Git Branching

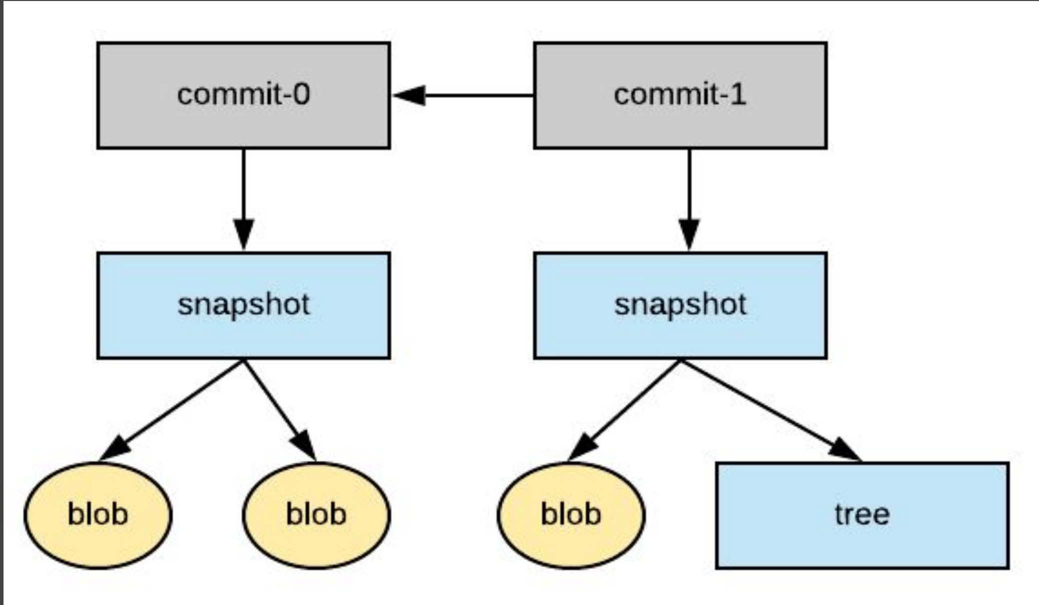
It’s killer feature

By Kostas Koyias

How does Git store data?

For each commit made, Git stores a commit object, which holds some metadata along with a set of pointers to the direct ancestors (0 or more) of this commit. Each commit object also points to the actual content staged (a snapshot).

The git repository along with any other sub-directory are all represented as trees & the files they contain as blobs.



What are branches really?

Branches are actually <u>commit pointers</u>.

Each commit is uniquely identified by a commit hash. When a branch `b` is set to point to a commit `c` , a new file is created under .git/refs/heads called `b` containing the 40 character long SHA-1 hash of `c` and a newline.

So the actual size of a branch is just 41 bytes.

Tip

Branch early & often.

Basic Operations-1

➔ **Create**

git branch <new-branch> <on-which>

git checkout -b <new-branch> <on-which>

➔ **Manipulate**

git branch **-f** <branch-name> <on-which>

➔ **Delete**

git branch **-d** <branch>

(must not contain uncommitted changes, or else force with **-D**)

Important:

Removing a branch does not mean all commits that were made on it are lost too. They still exist, but nothing points on them, like a commit made while being on detached HEAD state. All these can always be recovered with git reflog or git fsck.

Basic Operations-2

➔ **Combine**

Merge a series of end-points into one.

- Create a new commit under the branch you merge into.
- Retain commit history.

Rebase a work line into another.

git rebase <base> <local-or-checked>

- Apply your work from the nearest ancestor and on, on top of another branch.
- History seems to be linear.

git cherry-pick <c0> <c1> … <cn>
apply the changes those commits introduce on top of HEAD.

.

A popular branching model.

Simple and effective, with two types of branches.

Main Branches

Infinite lifetime.

