

# git-flow cheatsheet

Fork me on GitHub

created by Daniel Kummer



efficient branching using git-flow by Vincent Driessen

translations: English - Castellano - Português Brasileiro - 繁體中文(Traditional Chinese) - 简体中文(Simplified Chinese) - 日本語 - Türkçe - 한국어(Korean) - Français - Italiano - Nederlands - Русский(Russian) - Deutsch(German) - Català(Catalan) - Română(Romanian) - Ελληνικά(Greek) - Українська(Ukrainian) - Tiếng Việt(Vietnamese) - Polski - العربية - فارسی - Lietuviškai(Lithuanian) - Azərbaycanca(Azerbaijani) - Bahasa Indonesia

## About

git-flow are a set of git extensions to provide high-level repository operations for Vincent Driessen's branching model.

more



This cheatsheet shows the basic usage and effect of git-flow operations



## Basic tips

- ★ Git flow provides excellent command line help and output.  
Read it carefully to see what's happening...
- ★ The macOS/Windows Client [\*Sourcetree\*](#) is an excellent git gui  
and provides git-flow support
- ★ Git-flow is a merge based solution. It doesn't rebase feature  
branches.

★ ★ ★

## *Setup*

- ★ You need a working git installation as prerequisite.
- ★ Git flow works on macOS, Linux and Windows

★ ★ ★

### *macOS*

Homebrew

```
$ brew install git-  
flow-avh
```

Macports

```
$ port install git-  
flow-avh
```

### *Linux*

```
$ apt-get install git-  
flow
```

### *Windows (Cygwin)*

```
$ wget -q -O - --no-  
check-certificate  
https://raw.githubusercontent.com/  
petervanderdoes/gitflow-  
avh/develop/contrib/git  
flow-installer.sh  
install stable | bash
```

For detailed git flow  
installation instructions please  
visit the [\*git flow wiki\*](#).



You need wget and util-linux  
to install git-flow.

# *Getting started*

Git flow needs to be initialized in order to customize your project  
setup.

★ ★ ★

## *Initialize*

Start using git-flow by  
initializing it inside an existing  
git repository:

```
git flow init
```

You'll have to answer a few  
questions regarding the  
naming conventions for your  
branches.

It's recommended to use the  
default values.



## *Features*

- ★ Develop new features for upcoming releases
- ★ Typically exist in developers repos only

★ ★ ★

## *Start a new feature*

Development of new features starting from the 'develop' branch.

Start developing a new feature with

```
git flow feature start MYFEATURE
```

This action creates a new feature branch based on 'develop' and switches to it

## ***Finish up a feature***

Finish the development of a feature. This action performs the following

- ★ Merges MYFEATURE into 'develop'
- ★ Removes the feature branch
- ★ Switches back to 'develop' branch

