

WILD
CODE
SCHOOL

GitHub

https://github.com

What is GitHub?

Version Management **Web** Service, created in 2008

- The biggest source code host
- Based on Git
- Free for public (open-source)/private and paid repositories for advanced features

Alternatives to GitHub

- GitLab
- Bitbucket
- ...

GitHub Functionalities

Real social network of developers

- ❖ Project follow-up
- ❖ Follow-up of people
- ❖ Team building

Services for projects

- ❖ Wiki / Project Page
- ❖ Problem (issue) Tracking

Integration of external services

- ❖ Integration with Authentication & Authorization Protocols (SSO)

Forking

GitHub encourages forking, that is to **copy a project** (open-source), to install it on one’s **own GitHub (or any other git provider) account** and to **modify** it according to one’s **needs**

A pull request (request for contribution) can **easily be made**, if necessary, to the **owner of the initial project** (to **enable back-merging** of changes).

WILD
CODE
SCHOOL

GitLab

https://gitlab.com/

What is a GitLab?

GitLab is a **DevOps software** that combines the ability to **develop, secure, and operate** software in a single application.

Initially, it was as a **source code management solution** to collaborate within a team on software development that evolved to an integrated solution covering the software development life cycle, and then to the **whole DevOps life cycle**

GitLab aims for uniting “development and operations in one user experience.”

Both, GitLab and GitHub are **web-based Git repositories**.

Differences to GitHub

- **Authentication Levels**

With GitLab, you can **set and modify people's permissions** according to their **roles**. In GitHub, you can decide if someone gets **read or write access to a repository**.

- **GitLab CI vs GitHub Actions**

One of the big differences between GitLab and GitHub is the **built-in Continuous Integration/Delivery of GitLab**.

GitLab has clearly been addressing the DevOps market earlier than its competitor as well as offering an operations dashboard that lets you understand the dependencies of your development and DevOps efforts.

- GitLab offers **mores features**, especially tailored for **onsite enterprise projects**.
- GitHub offers a workflow and **minimal feature set** which makes it **useable for huge projects** (OSS).

https://usersnap.com/blog/gitlab-github/

Differences to GitHub (Terminology)

- **Pull request** => **Merge request** and displayed with this icon.
- **Organisations** => **Group** as shown in
- **Gist** => **Snippets**

Forking Workflow

This means that **each contributor** has not one, but **two Git repositories**: a private local one and a public server-side one. The Forking Workflow is most often seen in public open source projects.

Developers **push to their own server-side repositories**, and only the project maintainer can push to the official repository. This allows the **maintainer to accept commits** from any developer without giving them write access to the official codebase.

Forked repositories are generally "server-side clones" and **usually managed and hosted by a 3rd party** Git service like GitHub, GitLab or Bitbucket.

Forking Workflow

Sample Forking Workflow for a Open Source project:

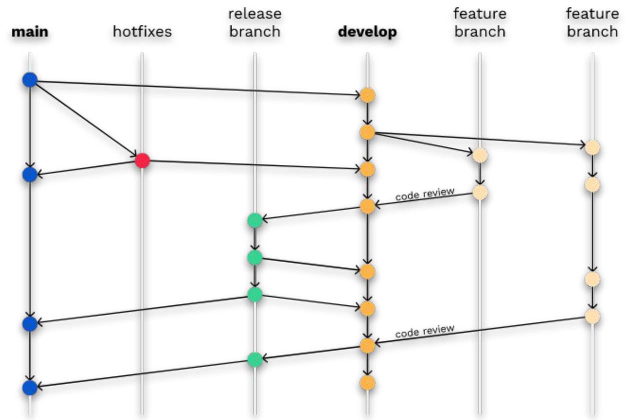
- You want to **contribute** to a code base **hosted at** github.com/userA/open-project
- Using GitHub you **create a fork of the repo** to github.com/YourName/open-project
- On your local system you execute git clone on https://github.com/YourName/open-project to get a **local copy** of the **remote repo**
- You create a **new feature branch** in your local repo
- Work is done to complete the new feature and git commit is executed to save the changes
- You then **push** the new feature branch to **your remote forked repo**
- Using GitHub you open up a **pull request for the new branch against the original repo** at github.com/userA/open-project

