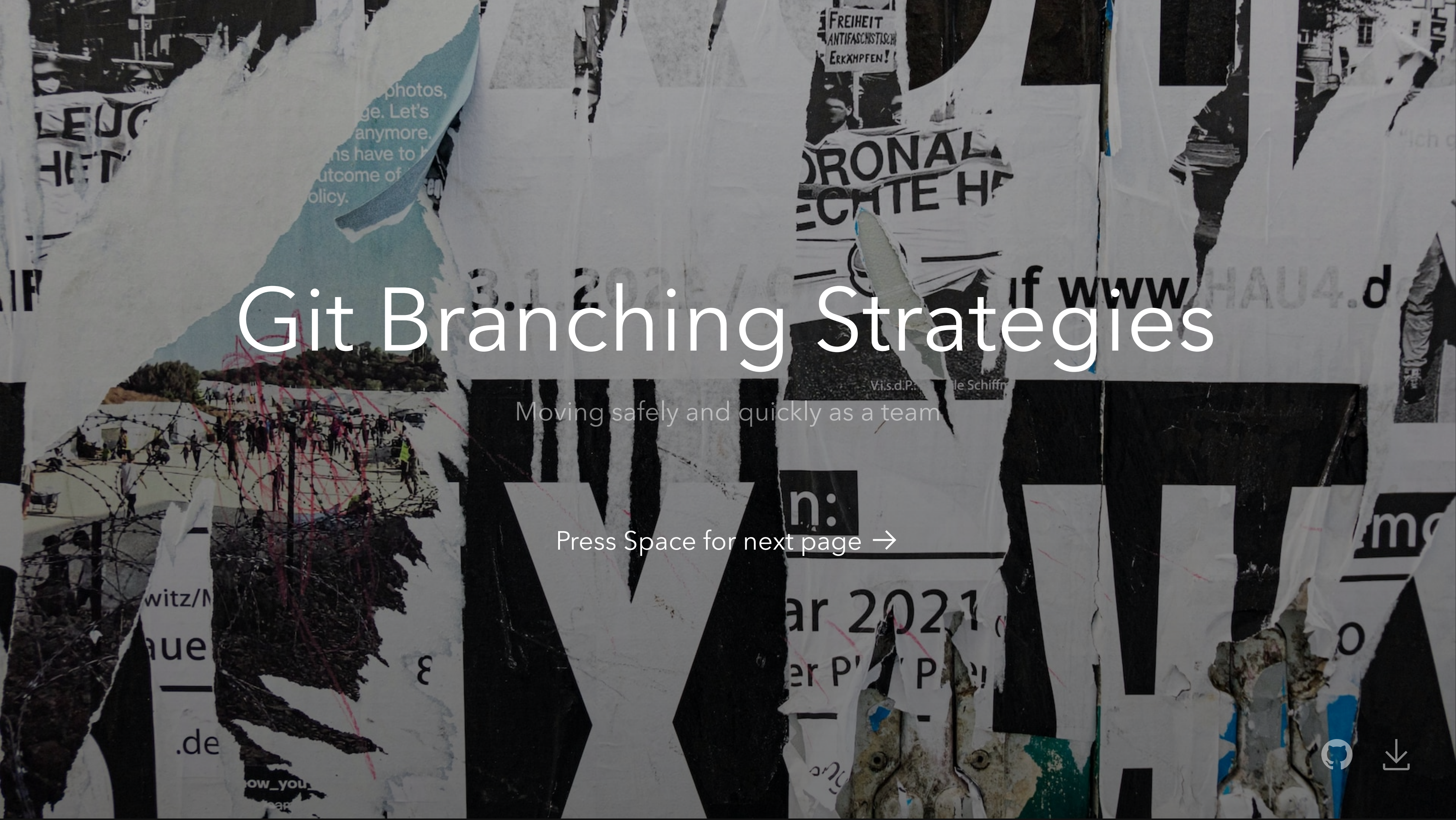


Git Branching Strategies

Moving safely and quickly as a team

Press Space for next page →

Kevin Chen

Software Engineer at Teaching Strategies.

Formerly at The Drivers Coop, Care.com, and IrisVR.

Fan of chess, racket sports, skiing, guitar.

kevinmichaelchen

kevinmchen

kevin.chen@teachingstrategies.com

Table of Content

What is a Git branch?

- 🖖 Branches allow developers to **diverge** from the main branch by creating separate branches to isolate code changes
- 👉 A branch is essentially a reference or a **pointer** to the latest commit in a given context; it’s not a container for commits.
- 🪶 The Git branching model is **lightweight** compared to other version control systems.
- ⛓️ Branches allow for **independent** lines of code that branch off the master branch, allowing developers to work independently before merging their changes back to the code base.