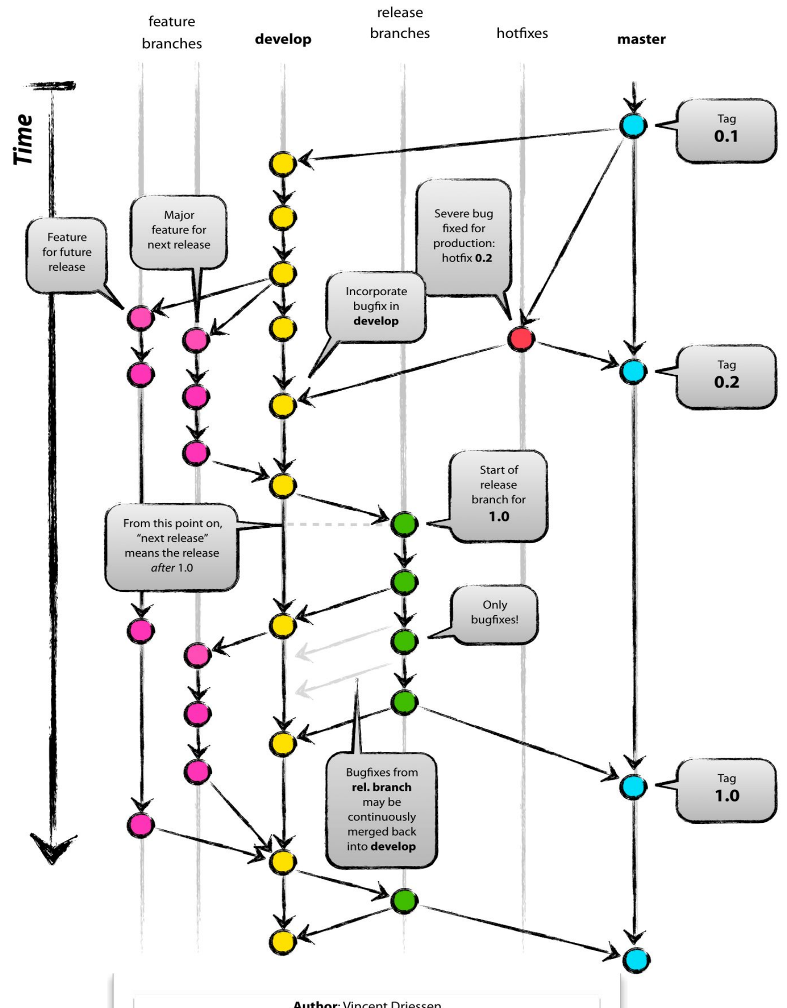
Reflect release & deployment

Supporting branches

These branches

- allow parallel feature development
- prepare for production releases
- assist in quickly fixing live production problems.

They have a <u>limited lifetime</u>, eventually they will be removed.


