George Stocker
Discovering how to build better software, together.

Please stop recommending Git Flow!

Git-flow is a branching and merging methodology popularized by this blog post, entitled "A Successful Git branching model".

In the last ten years, countless teams have been snookered by the headline and dare I say *lied to*.

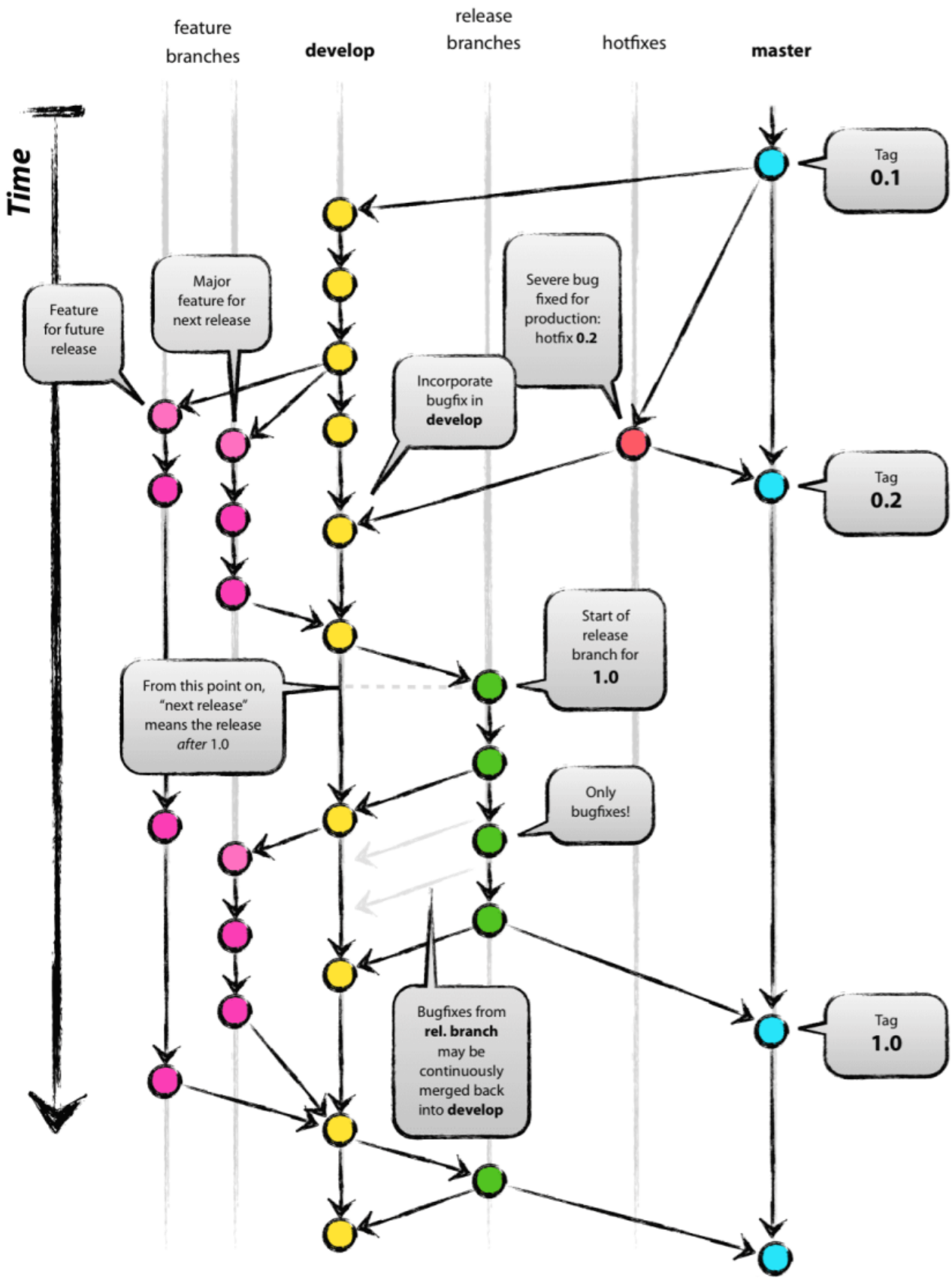
If you read the blog post, the author claims they successfully introduced it in their projects, but purposefully *doesn't talk about the project details that made it successful*.