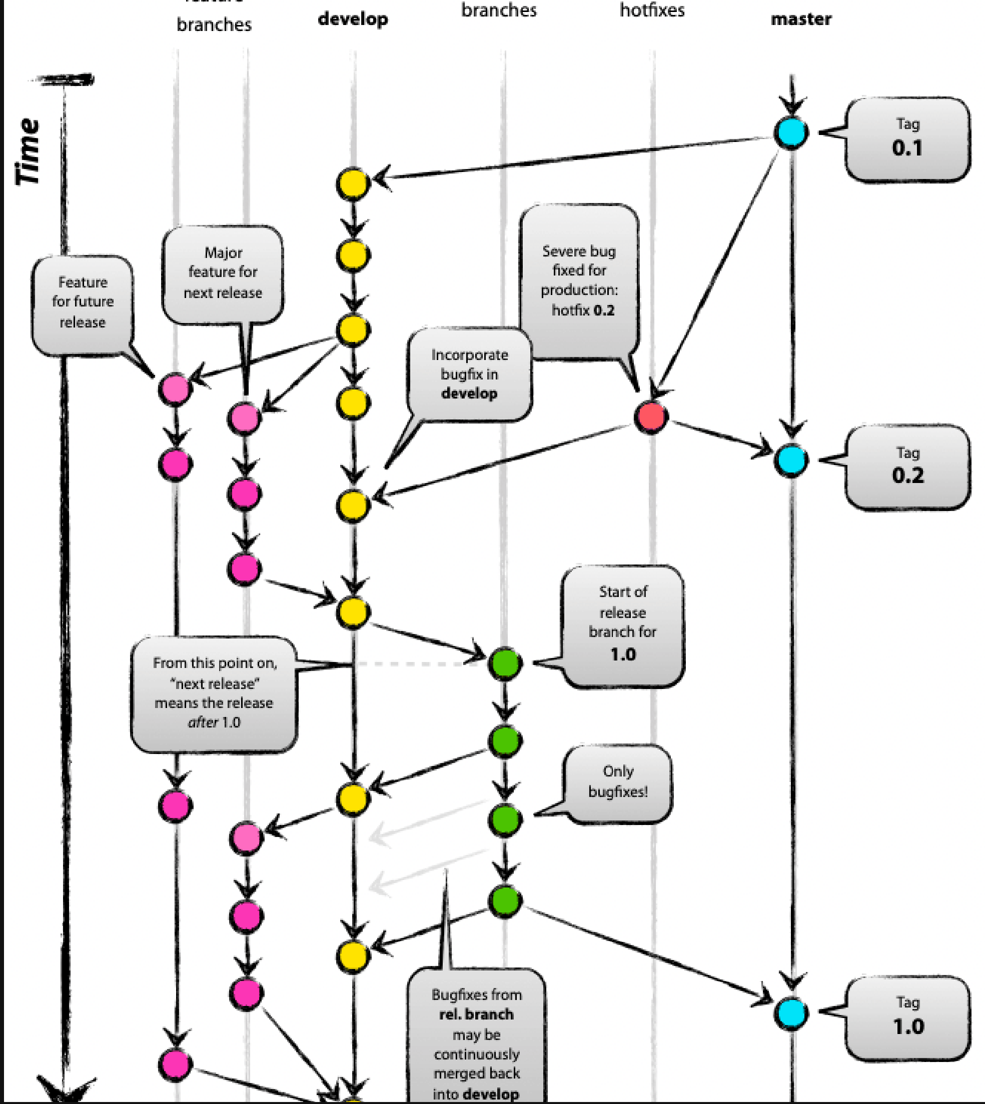
What is a Git branching strategy?

Branching strategies help to

- 🧠 Enhance productivity by ensuring proper **coordination among developers**
- ⛓️ Enable **parallel development**
- 🏗️ Help organize a series of **planned, structured releases**
- 🛣️ Map a clear **path** when making changes to software through to production
- 🐛 Maintain a **bug-free** code where developers can quickly fix issues and get these changes back to production without disrupting the development workflow

GitFlow

Created by Vincent Driessen:[1]

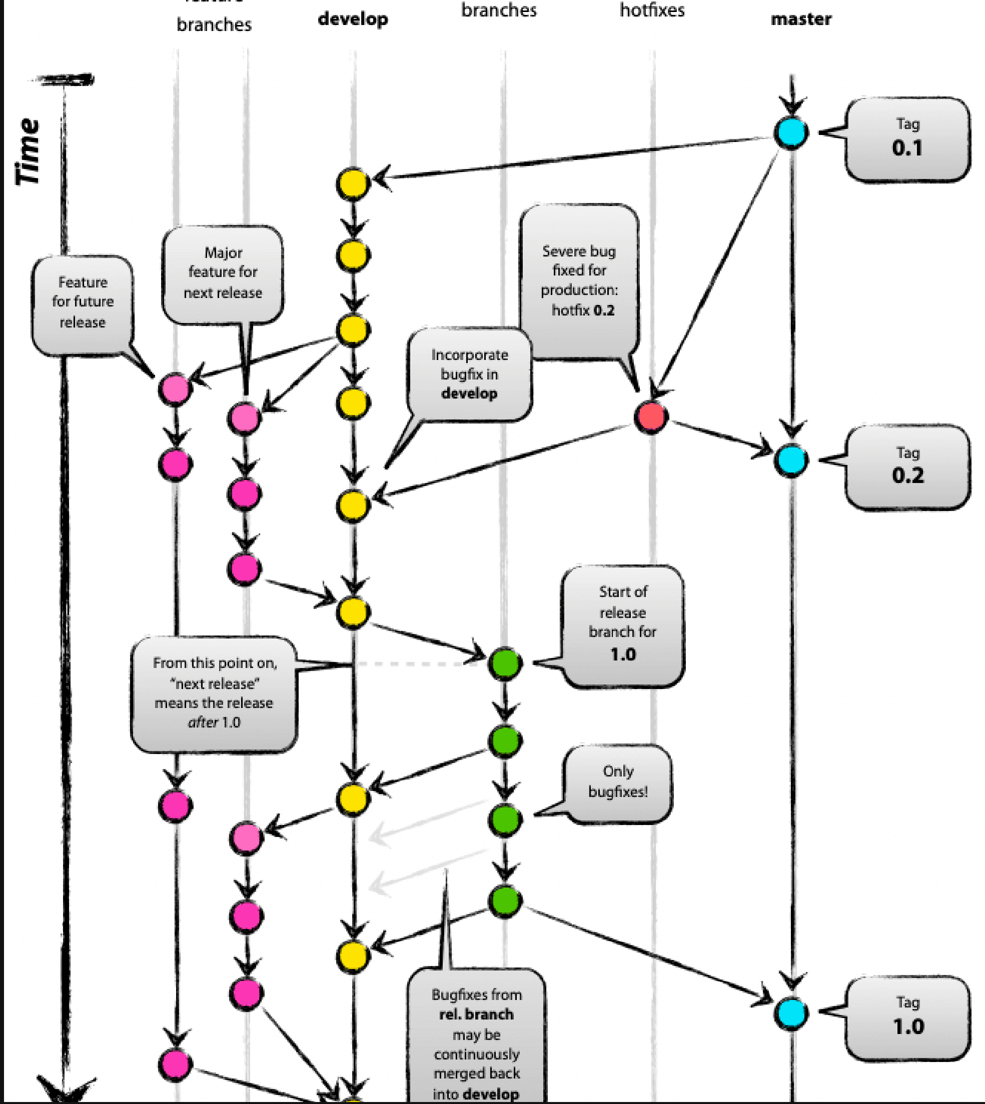


branches
develop
branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Tag 1.0
Bugfixes from rel. branch may be continuously merged back into develop

GitFlow

Created by Vincent Driessen:[1]

There are 2 immortal branches:

