Fork me on GitHub

git-flow cheatsheet

created by Daniel Kummer Tweet

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 - 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 OSX/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 OSX, Linux and Windows

★ ★ ★

OSX

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.github.com/petervanderdoes/gitflow-avh/develop/contrib/gitflow-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

★ ★ ★

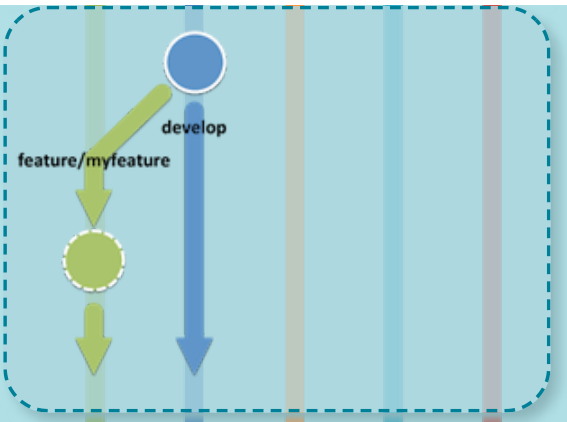
Start a new feature

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

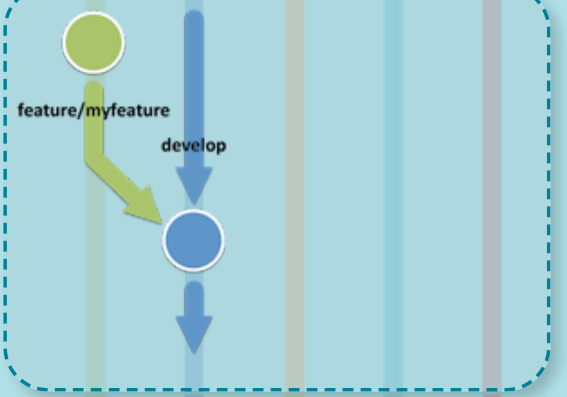


Finish up a feature

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

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

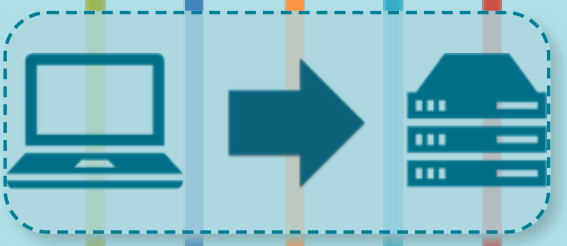
```
git flow feature finish MYFEATURE
```



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

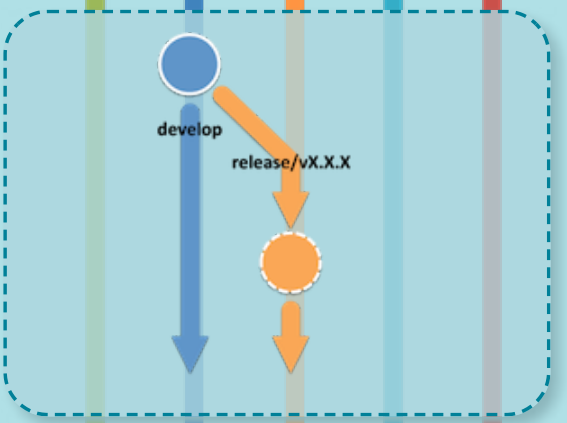
```
git flow feature track
MYFEATURE
```

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.

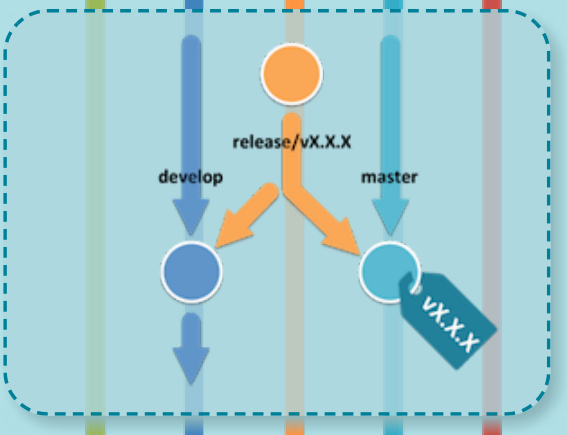
★ ★ ★

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 --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.