And for the rest of us, this is mistake #1 of trusting the blog post. I'll claim it as a truism that not all strategies work in all situations, with all people, in all contexts, and I apply that same logic to this branching model.

The end, right? Well, not quite. I can tell a few of you are unconvinced by this line of reasoning, so let's dig deeper into why the gitflow branching model should *die in a fire*.

GitFlow Is Complicated On its Face

Even before you think about Microservices, or continuous delivery, gitflow is complicated. Take a look at this image and tell me it's immediately intuitive:



(source: https://nvie.com/posts/a-successful-git-branching-model/)

So here you have feature branches, release branches, master, develop, a hotfix branch, and git tags. These are all things that have to be tracked, understood, and accounted for in your build and release process.

More so than that, you also need to keep track of what branch is what, *all the time*. The mental model you need to retain for this to be useful carries a high cognitive load. I've been using git for 10 years now, and I'm not even sure I'm at the point where I could mentally keep up with what's going on here.

Gitflow violates the "Short-lived" branches rule

In git, the number of merge conflicts with people committing to a branch will increase with the number of people working on that branch. With git-flow, that number increases even more, because there are three other branches (of varying lifetimes) that merge into develop: Feature branches, release branches, and hot-fixes. So now the potential for merge-conflicts is not linear, it's going to potentially triple the opportunities for merge conflicts.

No thank you.

While I hesitate to say "Worrying about merge conflicts" is a valid reason not to pursue a branching strategy like gitflow; the amount of potential complexity that is introduced when all these branches come together is too much to overlook. This would be fine if you have an organization with a low commit-velocity rate; but for any appreciable fast moving organization or startup, this won't be the case.

Gitflow abandons rebasing

I recognize rebasing is a complex topic; but it's important to this conversation. If you pursue gitflow, you're gonna have to give up rebasing. Remember, rebasing does away with the merge commit — the point where you can see two branches coming together. And with the visual complexity of gitflow, you're going to need to visually track branches, and that means no rebasing if you want to unwind a problem.

Gitflow makes Continuous Delivery Improbable

Continuous delivery is a practice where the team release directly into production with each “check-in” (in reality, a merge to master), in an automated fashion. Look at the mess that is gitflow and explain to me how you’re going to be able to continuously deliver *that*?

The entire branching model is predicated off a predictable, long term release cycle; not off releasing new code every few minutes or hours. There’s too much overhead for that; not to mention one of the central practices of CD is to roll-forward with fixes; and Gitflow treats hotfixes as a separate entity to be carefully preserved and controlled and separated from other work.

Gitflow is impossible to work with in multiple repositories

With the advent of Microservices; there’s been more of a push towards the idea of micro-repos as well (cue commenter shouting “they’re orthogonal to each other”), where individual teams have control over their repositories and workflows, and where they can control who checks in to their repositories and how their workflows work.

Have you ever *tried* a complex branching model like gitflow with multiple teams, and hoped for everyone to be on the same page? Can’t happen. Soon, the system becomes a manifest of the different revisions of the different repositories, and the only people who know where everything is at are the people pounding out the YAML to update the manifests. “What’s in production” becomes an existential question, if you’re not careful.

Gitflow is impossible to work with in a monorepo as well

So if micro-repos are out due to the difficulty in coordinating releases, why not just one big branching workflow that all the microservices teams have to abide by for releases?

This works for about 3.2 seconds, or the time it takes for a team to say "This has to go out now", when the other teams aren't ready for their stuff to be released. If teams are independent and microservices should be independently deployable, you can't very well tie your workflow to the centralized branching model you created in your mono-repo.

Who should (and shouldn't) use Gitflow?

If your organization is on a monthly or quarterly release cycle and it's a team that works on multiple releases in parallel, Gitflow *may* be a good choice for you. If your team is a startup, or an internet-facing website or web application, where you may have multiple releases in a day; gitflow isn't good for you. If your team is small (under 10 people), gitflow puts too much ceremony and overhead into your work.

If your teams, on the other hand, are 20+ people working on parallel releases, gitflow introduces just enough ceremony to ensure you don't mess things up.