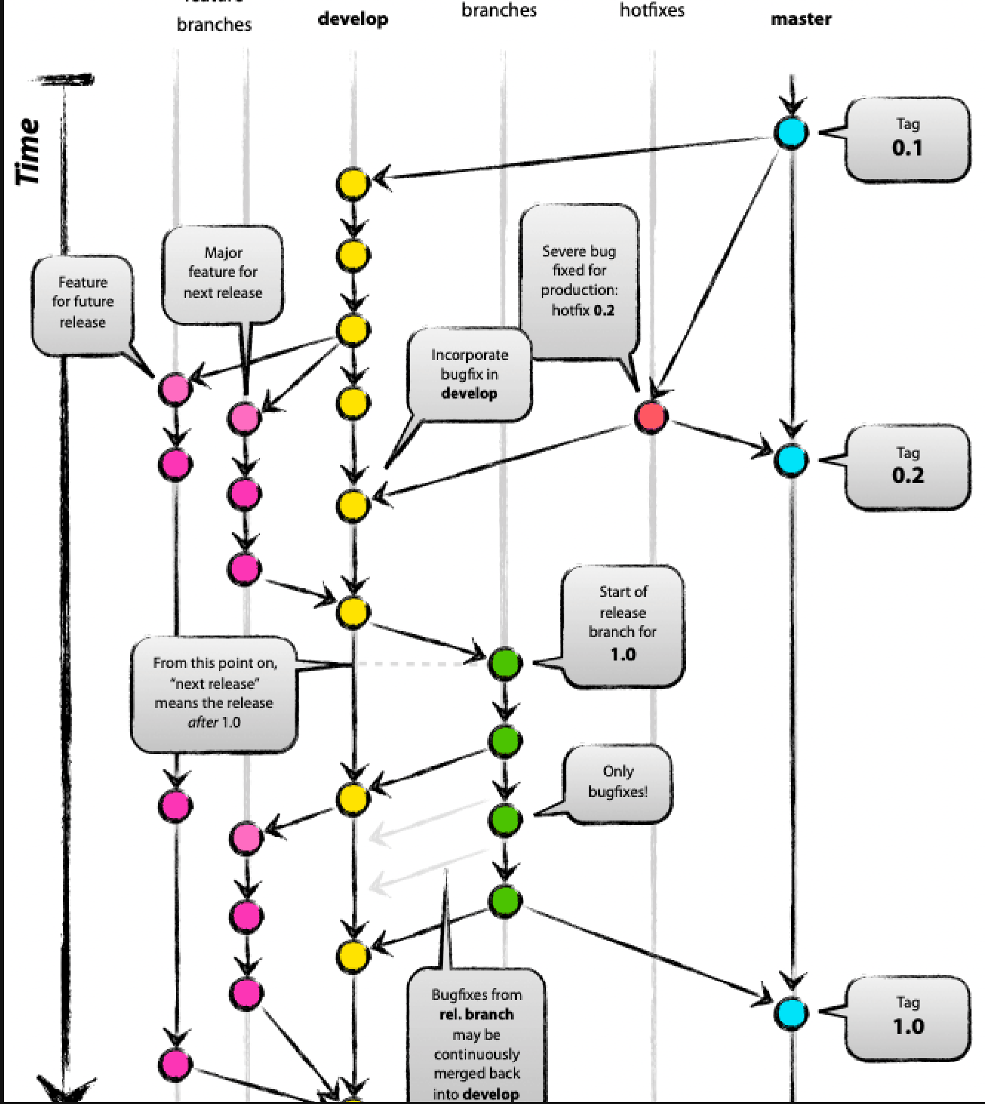

branches
develop
branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

GitFlow

Created by Vincent Driessen:[1]

There are 2 immortal branches:

- `master` branch is always production-ready.



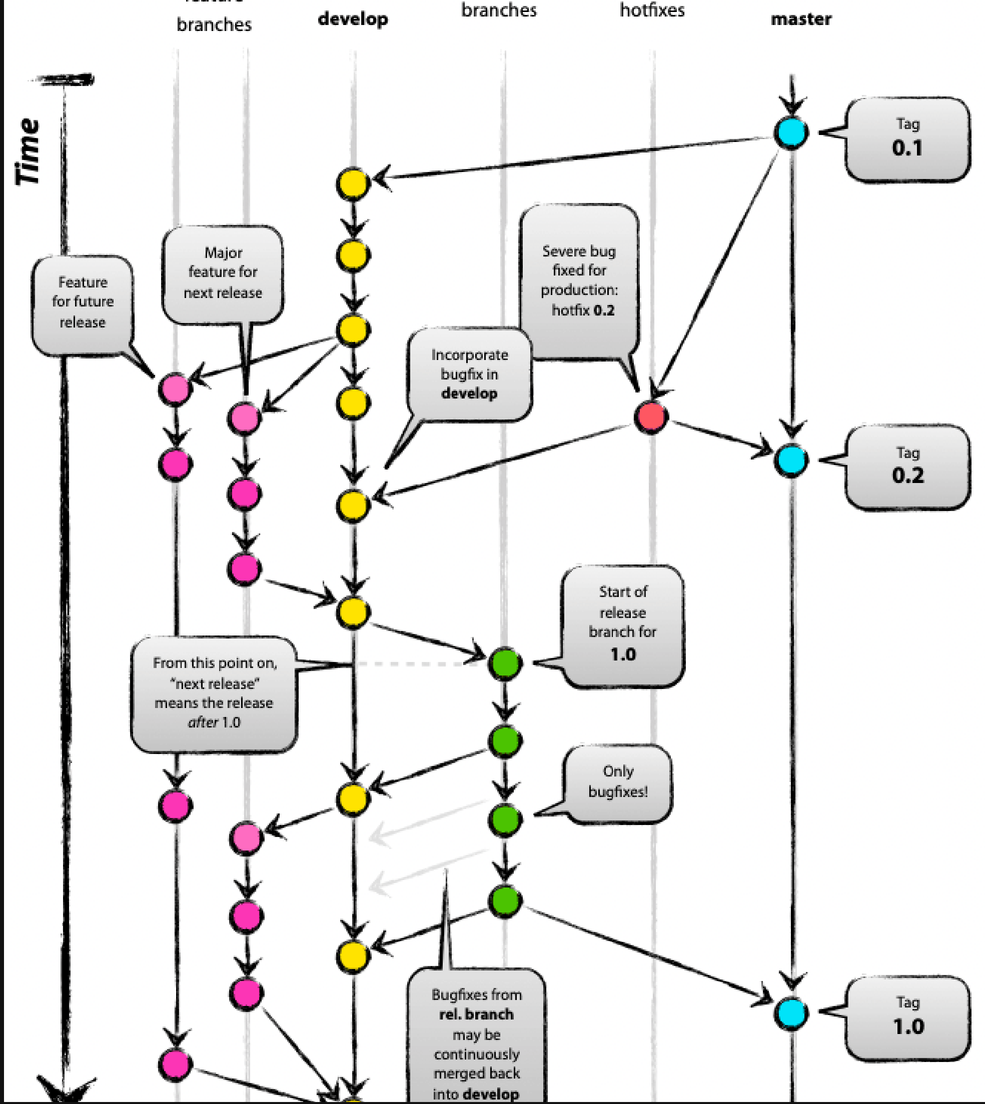
branches
develop
branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