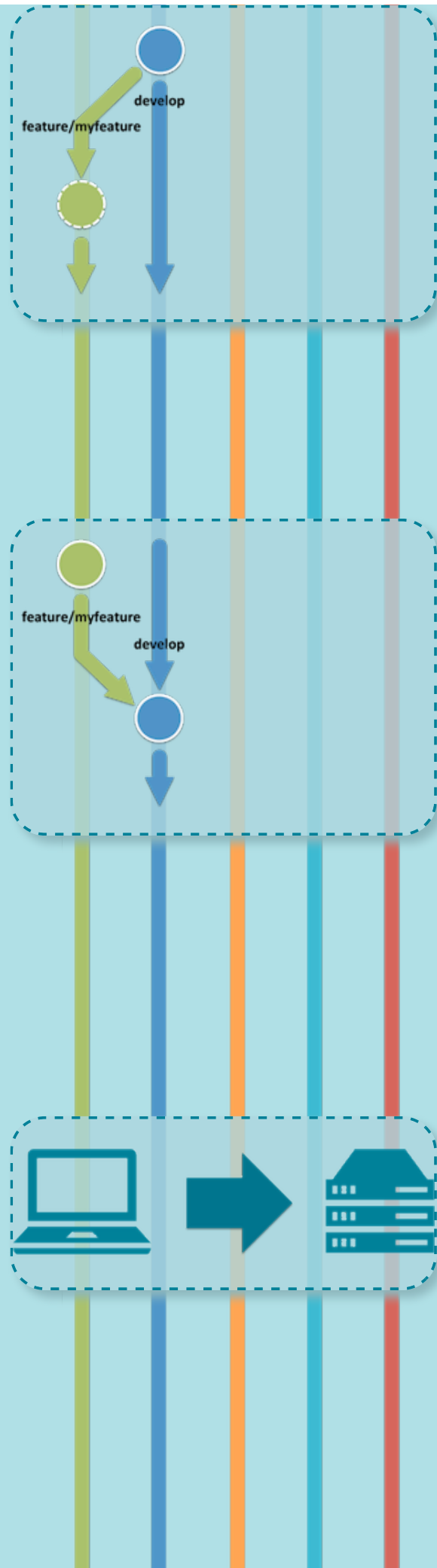
```
git flow feature finish MYFEATURE
```

## ***Publish a feature***

Are you developing a feature in collaboration?

Publish a feature to the remote server so it can be used by other users.

```
git flow feature publish MYFEATURE
```



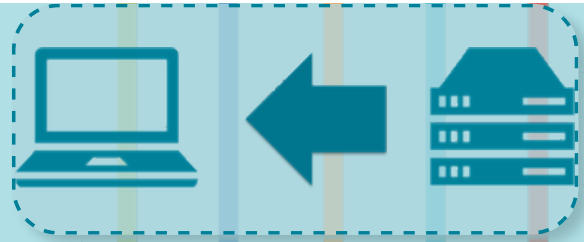
## Getting a published feature

Get a feature published by another user.

```
git flow feature pull  
origin MYFEATURE
```

You can track a feature on origin by using

```
git flow feature track  
MYFEATURE
```



## Make a release

- ★ Support preparation of a new production release
- ★ Allow for minor bug fixes and preparing meta-data for a release

★ ★ ★

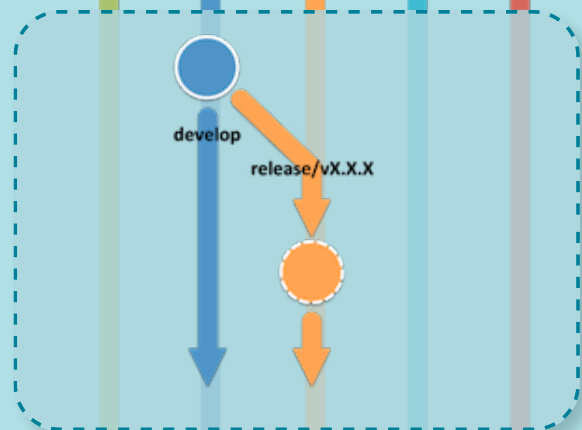
### Start a release

To start a release, use the git flow release command. It creates a release branch created from the 'develop' branch.

```
git flow release start  
RELEASE [BASE]
```

You can optionally supply a [BASE] commit sha-1 hash to start the release from. The commit must be on the 'develop' branch.

★ ★ ★



It's wise to publish the release branch after creating it to allow release commits by other developers. Do it similar to feature publishing with the command:

```
git flow release  
publish RELEASE
```

(You can track a remote release with the

```
git flow release track RELEASE  
command)
```