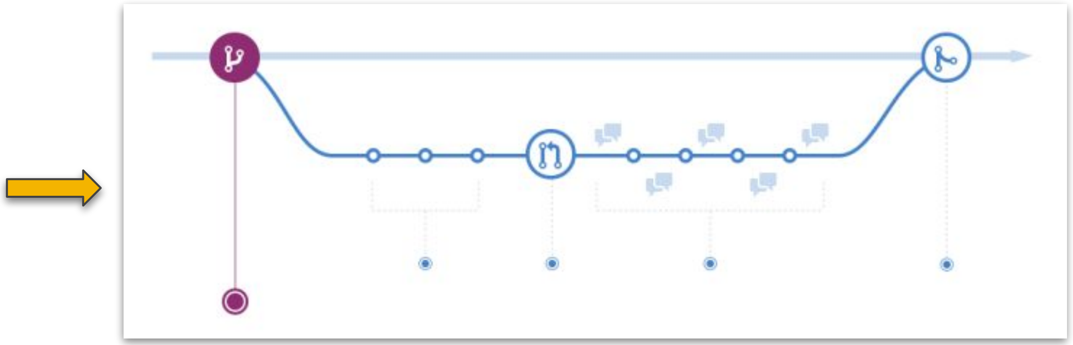
https://www.atlassian.com/git/tutorials/comparing-workflows

GitHub Flow

The GitHub Flow is a **lightweight**, **branch-based** workflow.

It is useful developers, but can be reused for other artifacts like **documentation, specifications, and other collaborative tasks**.





https://lucamezzalira.com/2014/03/10/git-flow-vs-github-flow/
https://reviewpad.com/blog/github-flow-trunk-based-development-and-code-reviews/

GitHub Flow

Features of the GitHub Flow:

- ❖ Anything in the **main** branch is **deployable**.
- ❖ Something new is **created in a descriptively named branch off of main** (ie:new-oauth2-scopes)
- ❖ **Commits** are **regularly pushed** to the server
- ❖ For feedback or help, or if the work is finished, **a pull request is opened**
- ❖ After **review** and **sign-off** on the feature, it can be **merged into main**
- ❖ Once it is **merged and pushed** to main, it should be **deployed immediately**

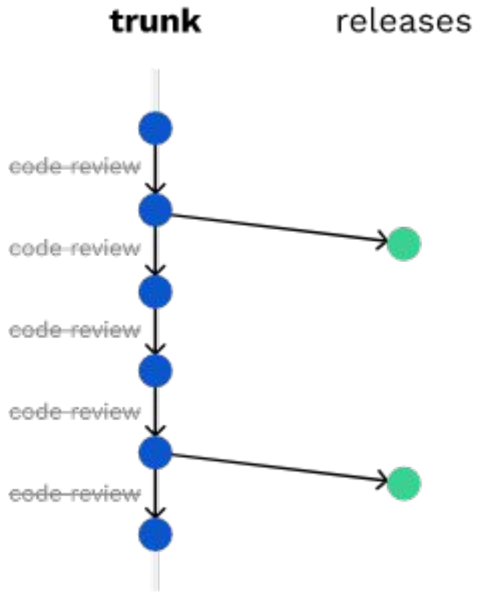
https://docs.github.com/en/get-started/quickstart/github-flow

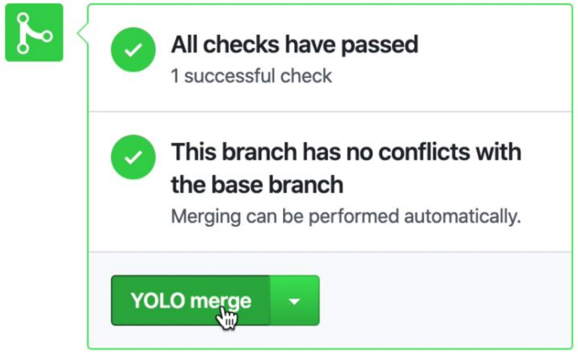
WILD
CODE
SCHOOL

Progressive Git Workflows

Trunk-based Development

Trunk is deployable at all times. Changes should be **summited daily**. Unfinished features should be unexposed with the help of **feature flags** (YOLO Software development).