GitFlow

Created by Vincent Driessen:[1]

There are 2 immortal branches:

- `master` branch is always production-ready.
- `develop` always has the latest delivered dev changes.


branches
develop
branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
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
Tag 1.0

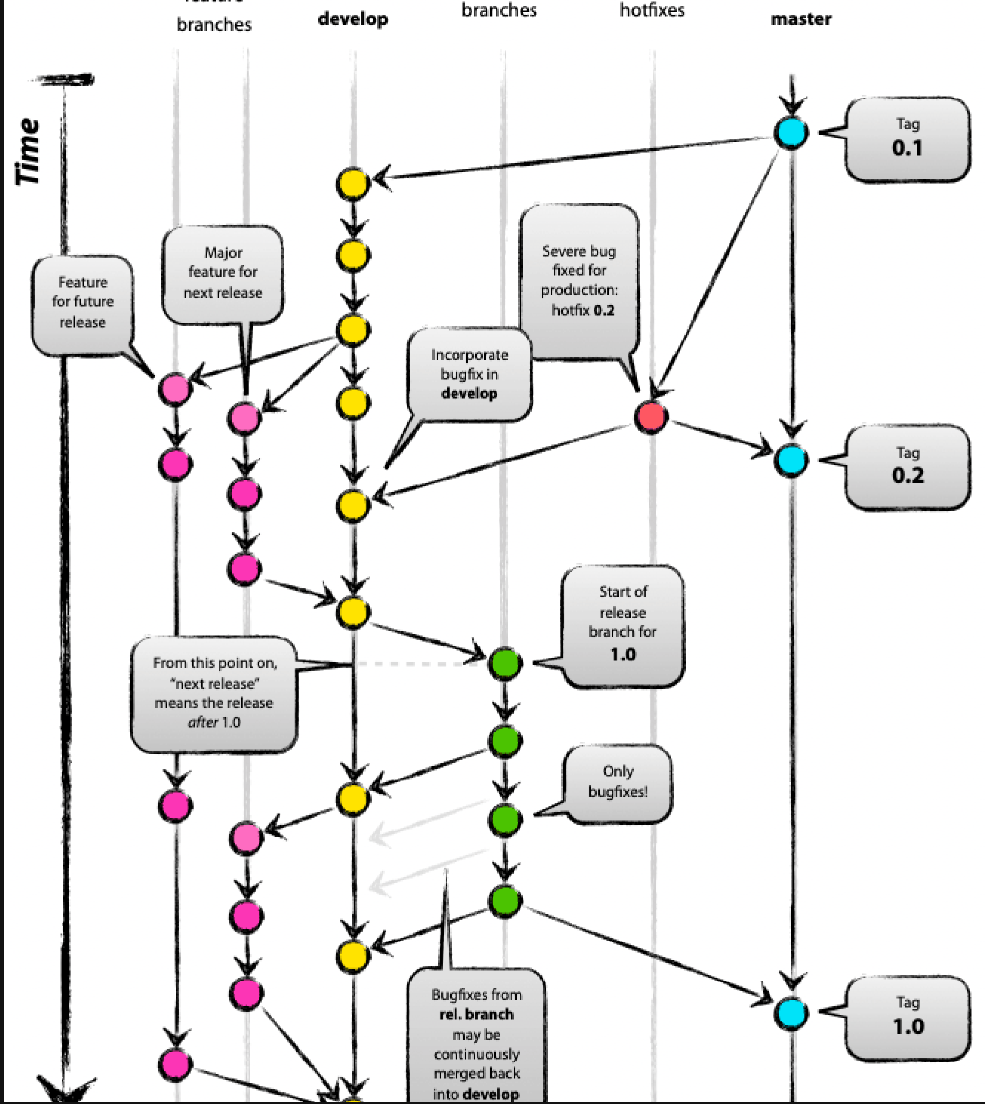
GitFlow

Created by Vincent Driessen:[1]

There are 2 immortal branches:

- `master` branch is always production-ready.
- `develop` always has the latest delivered dev changes.

There are 3 types of supporting branches:


branches
develop
branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
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
Tag 1.0