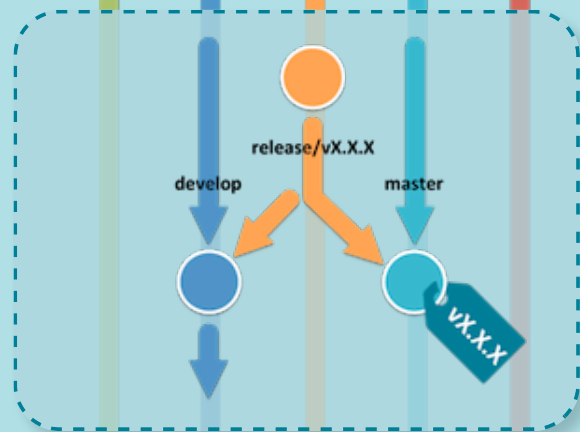
## ***Finish up a release***

Finishing a release is one of the big steps in git branching. It performs several actions:

- ★ Merges the release branch back into 'master'
- ★ Tags the release with its name
- ★ Back-merges the release into 'develop'
- ★ Removes the release branch

```
git flow release finish  
RELEASE
```

Don't forget to push your tags with `git push origin --tags`



## ***Hotfixes***

- ★ Hotfixes arise from the necessity to act immediately upon an undesired state of a live production version

★ May be branched off from the corresponding tag on the master branch that marks the production version.

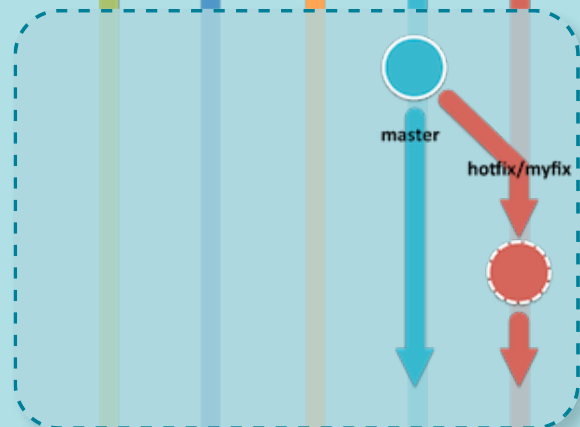
★ ★ ★

## ***git flow hotfix start***

Like the other git flow commands, a hotfix is started with

```
git flow hotfix start  
VERSION [BASENAME]
```

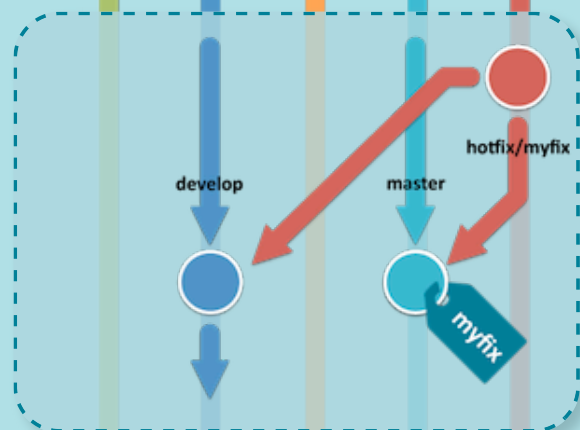
The version argument hereby marks the new hotfix release name. Optionally you can specify a basename to start from.