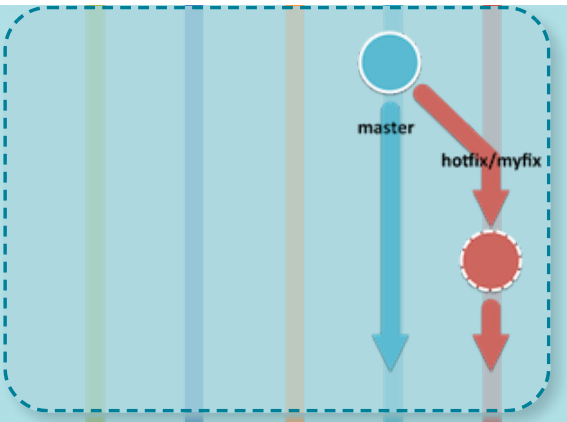
★ ★ ★

git flow hotfix start

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

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

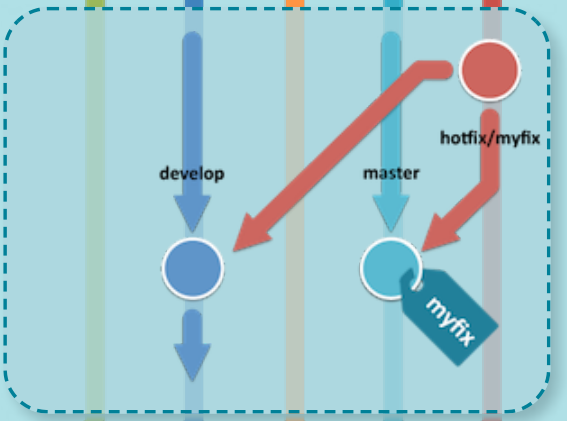
The version argument hereby marks the new hotfix release name. Optionally you can specify a basename to start from.