GitFlow

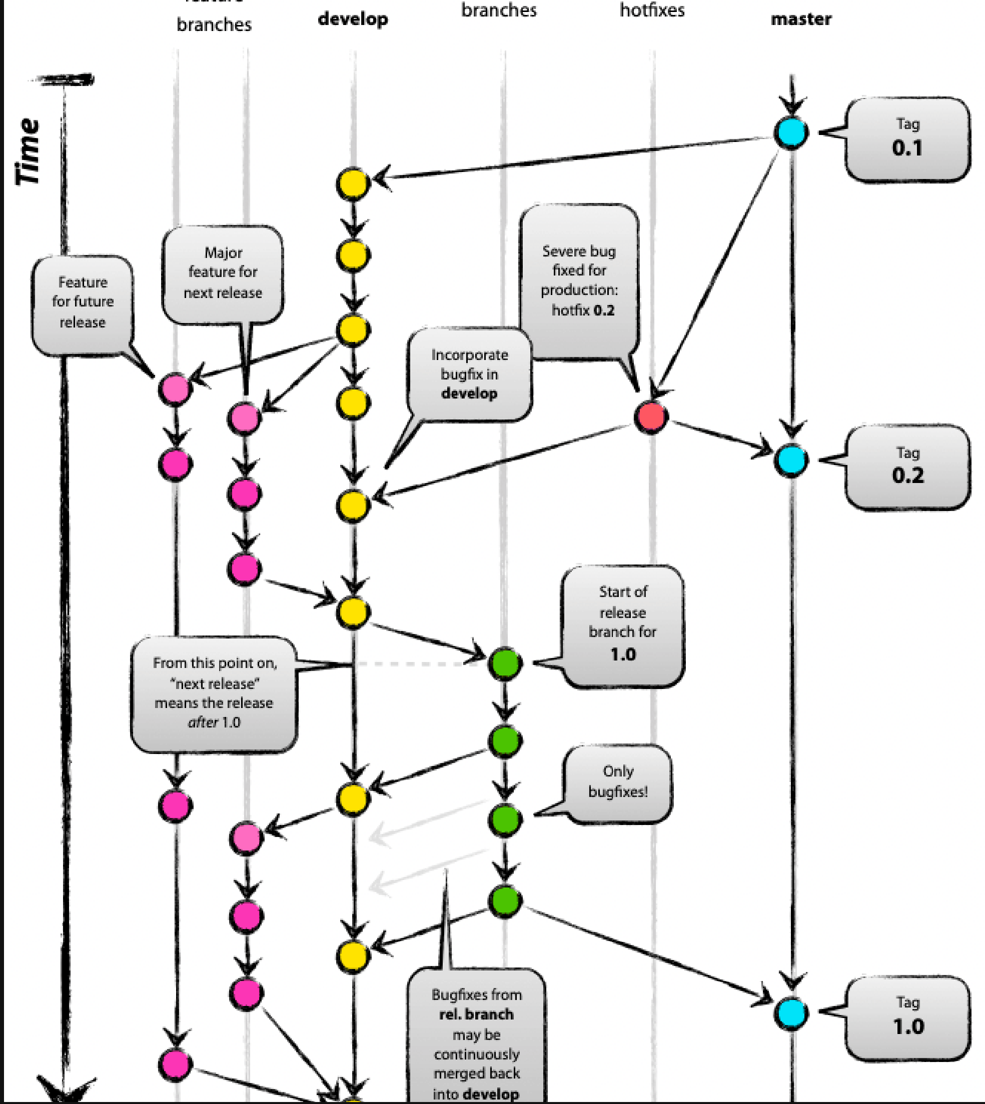
Created by Vincent Driessen:[1]

There are 2 immortal branches:

- `master` branch is always production-ready.
- `develop` always has the latest delivered dev changes.

There are 3 types of supporting branches:

- **feature** – both branched off and merged back into `develop`; typically only exist on localhost, not in the remote.


branches
develop
branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Tag 1.0
Bugfixes from rel. branch may be continuously merged back into develop

GitFlow

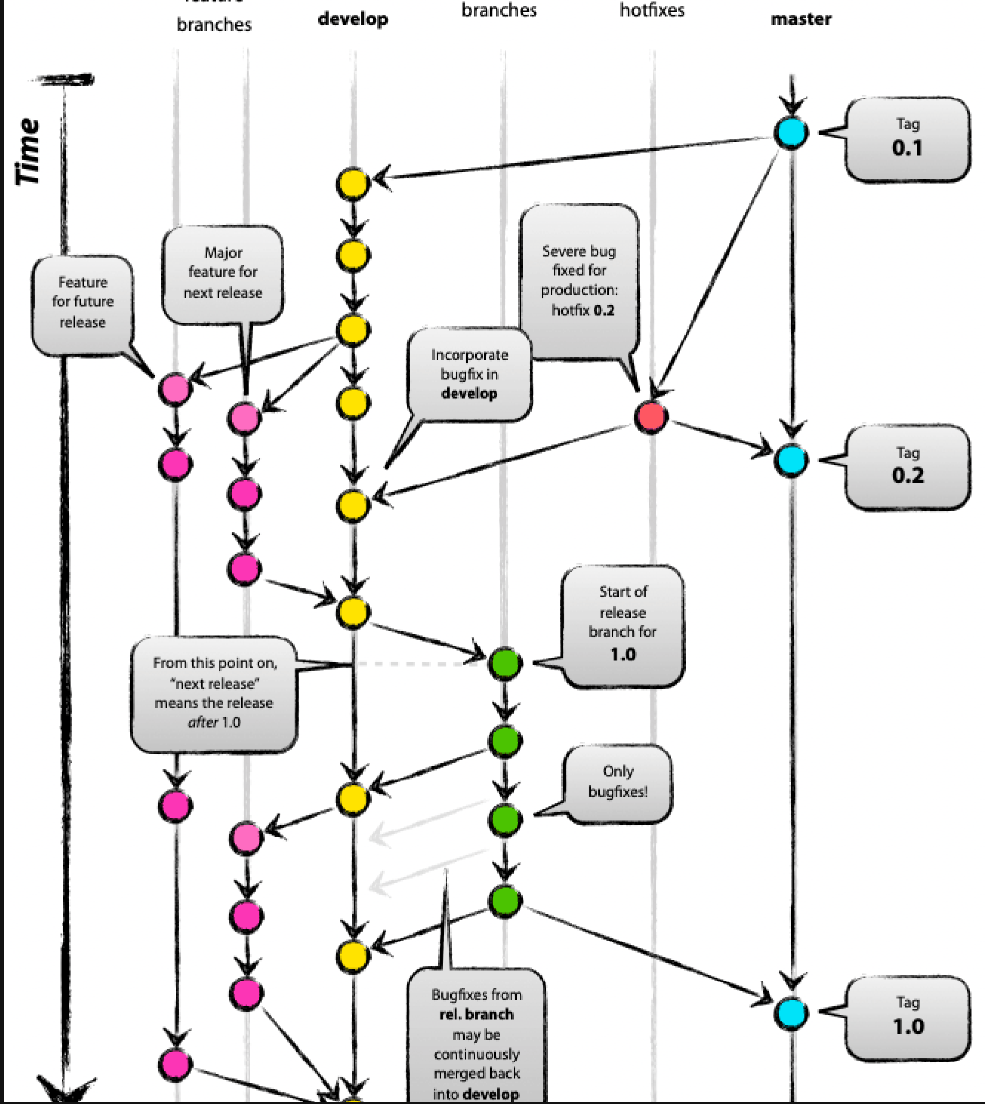
Created by Vincent Driessen:[1]

There are 2 immortal branches:

- `master` branch is always production-ready.
- `develop` always has the latest delivered dev changes.