All checks have passed

1 successful check

This branch has no conflicts with the base branch

Merging can be performed automatically.

https://reviewpad.com/blog/github-flow-trunk-based-development-and-code-reviews/

Trunk-based Development Best Practices

- ❖ Develop in **small batches**
- ❖ Use **Feature flags**
- ❖ Implement comprehensive **automated testing**
- ❖ Perform **asynchronous code reviews**
- ❖ Have **three or fewer active branches** in the application's code repository
- ❖ **Merge** branches to the trunk **at least once a day**
- ❖ **Build fast** and **execute immediately**

Trunk-based development is currently the standard for **high-performing engineering teams** since it sets and **maintains** a **software release cadence** by using a **simplified** Git **branching strategy**.

Plus, trunk-based development gives engineering teams **more flexibility and control** over how they **deliver software** to the end user.

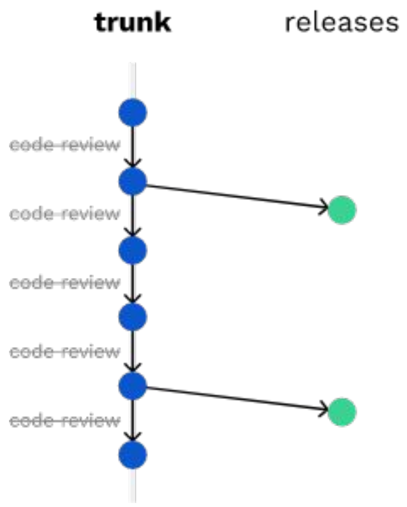
But it has to be well prepared and checked against the project requirements.

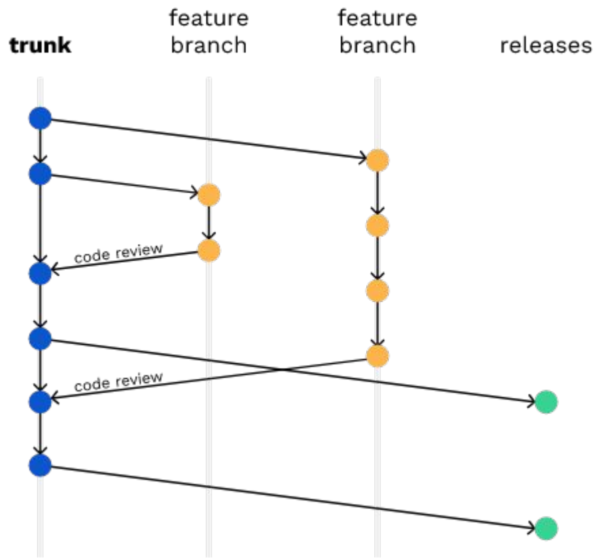
https://reviewpad.com/blog/github-flow-trunk-based-development-and-code-reviews/

Scaled Trunk-based Development (STBD)

STBD branching model uses **short-lived feature branches** with a life span of a few days (maximum) before merging to the **trunk, which is deployable at all times**.

Minimal work in progress to **avoid merging problems** and to facilitate **easier and faster code reviews**. STBD explicitly suggests code reviews.