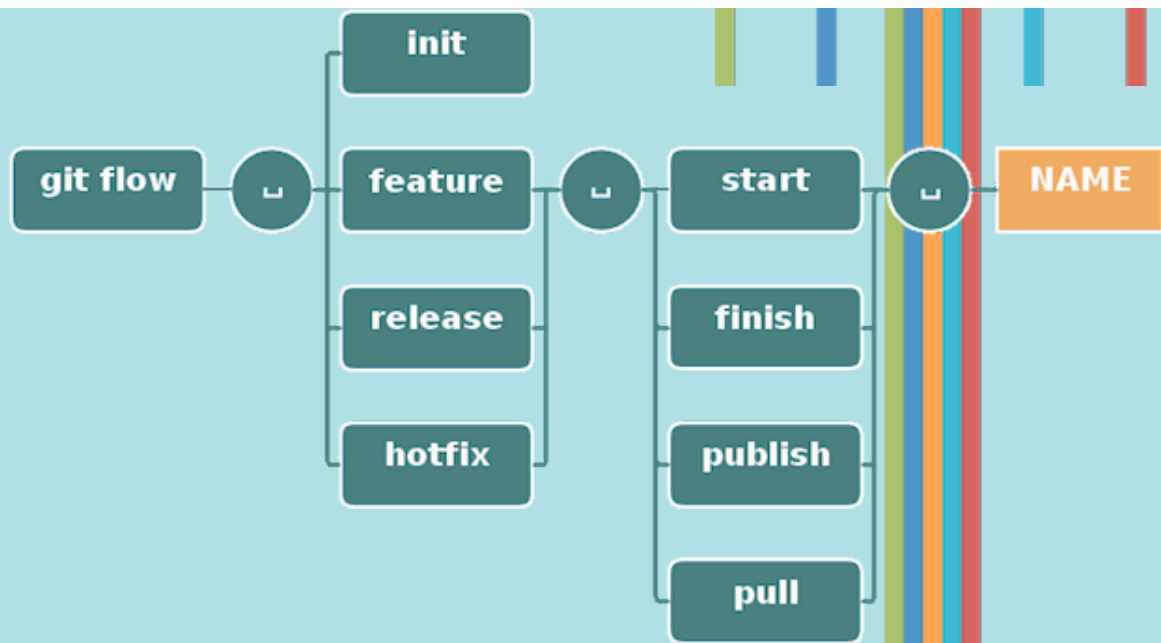
## ***Finish a hotfix***

By finishing a hotfix it gets merged back into develop and master. Additionally the master merge is tagged with the hotfix version.

```
git flow hotfix finish  
VERSION
```



# ***Commands***



## *Backlog*



- ★ Not all available commands are covered here, only the most important ones
- ★ You can still use git and all its commands normally as you know them, git flow is only a tooling collection
- ★ The 'support' feature is still beta, using it is not advised
- ★ If you'd like to supply translations I'd be happy to integrate them



## *Comments*



## Sponsored Links

**Koka dina ägg på detta viset - Resultatet är enastående!**

Shopido

**Beräkna värdet på din bil**

vikoperdinbil.se

**U.S. grants 55,000 Green Cards cards! Can you win? Free check!**

U.S Green Card - Free check

**Take Control Of Your Finances And Earn Profit With This Tool!**

letyshops

**Spelet kan hålla dig vaken hela natten**

Good Game Empire

**App gjord av 150+ språkexperter får dig att prata ett nytt språk på 3 veckor**

Babbel

41 Comments

git flow cheatsheet

1 Login ▾

Recommend 26

Tweet

Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?



Name

**Ben Yitzhaki** · 2 years ago

what is the best practice for fixing issues in release sprint while a new release is already on development (so we can't edit the develop branch)? do them directly on the release branch? how can we handle Pull Request for the fixes done on the release branch etc..?

3 ^ | ▾ · Reply · Share ›



**Evan Butler** → Ben Yitzhaki · a year ago

IMO yes, make a branch off of releaseCandidate, then just create a PR to merge the fix branch into releaseCandidate. Then the fix goes out with that release. After the release is merged with master, you can backmerge master into develop to get those changes into develop.

^ | ▾ · Reply · Share ›



**HK** → Ben Yitzhaki · 2 years ago

in git flow the develop branch contains the code for the next release. if this code needs bugfixes then you are not ready for the release and should continue with the development version of the code.

^ | v · Reply · Share ›

**Ben Yitzhaki** → HK · 2 years ago

Thats not practical. In real life, you have bugs on the release while you got to continue working on new features and eventually merge them to develop, even if there are still open issues on the previous release

10 ^ | v · Reply · Share ›

[Show more replies](#)

**Sidney de Moraes** · 2 years ago

"Commands" section misses the "track command", doesn't it?

2 ^ | v · Reply · Share ›

**danielkummer** **Mod** → Sidney de Moraes · 2 years ago

You're right! However I currently don't have the time to adjust the graphics for it, I'm sorry about that.