There are 3 types of supporting branches:

- **feature** – both branched off and merged back into `develop`; typically only exist on localhost, not in the remote.
- **hotfixes** – branched off `master` and merged back into both `developer` and `master`


branches
develop
branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
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
Tag 1.0

GitFlow

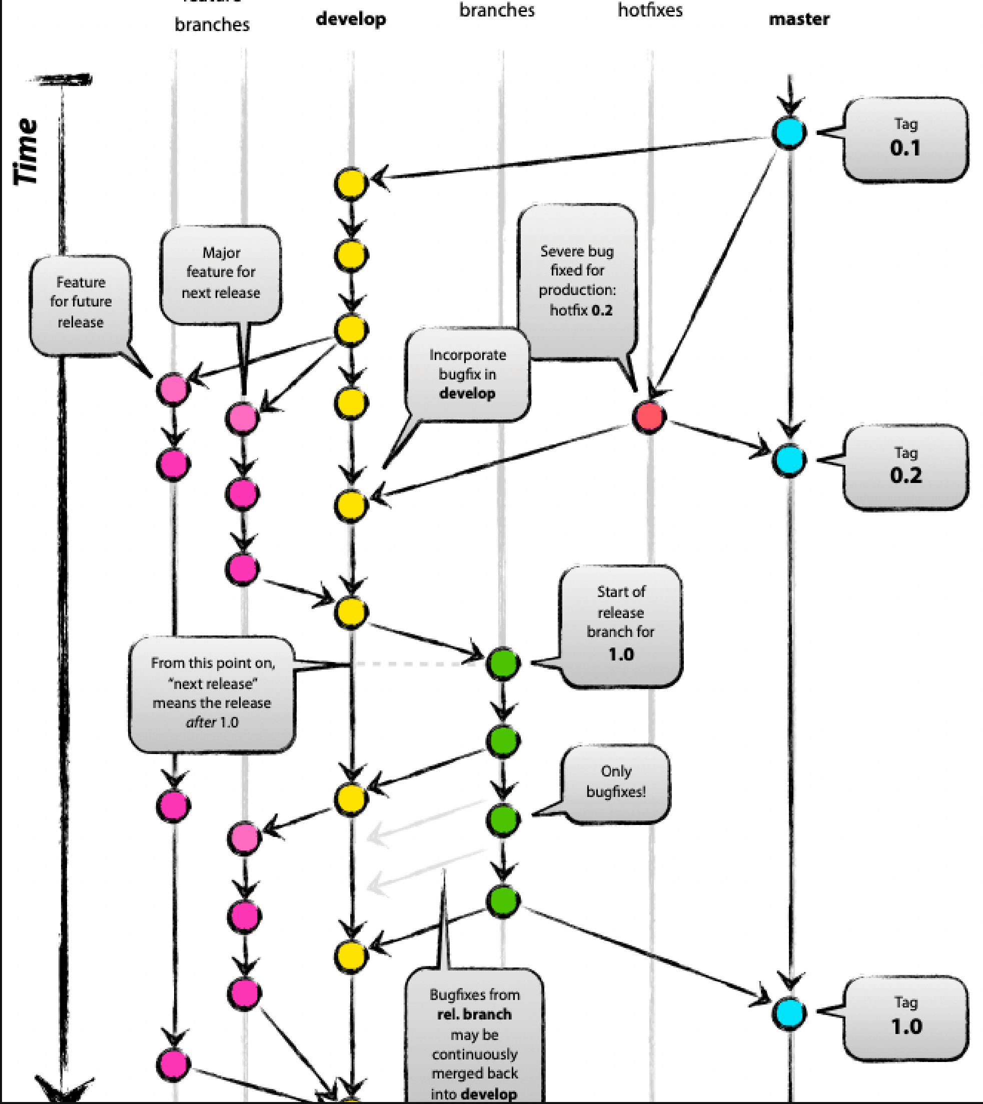
Created by Vincent Driessen:[1]

There are 2 immortal branches:

- `master` branch is always production-ready.
- `develop` always has the latest delivered dev changes.

There are 3 types of supporting branches:

- **feature** – both branched off and merged back into `develop`; typically only exist on localhost, not in the remote.
- **hotfixes** – branched off `master` and merged back into both `developer` and `master`
- **releases** – branched off `develop` and merged back into both `developer` and `master`



branches
develop
branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Tag 1.0
Bugfixes from rel. branch may be continuously merged back into develop

GitFlow

Created by Vincent Driessen:[1]