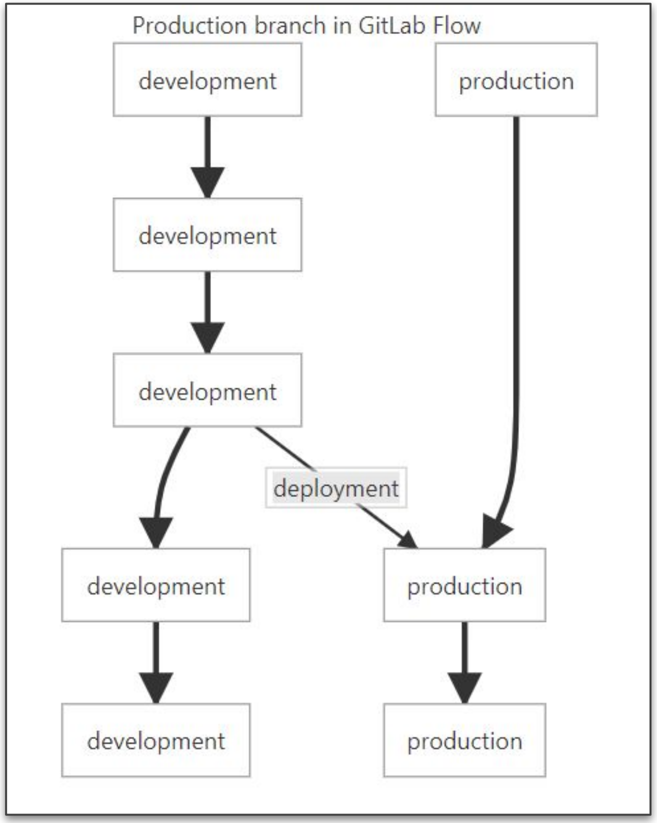


GitLab Flow

GitHub flow assumes you can **deploy to production** every time you **merge a feature branch**. While this is possible in some cases, such as SaaS applications, there are some cases where this is not possible, such as:

- ❖ You don't control the timing of a release. For example, an iOS application that is **released when it passes App Store validation**.
- ❖ You have **deployment windows** - for example, workdays from 10 AM to 4 PM when the operations team is at full capacity - but you also merge code at other times.

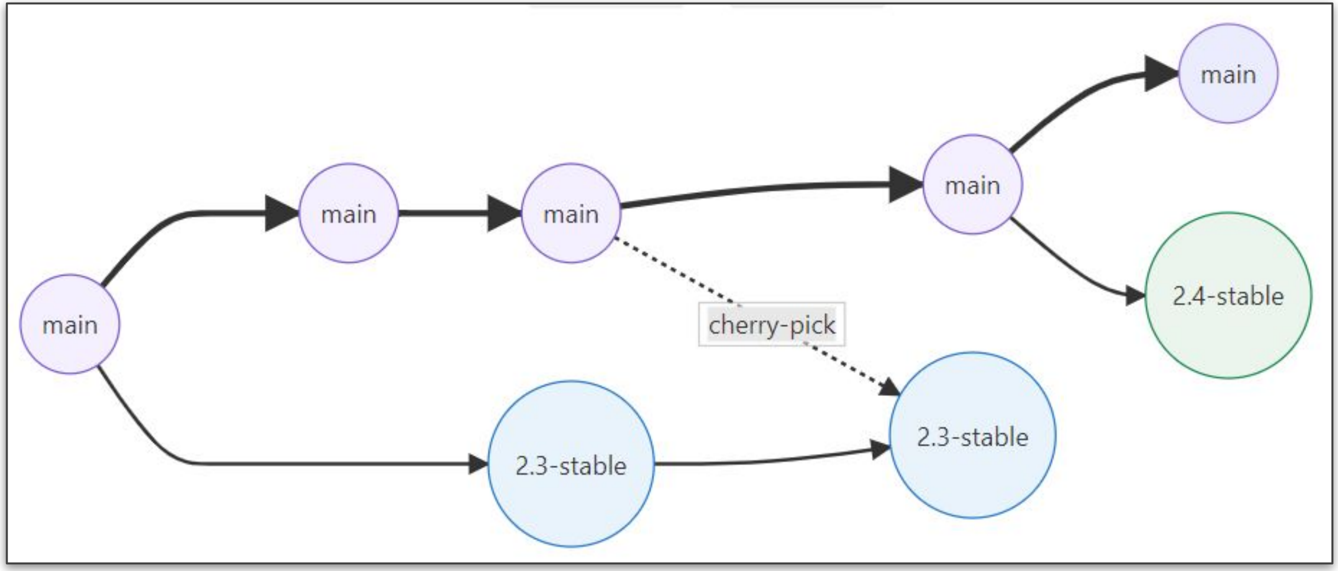
In these cases, you can make a **production branch that reflects the deployed code**. You can deploy a new version by **merging development into the production branch**.



https://docs.gitlab.com/ee/topics/gitlab_flow.html

GitLab Flow

Git allows a **wide variety** of **branching** strategies and **workflows**. Because of this, many organizations end up with **workflows that are too complicated**, not clearly defined, or not integrated with issue tracking systems. GitLab combines **feature-driven development** and **feature branches** with **issue tracking**.



https://docs.gitlab.com/ee/topics/gitlab_flow.html

Simplified Git Flow

“**GitFlow** was a bit **heavyweight** for us. We practice continuous deployment and release multiple times a day, which doesn’t lend itself to **the release-based workflow**.”