^ | v · Reply · Share ›

**Sidney de Moraes** → danielkummer · 2 years ago

Can I help?

^ | v · Reply · Share ›

[Show more replies](#)

**Rizky Syazuli** · 2 years ago

question: should i deploy before or after finishing the release/hotfix? another thing, i want to have a build task that appends the version number to my js filenames. supposedly i can do it by using `git describe` command to get the latest tag. but if i haven't finish the release/hotfix, that command only gets the previous release/hotfix, and not the current one.

2 ^ | v · Reply · Share ›

**Greg Quinn** → Rizky Syazuli · a year ago

You should always merge your release or hotfix back into master first, then tag the release in master, then deploy from master. Think of master as your single point of reference for all versions of your software you have released.

^ | v · Reply · Share ›

**Ren Lawrence** · 2 years ago

What do you suggest is the best practice when there are commits in develop that are not ready for the release when it is cut (maybe it's a feature client is not trained for yet)? I suppose the answer is: cut the release earlier, but that's not always practical. I've been in the habit of either adding revert commits to the release branch or cherry-picking commits into the release branch, but that hardly feels like best practice.

1 ^ | v · Reply · Share ›

**danielkummer** **Mod** → Ren Lawrence · 2 years ago

I'm not sure I understand your question correctly - but if you're able to pick a specific commit in the develop branch you deem "production ready" you can simply checkout that specific commit via its SHA-1 hash and then branch away the current release branch from there. If you need another specific commit further "up" the history for that specific release doing a cherry pick is a valid approach in my opinion as it's a lot easier and less error prone than creating a patch file...

^ | v · Reply · Share ›



**Ren Lawrence** → danielkummer · 2 years ago

The problem is, that the "production ready" features might be mixed up chronologically with "non-production ready" features, so you can't just reset the head in that case.

To avoid cherry-picking/reverting in the future I think the solution seems to be a.) cut branch early and b.) add feature flags for features that aren't ready yet.

^ | v · Reply · Share ›

[Show more replies](#)



**David** · 3 days ago

First of all thank you for this very useful page!

I've got a question about the *Make a release* > *Finish up a release* paragraph.

Merging any release branch into develop should be quite straightforward each time, eg. without any conflict.

What bothers me is that according to the graph above, merging a release branch into master should be straightforward as well!

Indeed, the merge should work for release branch 1. However, for release branch 2 (and above), merge conflicts will certainly occur.

Why? Because commits being part of release branch 1, previously merged into master, will be merged again when merging release branch 2 into master!

So, my question is: should merging a release branch into master (at least from release branch 2) always be done with force overwrite?

^ | v · Reply · Share ›



**Haisum Usman** · 4 months ago

Thanks for the information.

^ | v · Reply · Share ›



**Pablo Ezequiel Leone Signetti** · 5 months ago

For those who want something more practical...

Hotfixes are from master.

Bugfixes can be either from release/latest or development and tag the version that are fixing, current version on development or the next version in the release/latest branch.

You can bugfix a feature in development.

You can bugfix the release/latest branch before the official release.

Having a release/1.0.0 is useless in some environments, where preproduction is not generated automatically by CI. Our system is serving release/latest in preproduction and we always merge development into release/latest to be able to test the next release. Once is fully finished we create release/{version} to meet the standard and after merging into master we tag the new version.

As you can see we have, hotfix/xxx, feature/xxx, bugfix/xxx, release/xxx, development and master branches.

xxx: task code in Jira, Trello, etc...

Hope this helps!

^ | v · Reply · Share ›



**Vic** · 6 months ago

Awesome information. Especially for people who is starting on Coding world.

Congratulations

^ | v · Reply · Share ›

**Gabriel Maia** · 6 months ago

I'm a newbie to git-flow and I want to implement it in a team that requires code reviewing every time under a pull request. How does pull requests are made using this type of workflow?