Ok, so my team shouldn't use gitflow. What should we use?

I can't answer that. Not all branching models work for all teams, in all contexts, and all cultures. If you practice CD, you want something that streamlines your process as much as possible. Some people swear by Trunk-based development and feature flags. However, those scare the hell out of me from a testing perspective.

The crucial point I'm making *is to is ask questions* of your team: What problems will this branching model help us solve? What problems will it create? What sorts of development will this model encourage? Do we want to encourage that behavior? 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'.

Author's end note: I thought about using the 'considered harmful' moniker that is so common in posts like this; but then did a google search and realized someone else already wrote Gitflow considered harmful. That article is also worth your time.

 geostock / March 4, 2020 / Uncategorized

61 thoughts on "Please stop recommending Git Flow!"

Stephen Nield

March 4, 2020 at 10:03 pm

This is an interesting read. I'd be interested to know your own git workflow as at least one viable alternative.

Louie Christie

March 5, 2020 at 1:22 am

Ok, so my team shouldn't use gitflow. What should we use?

CR Drost

March 5, 2020 at 12:52 pm

Use Threeflow! (The "No Machete Juggling" blog if you have trouble googling it)

I mean every use-case is going to be different but if you are at a small startupy place with repos for single apps, Threeflow is going to be a solid improvement.

The basic idea is that you want in this case everyone in the same room having the same conversation, so you commit to the idea that we should have a place

where we merge dev code on the daily as our source of truth about the system. No long-lived feature branches. We subordinate all the rest of our process to this vision. "But I need to control what features get deployed in my releases"—Not A Problem For Your Git!! That needs to be handled in the logic of the repo itself, either with DB states or env vars that push that question to the last possible moment or by a checked in file that defines statically what features are enabled, which devs locally overwrite. "But if we have 7 feature toggles then the QA team has to test 2^7 = 128 different configurations, maybe"—you have to get them talking with everybody better and also restrict the flow of feature requests into your development system so that it is driven by your slowest step, a limit on the number of feature toggles we allow: and you have to clean up branches after they have been deployed for a week. "But my devs check in breaking code"—you need test suites, feature toggles, code review to make sure they don't, then.

You subordinate everything else, "I will do whatever I need to, to ensure that everyone is in the same room having the same conversation about the same codebase. That is the one thing I will not budge on, that all my developers checks in their work daily and see merge conflicts early so they can collaborate rather than step on each others' feet." And it works extremely well at smaller scales, IF you can get developer buy-in.

(Developers find feature toggles to just *look ugly* so you have to really lean hard on the fact of "look we need you to do this so that we don't waste half your time in release planning meetings and other crap. I know it is ugly but here are the ways we are going to bound its ugliness and let the code eventually become beautiful.")

Rune

March 9, 2020 at 9:25 am

I have only used feature toggles a handful times, but I suspect it is the way to go.

Assuming you have a sane release policy that is. If you find yourself doing lots of patches while maintaining several versions, then you will be stuck with a

more convoluted work flow.

My least favorite initialism: “LTS”. Imagine patching a LTS branch and then bring that fix forward into your current release as well as your main dev branch. Or better yet: You start out patching the current release and after a while somebody asks “what about those still running the LTS version?”. (my reply usually is: “Fine, the current release is now considered LTS. Have them upgrade as there are a bunch other useful fixes in there too.”) Oh hang on, should we merge the LTS fix into master as well..?

I suspect many of us could embrace Continuous Deployment with much more vigor. At least when the user interface is mostly web and no local deployment is involved. At that point I believe the full git workflow becomes overkill.

Geshan Manandhar

March 5, 2020 at 2:23 am

Use simplified gitflow.

ertre

November 19, 2020 at 10:35 am

This is the answer. You’re not forced to use every branch that GF proposes.

Craig

March 5, 2020 at 3:25 am

So, your post is to pull someone’s idea to pieces while offering nothing of value yourself? I am disappointed this article made it onto HN

Jeff

March 9, 2020 at 11:07 am

He literally wrote a thoughtful reason as to why he didn't recommend something specific and the most important thing of all, to ask questions of your team. Teams should be thinking for themselves more and relying on magical blog posts less. Don't trash someone if you aren't going to take the time to understand their point of view.