“**Github Flow** didn’t really address situations like **multiple engineers working on a single feature**.”

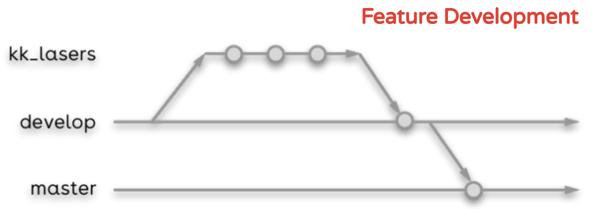
https://medium.com/goodtogoat/simplified-git-flow-5dc37ba76ea8

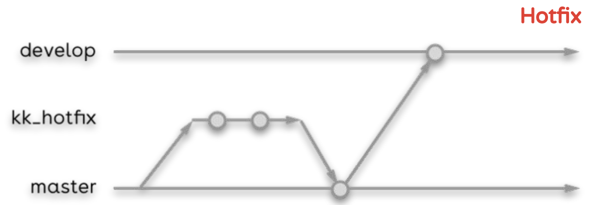
Simplified Git Flow

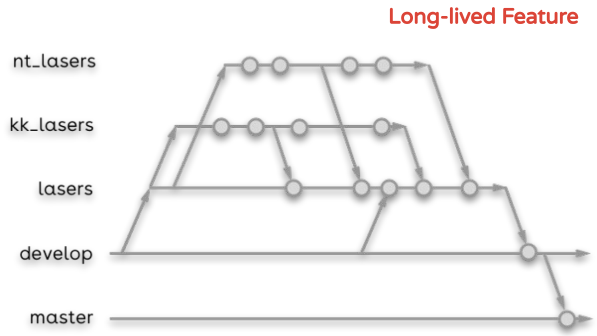
- **Avoids large, high-risk deploys**
- **Avoids constant, time-consuming deploys**

Might **not be appropriate** for your team if your deploy process is **heavyweight enough that you can't deploy every day**.

It also might **not scale** without modification to a team **larger than a few dozen devs**.







https://medium.com/goodtogoat/simplified-git-flow-5dc37ba76ea8

Criteria for Git Workflow Selection

Criteria for Git Workflow Selection (2nd Part)