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

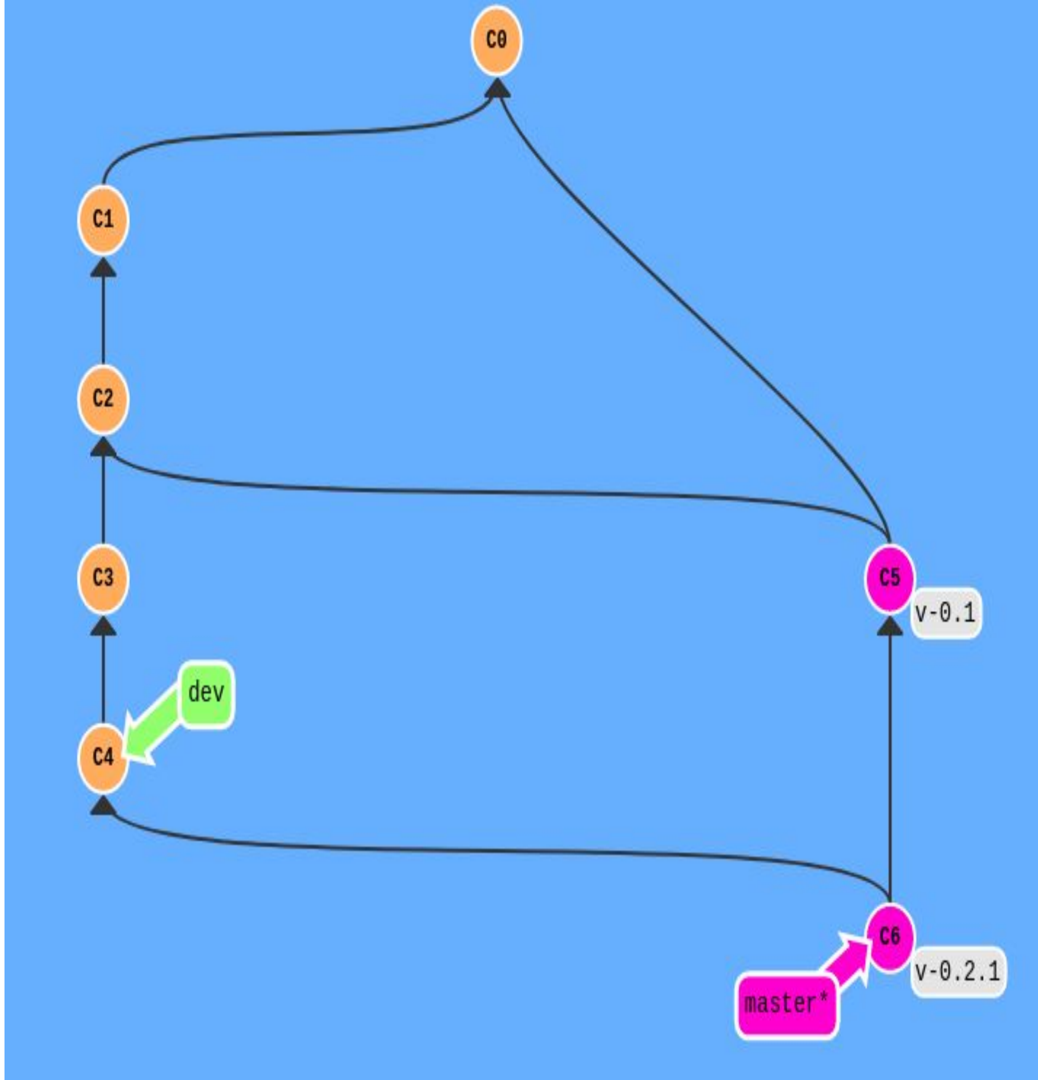
Main branches

master

- Reflects a production-ready state.
- Only merge back into master when a new version is ready.

develop

The branch on which all work is done.



Side branches-1, 2

feature

- **branch off:** develop,
- **back-into**: develop(or **discarded**)

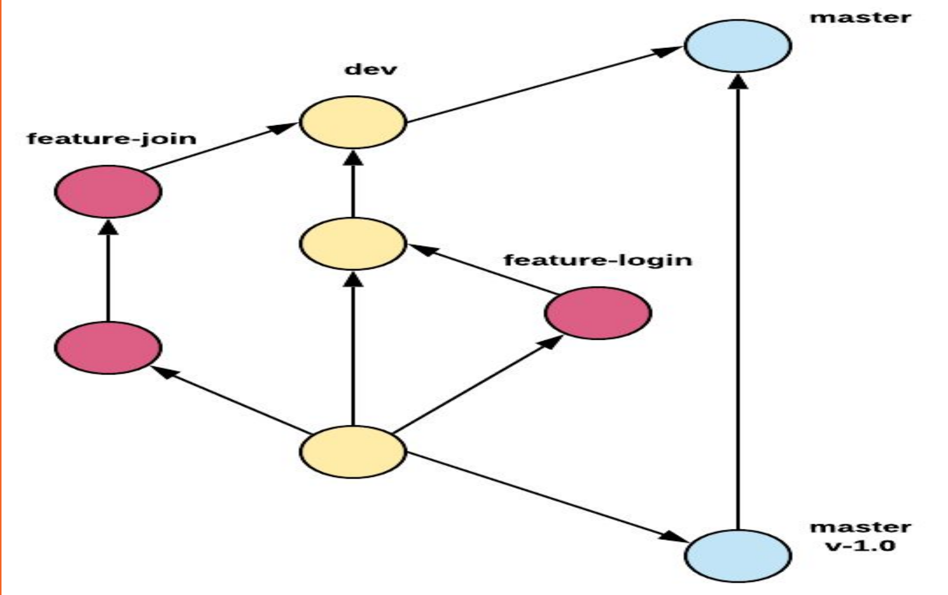
Used to develop a new feature / add some kind of new functionality.