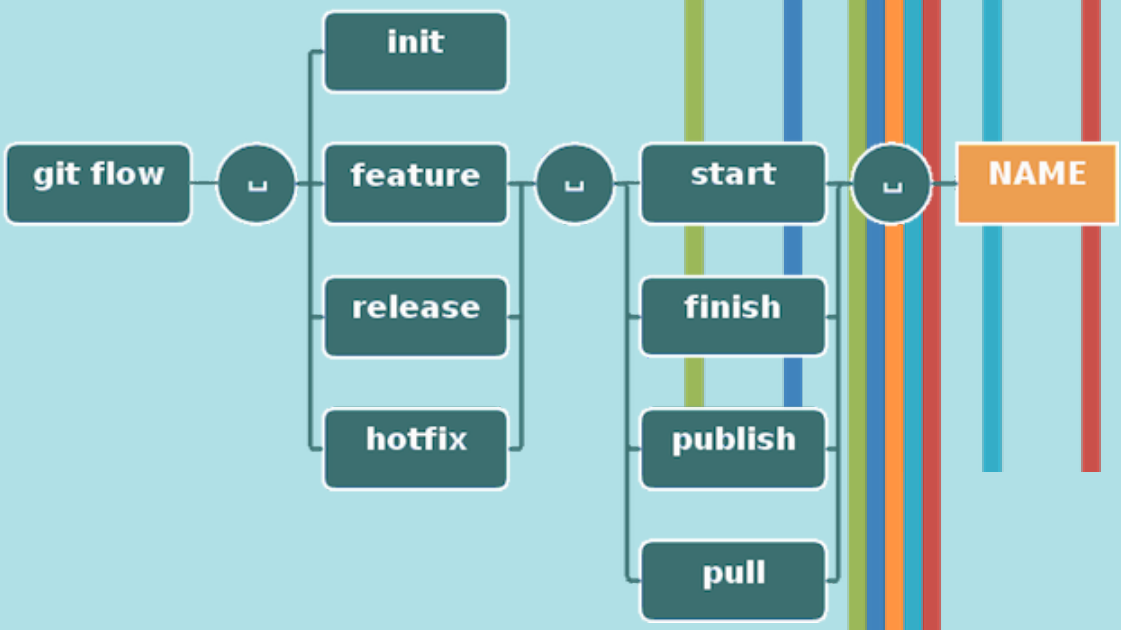
Finish a hotfix

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



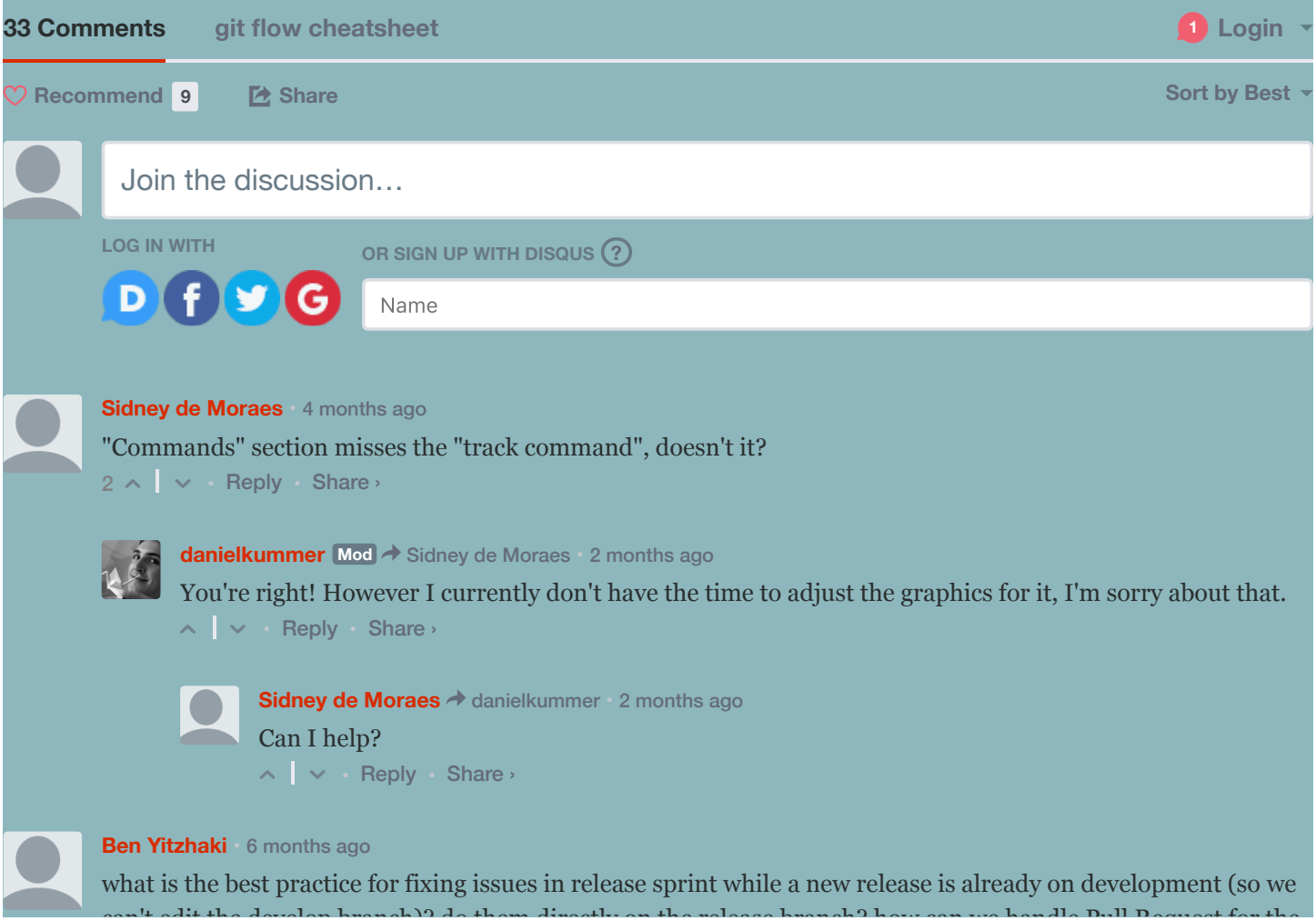
Backlog

★ ★ ★

★ 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



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..?

2 ^ | v · Reply · Share ›

HK ➚ Ben Yitzhaki · 4 months 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 · 4 months 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 thete are still open issues on the previous release

6 ^ | v · Reply · Share ›

Julien Rougeron ➚ Ben Yitzhaki · 2 months ago

I would say you open an hotfix on the master branch to fix the bug that came out when releasing

^ | v · Reply · Share ›

Rizky Syazuli · 7 months 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.

1 ^ | v · Reply · Share ›

Vytautas · 23 days 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 ›

Carlos · 25 days ago

Thanks for the Cheatsheet :D

^ | v · Reply · Share ›

Armando Ibarra · 2 months ago

I love it!

^ | v · Reply · Share ›

yemiwebby · 2 months ago

This is cool. Very helpful

^ | v · Reply · Share ›

Ren Lawrence · 2 months 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.

^ | v · Reply · Share ›

danielkummer Mod ➚ Ren Lawrence · 2 months 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

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...

Reply · Share ›

Ren Lawrence ➚ danielkummer · 2 months 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.

Reply · Share ›

danielkummer Mod ➚ Ren Lawrence · 2 months ago

I see your issue - in the future I'd recommend using more feature branches and merging them back on feature completeness...

Reply · Share ›

Jepessen · 5 months 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?

Reply · Share ›

HK ➚ Jepessen · 4 months 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 Reply · Share ›

vinamelody · 7 months 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?

Reply · Share ›