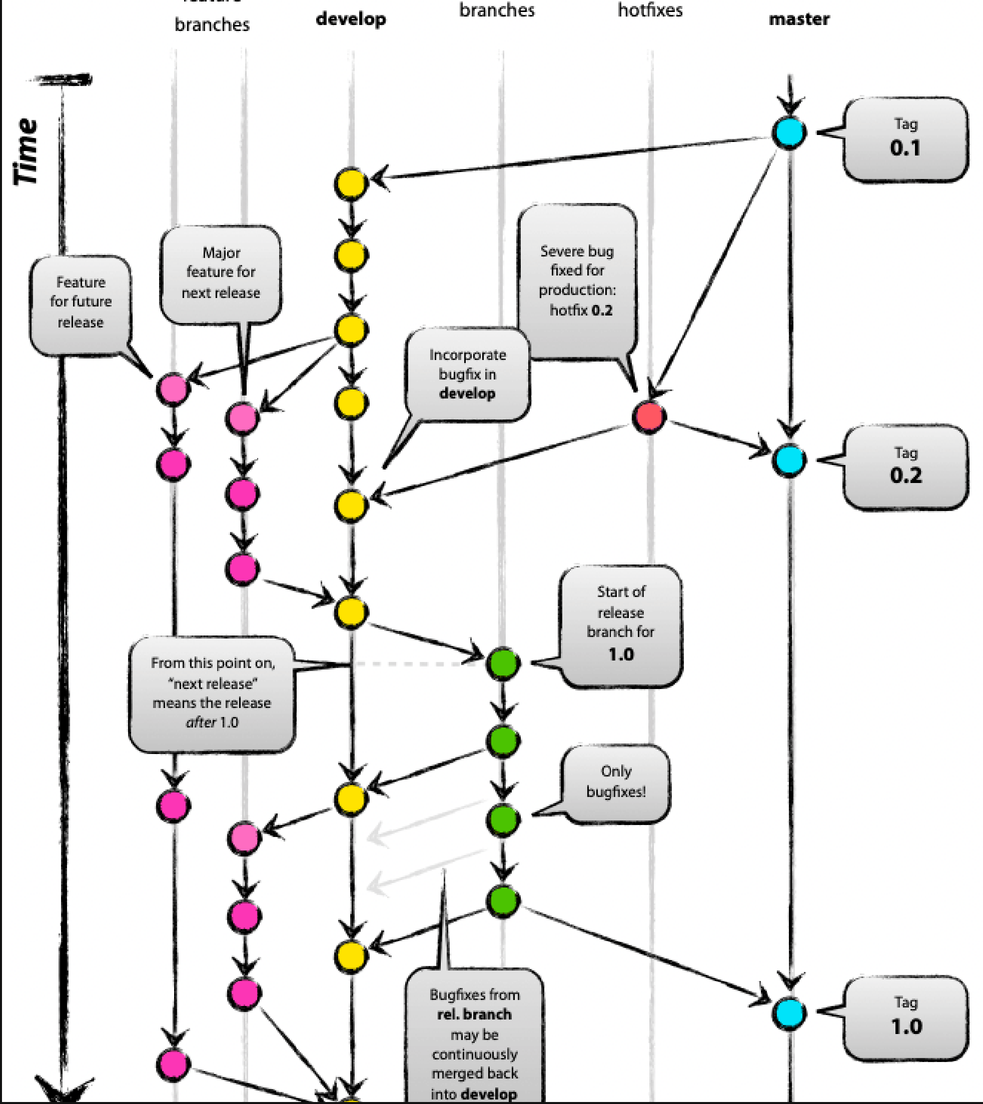
There are 2 immortal branches:

- `master` branch is always production-ready.
- `develop` always has the latest delivered dev changes.

There are 3 types of supporting branches:

- **feature** – both branched off and merged back into `develop`; typically only exist on localhost, not in the remote.
- **hotfixes** – branched off `master` and merged back into both `developer` and `master`
- **releases** – branched off `develop` and merged back into both `developer` and `master`

When `develop` basically resembles a new release, that's when the release branch is created.


branches
develop
branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Tag 1.0
Bugfixes from rel. branch may be continuously merged back into develop

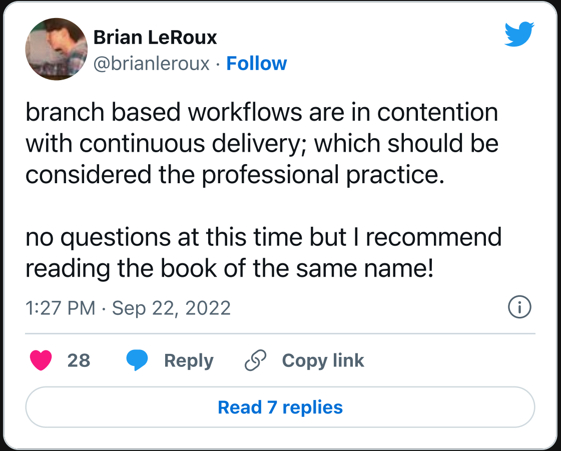
Downsides of GitFlow

A note from Gitflow's creator:

> Web apps are typically continuously delivered, not rolled back, and you don't have to support multiple versions of the software running in the wild...

> If your team is doing continuous delivery of software, I would suggest to adopt a much simpler workflow (like GitHub flow) instead of trying to shoehorn git-flow into your team...

> If, however, you are building software that is explicitly versioned, or if you need to support multiple versions of your software in the wild, then git-flow may still be as good of a fit to your team as it has been to people in the last 10 years.


Brian LeRoux
@brianleroux · Follow

branch based workflows are in contention with continuous delivery; which should be considered the professional practice.

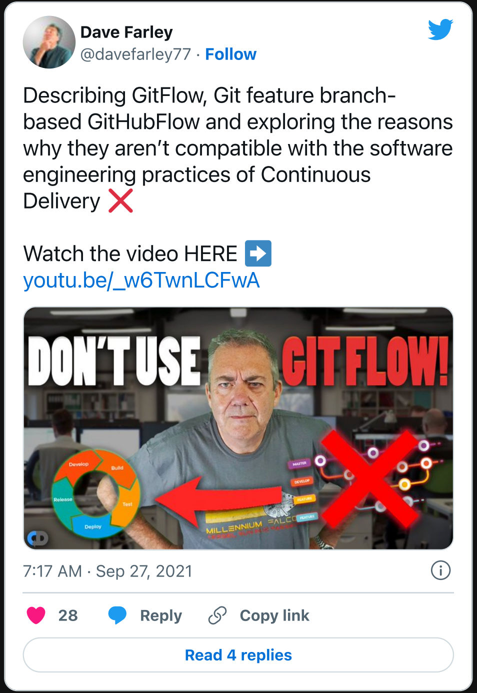
no questions at this time but I recommend reading the book of the same name!

1:27 PM · Sep 22, 2022

28 Reply Copy link

Read 7 replies

If not GitFlow, then what?


Dave Farley
@davefarley77 · Follow

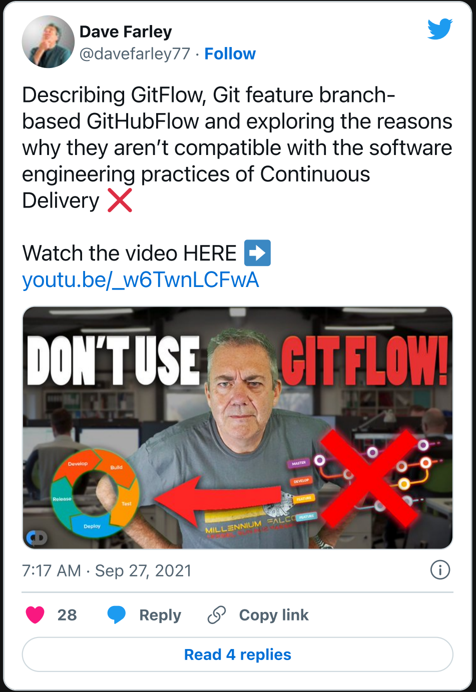
Describing GitFlow, Git feature branch-based GitHubFlow and exploring the reasons why they aren't compatible with the software engineering practices of Continuous Delivery ❌

Watch the video HERE ➡️ youtu.be/_w6TwnLCFwA

7:17 AM · Sep 27, 2021

If not GitFlow, then what?