Chris Graham

May 19, 2022 at 12:07 am

Devs (like people in general) suffer from a lot of group think and intellectual lazyness. Rather than mandating a solution, the best possible answer is: Ask the team what works for them.
Where this fails however, is when it is a young inexperienced team (who think they know it all) but lack the experience to not get into trouble. That's the time when I mandate things. To save them from themselves.

Kira

March 5, 2020 at 6:56 am

To be honest, you start with:

"doesn't talk about the project details that made it successful"
Well Gitflow is about 10 years old, so there are many good projects already worked with it.

"GitFlow Is Complicated On its Face"
Not really, maybe redundant but not complicated.

"Gitflow is impossible to work with in a monorepo as well"
No, it's maybe not the best solution for a fast agile devEnv. but else its working very well.
Even if you have a bi-weekly release cycle it's working fine.
I would love to share a project of our team, but as you can expect I'm not allowed

Ok, so my team shouldn't use gitflow. What should we use?
You can't answer that?
Well there are so many other workflows that you could give some examples

But yea, you showed some example where you can get problems but to say it's not good is kinda unfair.
So i think, you are not completely wrong, but I also have to say that you don't really grasp the concept behind gitflow as it's not really complicated if you understand git.
It's also discussed very often in different forums and so on.

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

---

Leszek

May 6, 2021 at 4:32 am

exactly.

Ian

March 5, 2020 at 10:27 am

Quite frankly, Git should die in a fire.

Not that what Git does is bad. But the idea that it needed no useful tree structured visual interface that looked somewhat like file explorer, SourceSafe or team foundation server is just programmer arrogance at its worst. Those interfaces happened for a solid human-factors *reason* and were not whimsical afterthoughts. They gave the user a mental model, showed both local and remote server information accurately and allowed pinpoint actions on a file, folder or project that you could plainly SEE. Command line options were available and I used them in batch files frequently, but the batch files were useful because I could SEE what I was doing, simultaneously, both locally and on the server.

Visibility and local action is what made older version control systems *immediately* useful with almost no training. Git, SourceTree GitHub desktop, et. al. Not so much. Git seems *designed* to hide information, possibly with the intent of preventing mistakes. Of course, if you make it difficult to make easy mistakes, you can also make it extremely difficult to do anything.

It's not that Git isn't learnable. I use it daily. It's just that it was a poorly thought out alternative. It's *technically* great. It's the human factors part that's a disaster.

And frankly, if you don't understand the human nervous system and you're making a human facing product, you're a lousy engineer because you either don't understand or ignore half the system you're building.

DarkSwordsman

March 5, 2020 at 1:45 pm

I've been a developer for only 2 or 3 years and I have no problem visualizing git branches and what not in the terminal. The only user interface I use is the Github website, and that's strictly for issues and PRs. I rarely look at actual branch graphs or anything.

erwer

November 19, 2020 at 10:40 am

I've been a developer for ~25 years at large projects and companies like IBM, and bleeding edge ones in Manhattan. I find Git to be overengineered and a lot of that is forced on the user developer. If you're a new developer and have lived in a vacuum you might not have experienced systems that don't have anywhere near that level of un-intuitiveness but still have most of the power. This is why big selling points of Git clients is to obscure that, which they can only do so much.

Gottfried Theimer

March 5, 2020 at 10:29 pm

“It’s not that Git isn’t learnable. I use it daily. It’s just that it was a poorly thought out alternative. It’s *technically* great. It’s the human factors part that’s a disaster.”

I agree with this and add that in this respect Git is typical for what comes from the Linux world.

Fabricio Bertoncello Scariot

September 28, 2020 at 8:58 pm

> Said everything I wanted to say, everything is turning into Linux, we are wasting time decorating and typing commands instead of spending this time on business improvements.

Thomas Wheeler

March 5, 2020 at 10:42 pm

“Quite frankly, Git should die in a fire.”

I would like to take your words out back and shoot them. Git’s commandline interface is quite rightly criticized, but git is the only versioning system I’ve used that gets it right[1]. Period. Its DAG, and especially, the concept that every commit represents the entire working tree at a moment-in-time, is both simple and correct.