Rizky Syazuli ➚ vinamelody · 7 months ago

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

Reply · Share ›

Adrien · 7 months ago

Excellent page, thank you!

Reply · Share ›

werwer · 8 months 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. :(

Reply · Share ›

Chaitanya Prabhu ➚ werwer · 7 months 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.

Reply · Share ›

Jan Stockfisch ➚ werwer · 8 months ago

while executing git flow feature try the -k option ;)
https://github.com/nvie/git...

Reply · Share ›

Jan Stockfisch ➚ Jan Stockfisch · 8 months ago

*git flow feature finish -k

Reply · Share ›

werwer ➚ Jan Stockfisch · 8 months ago

but that is not helpful. Basically using the -k option, you are merging the changes locally and just keeping the feature branch. But all you need to do is not merge it before a pull request has been approved. The -k option will simply not delete the branch. What is the point of creating a pull request after your changes have been merged to the dev branch, right ?

Reply · Share ›

Buzz ➚ werwer · 7 months ago

But you it merges with develop *locally* not remotely. After which you create a pull request on GitHub (or whatever you use). That way you know that develop can be merged with you branch without conflicts, and if your collaborator decides that they want to merge your pull request then it's a lot easier. Does that make sense?

Reply · Share ›

xtofl · 8 months 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.

Reply · Share ›

danielkummer Mod ➚ xtofl · 8 months ago

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

Reply · Share ›

Pablo Ezequiel Leone Signetti · 9 months 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

Reply · Share ›

Jan ➚ Pablo Ezequiel Leone Signetti • 8 months 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 ›

clarify • 9 months 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 • 3 years ago

A M A Z I N G !!!

^ | v • Reply • Share ›