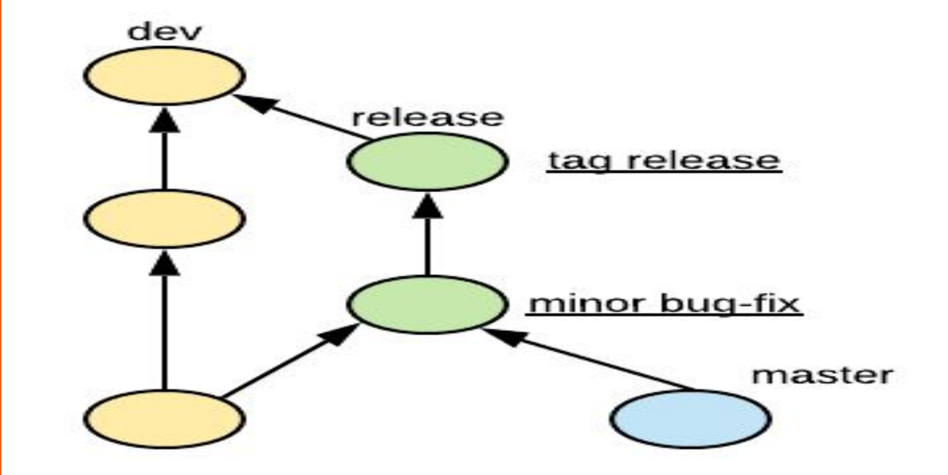
release

- **branch off:** develop,
- **back-into**: master & develop

Create when *develop* almost reflects a production-ready state. Prepare for a new production release(minor bug fixes, version number & tagging etc.)

Adding new features here is strictly forbidden.





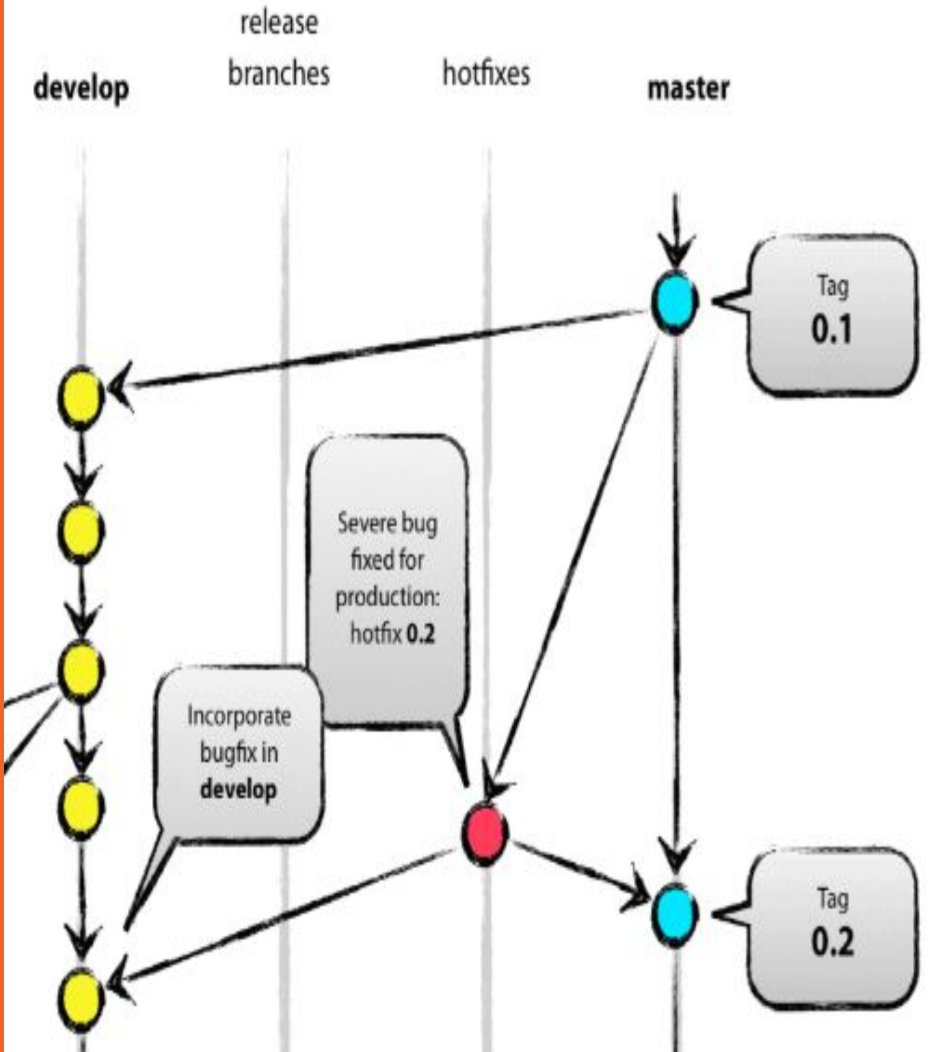
Side branches-3

hotfix

- **branch off**: master,
- **back-into**: master & (develop | release)