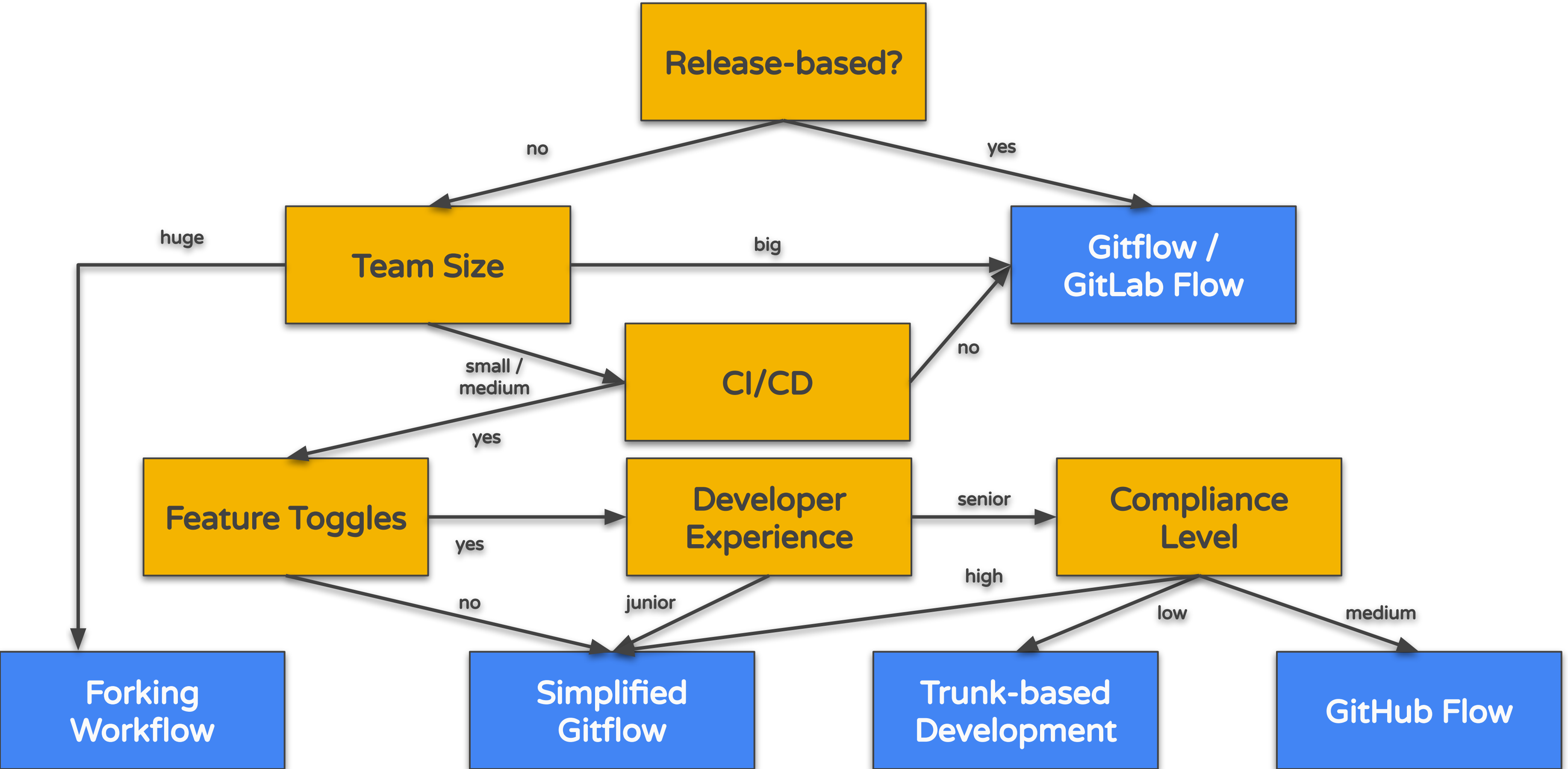
Questions to ask the team:

- ❖ **What problems will this Git workflow help us solve?**
- ❖ **What problems will it create?**
- ❖ **What sorts of development will this model encourage, does it fit to our model?**
- ❖ **Are we happy with the current development model?**

Any branching model you choose is ultimately meant to **make humans work together more easily to produce software**, and so the branching model needs to take into account the **needs of the particular humans using it**, not something someone wrote on the internet and claimed was ‘successful’.

https://georgestocker.com/2020/03/04/please-stop-recommending-git-flow/
https://www.freshconsulting.com/insights/blog/git-development-workflows-git-flow-vs-github-flow/

Decision Tree for Git Workflow Selection



Links and other information

About Git

- **Become a Git Guru:** https://www.atlassian.com/git/tutorials (incl. Git Workflows)

About Git Workflows

- **Interactive GitFlow:** http://danielkummer.github.io/git-flow-cheatsheet/
- **GitHub Flow:** https://docs.github.com/en/get-started/quickstart/github-flow
- **GitLab Flow:** https://docs.gitlab.com/ee/topics/gitlab_flow.html
- **Stop using Git Flow:** https://georgestocker.com/2020/03/04/please-stop-recommending-git-flow/
- **Simplified Git Flow:** https://gist.github.com/vxhviet/9c4a522921ad857406033c4125f343a5

Other Links

- **How-To write good Commit Messages:** https://cbea.ms/git-commit/
- **Use GitLab with GitHub Desktop:** https://itnext.io/how-to-use-github-desktop-with-gitlab-cd4d2de3d104