Constant, continuous feedback is ideal. We can't afford to wait too long to check if my changes work with yours.



Dave Farley
@davefarley77 · Follow

Describing GitFlow, Git feature branch-based GitHubFlow and exploring the reasons why they aren't compatible with the software engineering practices of Continuous Delivery ❌

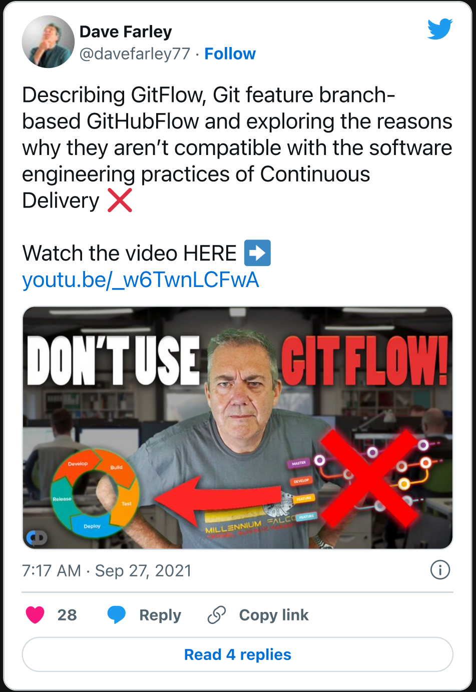
Watch the video HERE ➡️
youtu.be/_w6TwnLCFwA

7:17 AM · Sep 27, 2021

If not GitFlow, then what?

Constant, continuous feedback is ideal. We can't afford to wait too long to check if my changes work with yours.

Branches, by definition, are designed to isolate and hide change. Continuous integration, by definition, is designed to expose change.



Dave Farley
@davefarley77 · Follow

Describing GitFlow, Git feature branch-based GitHubFlow and exploring the reasons why they aren't compatible with the software engineering practices of Continuous Delivery ❌

Watch the video HERE ➡️
youtu.be/_w6TwnLCFwA

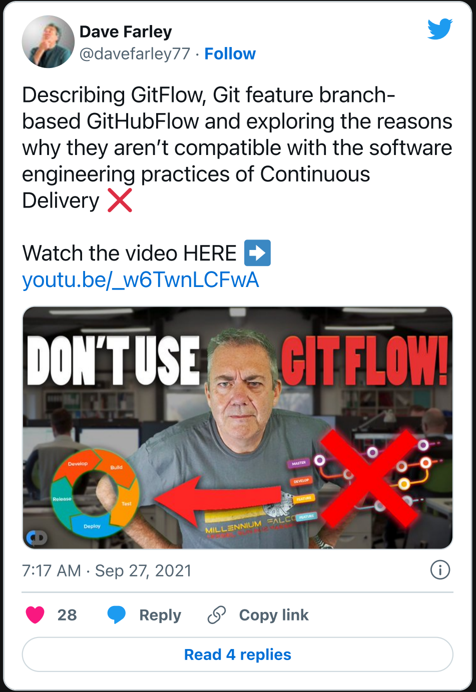
7:17 AM · Sep 27, 2021

If not GitFlow, then what?

Constant, continuous feedback is ideal. We can't afford to wait too long to check if my changes work with yours.

Branches, by definition, are designed to isolate and hide change. Continuous integration, by definition, is designed to expose change.

GitHub Flow[1] (aka plain old *feature branching*) is a good alternative.



Dave Farley
@davefarley77 · Follow

Describing GitFlow, Git feature branch-based GitHubFlow and exploring the reasons why they aren't compatible with the software engineering practices of Continuous Delivery ❌

Watch the video HERE ➡️
youtu.be/_w6TwnLCFwA

7:17 AM · Sep 27, 2021

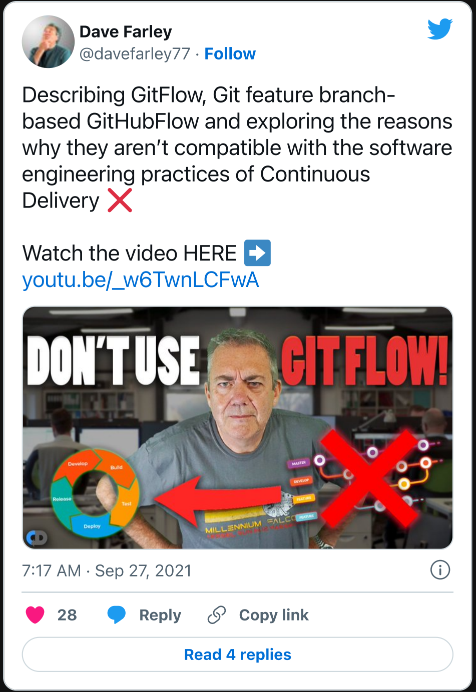
If not GitFlow, then what?

Constant, continuous feedback is ideal. We can't afford to wait too long to check if my changes work with yours.

Branches, by definition, are designed to isolate and hide change. Continuous integration, by definition, is designed to expose change.

GitHub Flow[1] (aka plain old *feature branching*) is a good alternative.

Farley encourages developers to merge directly into trunk, but that seems to subvert the Pull Request reviewing process. PRs are also useful for squashing commits and keeping trunk history tidy.



Dave Farley
@davefarley77 · Follow

Describing GitFlow, Git feature branch-based GitHubFlow and exploring the reasons why they aren't compatible with the software engineering practices of Continuous Delivery ❌

Watch the video HERE ➡️
youtu.be/_w6TwnLCFwA

7:17 AM · Sep 27, 2021

28 Reply Copy link

Read 4 replies

What about marketing releases?

What about marketing releases?

Long-lived branches are eschewed.

Commits are flying into main branch.

A robust testing suite inspires confidence.

What about marketing releases?

Long-lived branches are eschewed.

Commits are flying into main branch.

A robust testing suite inspires confidence.

But what about work done towards a marketing release with a special launch date?

What if a new feature isn’t supposed to be enabled until the New Year, for example?

Feature flags to the rescue

finito