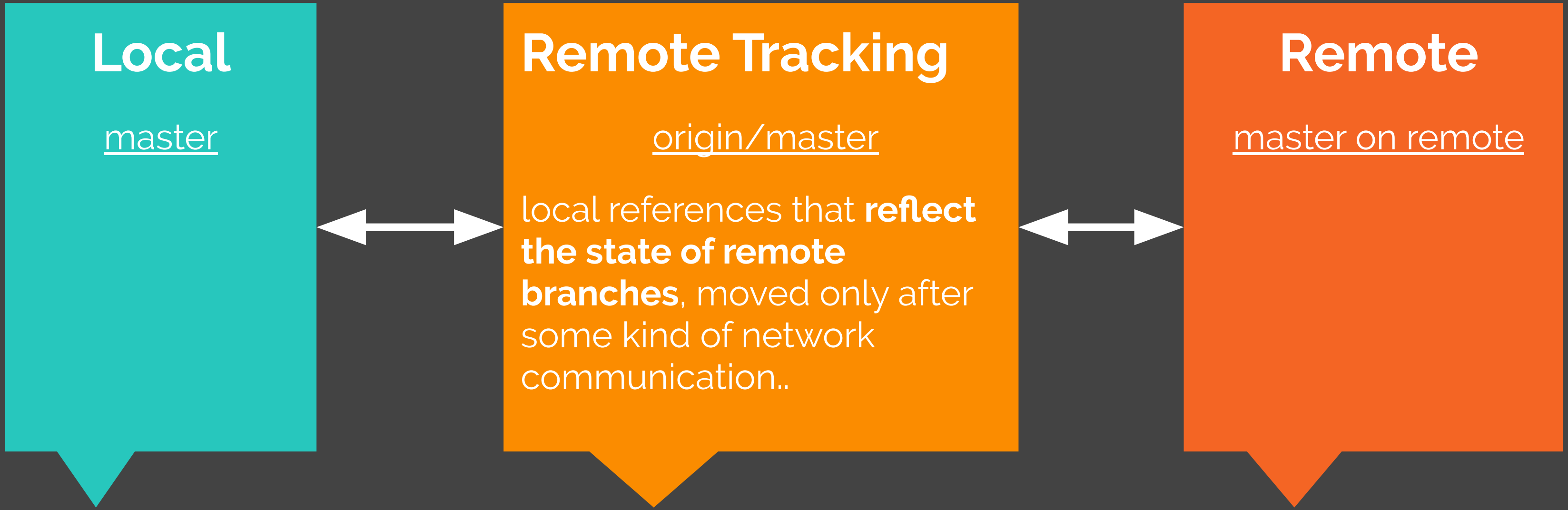
Fix an unexpected **severe bug** found in the latest release

If a release branch is active merge back into that instead of the develop(except if develop really needs those changes). Eventually, the release branch will be merged back into develop anyway.


develop
release branches
hotfixes
master
Tag 0.1
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2

Sync with remotes

We can set a merge target & a push destination for a local branch by making it track a remote-tracking one, which is actually a reflection of some branch on the remote.



Local

master

Remote Tracking

origin/master

local references that **reflect the state of remote branches**, moved only after some kind of network communication..

Remote

master on remote

Basic Operations

➔ **Track**

git checkout -b <new-branch> <on-which>

git branch -u <remote-tracking> <local>

➔ **Pull**

git pull [remote] <src> : <dest>

Omitting <src> will create a new branch, if none is tracking <dest>.

➔ **Push**

git push [remote] <src> : <dest>
Omitting <src> will delete <dest>.

If <dest> does not exist, it will be created.

References

1. A successful branching model

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

2. Pro Git

https://git-scm.com/book/en/v2

3. Learn Git Branching

https://learngitbranching.js.org/