100x over and over, I would rather use a tool that gets it right and has a completely shit user interface, than a tool that has a beautiful user interface but doesn’t get things right.

[1] I’ve heard great things about Mercurial and couple other tools, but I know git and this thread is about git.

Fabricio Bertoncello Scariot

September 28, 2020 at 9:01 pm

For those who knew SourceSafe, CVS and SVN, it doesn't, Git doesn't make any sense. I didn't see any benefit, and until today I haven't met anyone who uses the big difference that is the local commit

---

erertre

November 19, 2020 at 10:42 am

It's possible to have both. Software development didn't start 15 years ago and was just fine.

John Blanco

April 14, 2021 at 10:39 am

Sure, software development existed more than 15 years ago before git, but let me tell you what it was like:

I used to use Subversion. Making branches was a nightmare of nightmares. I never did it on my own, we hired somebody to manager it. We paid someone 50K a year to do just that.

Perforce…oh, Perforce. To avoid conflicts, you LOCKED files in Perforce. Wanted to make a change to a source code file, better hope nobody was doing it first! They literally solved conflicts by not letting ANY TWO PEOPLE work on the same file at the same time.

SourceSafe? We had a TEAM of SourceSafe engineers to manage that shit.

So yeah, git is pretty good? Can it be complex? At times, maybe. But it's solving a VERY COMPLEX problem. Be thankful!

John

March 5, 2020 at 11:34 pm

git –all –decorate –oneline –graph
Alias that to ‘git adog’.

Git was designed for people who thought a certain way, who subscribed to a certain methodology, and who had a certain level of technical ability. Not all humans are the same. It wasn’t poorly thought out – it just wasn’t what you wanted. The designers of git aren’t the ones suffering from programmer arrogance at it’s worst.

Eduardo Brites

March 6, 2020 at 7:56 am

I love Git and I agree with you, I use Git on Visual Studio 2019 and I can’t rebember the last time I needed to use the command line.

Bloodgain

September 18, 2020 at 9:00 pm

You have fair criticisms of the git command line interface, but git was never really supposed to be the direct interface for the general user. That’s why GUIs were developed for it. These GUIs *should be* developed separately, and the design philosophy of git is to support outside tools by making everything available, including the plumbing and the repository tree itself.

That said, for power users (i.e. me), the command line porcelain commands are excellent. I sometimes have to reference a doc page (`git help log`) to remember the options — just like everything else in shell world — but I can find out anything I want with a command or three. Yes, the litany of options available for commands like `git log` is overwhelming, but the options are mostly intuitive, and this is exactly the use case that aliases are for. The default output of `git log` is only moderately useful for the most common purposes, which is

true of many other git commands, but it's also the most obvious output: "show me the commit log".

Xedni

October 10, 2020 at 4:53 am

To paraphrase David MacIver:
"Optimising your notation to not confuse people in the first 10 minutes of seeing it but to hinder readability ever after is a really bad mistake."
Coming from svn/tfs/visit, yes. Git is daunting. I'll admit it took me about a good year to feel like I wasn't an active threat to the consistency of any code base I worked on. But getting serious about understanding Git is essential to being a programmer because a) its actually *really* well designed and b) people know it. If all you're after is a Gui, there are plenty (Git ships with a minimalist Gui; another example is GitKraken). But the real reason it seems poorly designed is (I assert) user error. Git is dramatically different from most other source control systems. harpsichord when you're used to playing an accordion. Yes they both make noise, but the similarities don't extend much beyond that. So "fixing" git to be more like svn is like asking your car dealer to put the swimming pool back in your car. Cars don't have swimming pools, and you'll just end up with a car that handles like crap, with a pool nobody ever swims in.

All poetic waxing aside, if you're still shakey with Git, I'd recommend this video:
https://youtu.be/1ffBJ4sVUb4
You may not come out the other side ready to merge 10 different code branches while spinning a basketball on your nose, but you *will* understand the design decisions that went into it, and identify places you may have been hammering square pegs into round holes.