^ | v · Reply · Share ›

**lusajo malopa** · a year ago

i keep getting this every time i try to create release branch 'git flow release start 1.1.5' fatal: '[D?D?D?D?C?C?C1.1.5' is not a valid branch name.

Fatal: Could not create release branch 1.1.5'.

^ | v · Reply · Share ›

**Nirav** · a year ago

Hello Daniel,

I am confused what to do in my scenario. We have a team with 15 to 20 developer. We have 3 branches as of now. 1) Develop 2) Beta 3) Master

For new feature every one checkout with new local branch from 1)dev. After finishing the feature developer will push the changes to its local and origin dev branch.

In between some other developer had also made the changes and pushed to dev branch with no conflicts.

But now i want to release the 1st developer feature changes to beta branch, and when i try to merge that changes it gives shows the all changes done by other developer as well in the selection.

So how to tackle with these scenario, i there any flow to release feature wise functionality from dev > to beta > to master.

Pls guide.

Thanks in advance.

^ | v · Reply · Share ›

**Tim Anderson** → Nirav · a year ago

Your team should be running git flow feature rebase before finishing a feature to ensure this doesn't happen. Before finishing, run the following: git checkout develop && git pull && git flow feature checkout [your-feature] && git flow feature rebase. Then fix conflicts, git add, git rebase --continue, then git flow feature finish

^ | v · Reply · Share ›

**Vytautas** · a year ago

What if I want to add multiplefixes to a hotfix release this forses me to create tag every time I fix a issue why cant this be optional?

^ | v · Reply · Share ›

**Tim Anderson** → Vytautas · a year ago

Why is this an issue for you? This is the whole idea behind the workflow - to make it easy to understand what has happened and when, and to enforce semver rules.

^ | v · Reply · Share ›

**Carlos** · a year ago

Thanks for the Cheatsheet :D

^ | v · Reply · Share ›

**Armando Ibarra** · 2 years ago

I love it!

^ | v · Reply · Share ›

**yemiwebby** · 2 years ago

This is cool. Very helpful

^ | v · Reply · Share ›

**Jepessen** · 2 years ago

In Master I've 1.0.0 and 2.0.0. Then I have to fix a bug in 1.0.0 and create the 1.0.1 release. How can I put it into the master branch?

^ | v · Reply · Share ›



**HK** → Jepessen · 2 years ago

in this case you would want to use a support branch (i.e. support/1.x) then all hotfixes to version 1.x are branched off from the support branch instead of master branch.

1 ^ | v · Reply · Share ›

**vinamelody** · 2 years ago

Thanks for sharing this, I learn something new about git flow. One question, when you finish the hotfix,

1. do you merge or cherrypick to master?

2. do you merge or cherrypick to develop?

^ | v · Reply · Share ›



**Rizky Syazuli** → vinamelody · 2 years ago

merging to master/develop is done automatically by git-flow when you finish the hotfix

^ | v · Reply · Share ›



**Adrien** · 2 years ago

Excellent page, thank you!

^ | v · Reply · Share ›



**werwer** · 2 years ago

This is cool. But one thing here, when I finish working on a feature, and say git flow feature finish MYFEATURE, it merges that feature to dev branch locally, deletes that feature branch locally. So basically I can't create a pull request for others to review it. :(

^ | v · Reply · Share ›



**Chaitanya Prabhu** → werwer · 2 years ago

It means, the changes are not yet on the remote, so you need to publish it before you raise a pull request. Do git flow feature publish MYFEATURE before you do finish on it and you should be able to create a pull request after that.

^ | v · Reply · Share ›



**Jan Stockfisch** → werwer · 2 years ago

while executing git flow feature try the -k option ;)

<https://github.com/nvie/git...>

^ | v · Reply · Share ›



**Jan Stockfisch** → Jan Stockfisch · 2 years ago

\*git flow feature finish -k

^ | v · Reply · Share ›

[Show more replies](#)

**xtofl** · 2 years ago

Very explanatory! This, however, caused me to frown: `wget ... --no-check-certificate` is bad advice, especially if you execute the content afterwards! If I were you, I would take that off.