---

ewtwe

November 19, 2020 at 10:38 am

I agree, and the terrible unintuitive naming of certain functions is misleading at times. I still get terrified sometimes when I perform certain functions because

several times in the past it's exploded.
The whole system is overly complicated compared to something logical like good old sourcesafe (which does indeed have it's own issues)

I use Git daily, and Gitkracken is a fantastic UI, but I still think the technology has major issues.

Bryan Finster

March 5, 2020 at 10:36 am

So much good here. Posting this link internally.

Piotr Mionskowski

March 5, 2020 at 12:09 pm

I never liked git flow as well. For us, assuming a small team, the following works good https://brightinventions.pl/blog/how-do-we-use-git/

Carl

March 5, 2020 at 12:14 pm

Git flow is good enough to get started and will actually take you really far. It's a recipe for how to use branches effectively, because they're cheap.
Now there are a lot of other branching models that are good for a lot of different scenarios.
Complaining about one branching model without even recommending when the others are suitable makes for poor reading.

Rod

March 5, 2020 at 12:14 pm

I recommend Three-Flow as an alternative:
https://www.nomachetejuggling.com/2017/04/09/a-different-branching-

strategy/

Tristan Zimmerman

March 5, 2020 at 12:40 pm

I could not agree more and I had the same reaction when I saw that diagram many years ago.

For anyone asking what to do instead of Gitflow, I'd suggest this: Start simple.

– Don't commit to Master (obvs).
– All work is done in a separate branch that will be merged to master.
– Keep your PRs small so they are easy to understand and easy to review.
– Keep Master in a constantly deployable state.

To handle the complications of adding in large features, find ways to hide them in prod so they can get into Master as early as possible. Feature flags are fantastic for this, but even building a temporary route so you can view the changes but not expose them to your user will work just fine.

Also, writing tests for work also makes small, frequent merges less of a dangerous game. They aren't perfect, but if everything has a test around it then you're less likely to break things even if you're merging a dozen PRs a day.

Anyway, love the post. 👍

Bill

July 13, 2020 at 8:38 am

This is essentially gitflow, the author has somehow confused feature branching with branches living forever.

I'm not entirely sure why he believes that gitflow does not work with CI/CD since when using gitflow one only releases from a release branch, which is

taken from the development branch, which is a composite of all of the (short lived) feature branches.

feature/hotfix/bugfix branches are short lived, nothing in gitflow says that these branches should live longer than they are needed

gitflow might not be best suited to everyone and every project, but it certainly does not pose any problems for continuous integration or delivery. It does mean that a team does not need to implement feature hiding at all -which generally entails extra code which is superfluous to the actual solution.

Ryan Cline

March 5, 2020 at 3:58 pm

To pick up where the author left off:

In short, Trunk-based Development is the solution you are looking for.
https://trunkbaseddevelopment.com/

Trunk-based Development has it all, and that website should be considered required reading for anyone implementing or upgrading their personal or professional development workflow.

If that site doesn't convince you, then read State of Devops 2019
https://services.google.com/fh/files/misc/state-of-devOps-2019.pdf

So long GitFlow, what we had...was not good.

Cheers, to your better development future! Welcome!

Bill

July 13, 2020 at 8:40 am

I'm really not convinced that trunkbaseddevelopment differs from gitflow at all

---

Eric Rini

March 5, 2020 at 4:43 pm

Yup. Git flow is a good example of people knowing what to do and forgetting why they were doing it.

Ben

March 5, 2020 at 5:00 pm

I echo the other commenters: can you give an example of a project you worked on, and what Git branching model was successful and why? Otherwise this post only serves to discourage and confuse people like myself who have used Gitflow in the past.

d potter

March 5, 2020 at 6:24 pm

Huh? There are way less merge conflicts when using git flow correctly. The diagram above is missing a few ghost merges from develop down to feature branches (every time there's a push by someone else to development, specifically). But that's one of the main draws – having a highly hierarchical and rigidly adhered-to branching strategy eliminates merge conflicts that aren't directly related to code the developer who's doing the pull created themselves.

I could go point by point on why it seems like you've never worked with people who knew how to do this correctly and why that's coloring your opinion, but I just got a 70 year old on this model from TFS last year and got him loving it, so it's not too late for you.

But just FYI: "This has to go out now" is how you know to use a hotfix branch which works against only production/master code. "What's in production" is always easy – it's what's in master. Continuous delivery is a checkbox for development, and you add a combo/dropdown that gets updated once per release for QA to point to the new release branch. And if you can't coordinate releases and

shared code with GitFlow, you're not going to have any better time rolling your own structure and keeping track of that being different every release...

Good luck out there.

Bill

July 13, 2020 at 8:42 am

I agree 100%

Using gitflow is a change in mindset, but it certainly does not prevent CI or CD.

I can't get my head around why the author assumes that feature branches are not short lived when using gitflow

---

Zeno Lee

March 5, 2020 at 11:09 pm

Trunk based development is a great simple alternative

Brent DeMark

March 7, 2020 at 7:52 pm

This is what we've been using for years. And it works well. I would say start here, and if you need something more, then look to more involved branching strategies like git flow. I would caution against jumping to something so complicated right out of the gate for new projects.

Bill

July 13, 2020 at 8:43 am

It's actually no different to gitflow, so it's not really an alternative

---

Scott Davey

March 6, 2020 at 3:46 am

I've been successfully using Git Flow for well over 5 years in a multi-repo, micro-service, continuous delivery environment and I've found it to be a really good fit for our team.

What Git Flow did for us is to standardise a development flow, which removed differences between our Git experts and others, and gave everyone a shared mental model so make teamwork better.

Despite this blog post's experiences, in our team it actually reduced merge conflicts, reduced long running branches, and eliminated lost branches because it also eliminated tricky git-fu merges of topic branches along with a bike-shedding.

It gave our workflow better clarity.
We have even tied our processes to git flow concepts, and this meant we all had a shared understanding across the team including devs, ops and even managers, who knew, for example what type of work could be a hotfixed vs needing a full release.

Could all these benefits come another way? Well, certainly. But for us there was a clear before/after transition where all our prior git problems disappeared after we introduced Git Flow. It isn't perfect, but it solves more problems than it creates for us, and we have made it work in a continuous delivery, micro-service and buzzword compliant environment.

Paul Hammant

March 6, 2020 at 4:19 am

Hey George, it's trunk-based development you're meaning not trunkless development. It's been written and talked about for 20 years now 🙂

Salem Korayem

March 6, 2020 at 4:46 am

Git flow is a scalable methodology that you. The one in the diagram is the extreme case.

In my company, we use master, staging (like develop branch in the diagram) and feature branch everything from staging. That's it. No hotfix or release branch.

We link all PRs with Jira through a script which updates Jira issue a custom field that specifies whether this issue has been merged or not and if merged, in which branch: staging or master or both. This way we can easily do a JQL and know where is each issue at.

If I feature branch from staging today and by the time I finish work, the staging bench gets updates, I rebase the feature branch onto staging.

The fact that you dont suggest an alternative might imply you didn't try anything else that actually works before writing this post.

Bob Barbell

March 6, 2020 at 8:20 am

This publication level is around this:

Cars are bad! Because:
– demands on gas
– you should have driving licence and study laws to drive a car
– they move only when you start engine and press pedals
– requires periodical service
– there are higher risks of car accident
– you must decide which color your car should have

Any approach have its own advantages and disadvantages. And it is about knowing how to use accessible tools properly and not about proclaiming of "stop

recommending this tool”. Gitflow is the perfect tool for delivering changes to code and it is definitely better than many custom approaches people usually invent.
Gitflow real disadvantages mostly affect project management and planning processes and not the software development itself.

Captain Obvious

March 6, 2020 at 9:07 am

So to sum up: there is not only a git-flow approach and you should choose from teams need. Also sounds like a problem to handle more than one branch

---

Anushervon Saidmuradov

March 6, 2020 at 5:13 pm

This is something that works for us:

– Single master branch where all the changes go. No commits made here directly (exception can be made for hotfixes)
– Feature branches are created from the master HEAD and should be short lived.
– Environment branches (dev, stage, prod), unlike feature branches, are permanent and are always rebased from latest HEAD of master. Changes to these branches trigger the CI/CD pipeline to deploy into those environments automatically. We use separate Kubernetes clusters hosted on GCP/Azure/AWS as the deployment environments. No commits should be made directly to these branches, all commits go to master via feature branches