^ | v · Reply · Share ›



**danielkummer** Mod → xtofl · 2 years ago

You're right about that! The next cheatsheet version will be without skipping the certificate check.

^ | v · Reply · Share ›

**Pablo Ezequiel Leone Signetti** · 2 years ago

How do you handle git flow during sprint development? I found that during development, some sprint tasks are interdependant, so it isn't possible to branch from master as it's too way back in the history and there are functionality requiered from development branch to continue working on the sprint.

At the moment, I branch from development during the sprint and rebase the branch I'm working on from development. I found that using this way, master still always stable and we avoid doing lots of merges between branches to get the project to the state needed to continue development.

I think this part is missed everywhere, I couldn't find a documented way to avoid all this hassle.

And during development, hotfixes aren't branches from master, because the functionality we fix is probably an issue that was caused by feature conflicts, so we create hotfixes from development. Once development has all the sprint tasks merged and we fixed all hotfixes, we merge development into master. We don't use release branches as we don't have a pre-production server, so there's no point to have it.

But I feel like having development as a kind of master branch during development and change its meaning after development phases is quite confusing. Let me explain it better...

After development phases, the development branch will hold features based on current master branch. While on development phase the new features will be based on development branch.

Could you bring some light to me on how avoid this?

Thank you

^ | v · Reply · Share ›



**Jan** → Pablo Ezequiel Leone Signetti · 2 years ago

Hey Pablo. I don't know the specifics of your projects, but I'd assume that every sprint ends with a release, i.e. current develop gets merged into master. The new sprint starts developing new features from the develop branch. Hotfix branch is derived from master, so you could use it for critical fixes of already deployed features, but if you are fixing stuff in develop branch, it's really not a hotfix (also, you can work directly on develop branch of course).

Now it seems to me that you may be actually missing the release branch in your process; I guess this is the branch where you could fix all the conflicts of the existing features – think of it like a feature freeze, you are polishing the release and getting it ready for the end of sprint, while new features can be added on the develop branch.

Does that make sense?

1 ^ | v · Reply · Share ›



**Pablo Ezequiel Leone Signetti** → Jan · 5 months ago

By the time I wrote that comment I didn't understand git-flow well! :)

Now everything makes more sense – the problem was that git-flow doesn't explain how to deal

Now, everything makes more sense... the problem was that git-flow doesn't explain how to deal with development hotfixes or release hotfixes, even though is described, the names are quite confusing. If the tool used to manage git-flow doesn't allow you to add the version a hotfix is fixing you never know if it's a hotfix from master, release or development... it's difficult to collaborate!

To avoid this in Jira specifically we can state which version is fixing a hotfix, so we know from where to branch from... we made up this part as I didn't find it anywhere.

^ | v • Reply • Share ›

**clarify** • 2 years ago

Great work! for the beginners its very easy to understand the hardships of complexity in Git initially. thanks a ton!

A suggestion:  
if you can add headers to the branch colors on the top would be great.

Question:  
Are the colors used in the branches are standard GIT representations?

thanks again!  
Highly appreciate your work

^ | v • Reply • Share ›

**el3ctron** • 4 years ago

A M A Z I N G !!!

^ | v • Reply • Share ›

#### ALSO ON GIT FLOW CHEATSHEET


##### git-flow 備忘清單

8 comments • 4 years ago

 rayshih — yes

##### Tóm tắt Git-Flow

4 comments • 3 years ago

 HotAnime.me — Phương vẫn đã đọc bài viết này :3 :D


##### Шпаргалка по git-flow

27 comments • 5 years ago

 Иван — Класс

##### git-flow cheatsheet

158 comments • 6 years ago

 Adam Wright — That page-length diagram is truly adorable. My compliments to the designer.