---

Daniel Marbach

March 7, 2020 at 4:04 am

We had GitFlow for many years and the biggest problem we had is that people continuously forgot to bring back hotfixes into develop branch. Now with release flow this problem is gone and we can still maintain and hotfix multiple supported versions which is a key concern for us

https://docs.microsoft.com/en-us/azure/devops/learn/devops-at-microsoft/release-flow

---

Iurii

March 10, 2020 at 9:12 pm

1. What is on PRODUCTION? That's simple – master 🙂
2. Why are you mixing microservices and GIT? Why microservice team should depend on another? That's whole point that teams are fully independent one can has Git, another Subversion or X(put name). Different versioning etc.
3. Propose rather than blame.

Stefano

March 14, 2020 at 7:25 am

I stopped reading at the paragraph about rebaseing. Dude, if you "recognize rebasing is a complex topic", go back to the classroom and learn GIT from ground up. Rebase is – if you once understood it – not at all complex. I use GitFlow on a daily base and are able to rebase.

Bill

July 13, 2020 at 8:47 am

rebasing instead of merging from dev is a good practice IMO, and gitflow really does not prevent this at all

---

Alex

April 30, 2021 at 11:29 am

This was also my impression.

The author has the same complaints as junior devs first introduced to git and CI/CD as a concept. After you daily drive gitflow for a couple weeks and see the

issues it prevents/negotiates you never go back.

LincWong

March 19, 2020 at 9:38 pm

i think gitlab-flow is the answer.

Gesiel Kloeppel

March 20, 2020 at 4:16 pm

I don't understand how so many ppl like GitFlow. Are those ppl managing releases and hotfixes? I doubt it. If you think you like it and you don't have release and hotfix branches, you probably don't do GitFlow.

In my understand, GitFlow is valuable just if you need to support more than one version of your software. In other cases, a production-like master is enough.

Alexey Lebedev

June 5, 2020 at 11:25 am

Dear George,
Thank you so much for this post! It hurts me to see people using gitflow. I personally think that it's very ineffective and awkward model.
I remember how we were choosing branching model. We contiously rejected gitflow (and it was the best decision we ever made), although it became very popular.
We came up with the following quite simple branching model:
-we have master branch, which correponds to prod
-if we want to deploy some changes on prod, we prepare or branch, which will be merged into master
-branches for deployed tasks are merged into branch.
– is merged into master
-BEFORE ANY BRANCH IS MERGED IT MUST BE REBASED, conflicts resolved and ALL MERGES ARE ALWAYS FAST-FORWARD (without merge-commits)

We had 20+ people commiting in the same repository and our git history was clean and clear without a single merge commit. We've been using this model for years and it proved to be very good.
I feel bad when I look at people's repositories with all these merge commits... How do they even know, what's going on there??
Yes, you must know how to use rebase. But if you are professional developer you must be good with git

---

Bill

July 13, 2020 at 8:49 am

How is this any different to using gitflow?

Pingback: Weekly CW011-2020 – [ONLY DEV INSTANCE!] – Findings and thingies

---

ograterol

May 8, 2021 at 7:28 pm

Interesting ... I think gitflow is an alternative, but there are others that offer more capabilities. I like gitlab or github flow.

Pingback: This Is How We Branch - laredoute.io

Rineez Ahmed

July 18, 2021 at 11:09 pm

Just finished reading this and I feel so ClickBaited! x-(

While there are some valid points about not blindly following some workflow just because some people calls that defacto or standard(that's true for all practices,

not just workflow) the gist of the article doesn't match the title of the article. This post itself is recommending Gitflow for certain cases. So definitely CLICKBAIT!

Vali Spam

September 17, 2021 at 4:58 am

I call this a click bait article.

It has a catchy title "Please stop recommending Git Flow!". The title makes it sounds like an end-all-be-all take on gitflow. It then goes on to argue how you should not use gitflow if you have a microservices architecture which gets deployed often.

Most developers work on line-of-business apps where a conservative develop-qa-release-someday cycle is used. Most developers in my neck of the woods work on such systems.

Damien

November 17, 2021 at 6:05 pm

Really depends on how you use it. We have been using gitflow for 6 years and never had any issues with it. I still swear by it.