Git

Professional software development

Nathan Segers & Martijn Loth

What is git?

Linus Torvald

How it started

Git

- Created by Linus Torvald, for working together on Linux kernels.
- Created in roughly a couple of days in 2005.
- Open sourced from the start.

List: linux-kernel
Subject: Re: Kernel SCM saga..
From: Linus Torvalds <torvalds () osdl ! org>
Date: 2005-04-07 15:10:21
Message-ID: Pine.LNX.4.58.0504070747580.28951 () ppc970 ! osdl ! org
[Download RAW message or body]

On Thu, 7 Apr 2005, Paul Mackerras wrote:
>
> Are you happy with processing patches + descriptions, one per mail?

Yes. That's going to be my interim, I was just hoping that with 2.6.12-rc2 out the door, and us in a "calming down" period, I could afford to not even do that for a while.

The real problem with the email thing is that it ends up piling up: what BK did in this respect was that anythign that piled up in a BK repository ended up still being there, and a single "bk pull" got it anyway - so if somebody got ignored because I was busy with something else, it didn't add any overhead. The queue didn't get "congested".

And that's a big thing. It comes from the "Linus pulls" model where people just told me that they were ready, instead of the "everybody pushes to Linus" model, where the destination gets congested at times.

So I do not want the "send Linus email patches" (whether mboxes or a single patch per email) to be a very long-term strategy. We can handle it for a while (in particular, I'm counting on it working up to the real release of 2.6.12, since we _should_ be in the calm period for the next month anyway), but it doesn't work in the long run.

> Do you have it automated to the point where processing emailed patches
> involves little more overhead than doing a bk pull?

It's more overhead, but not a lot. Especially nice numbered sequences like Andrew sends (where I don't have to manually try to get the dependencies right by trying to figure them out and hope I'm right, but instead just sort by Subject: line) is not a lot of overhead. I can process a hundred emails almost as easily as one, as long as I trust the maintainer (which, when it's used as a BK replacement, I obviously do).

However, the SCM's I've looked at make this hard. One of the things (the main thing, in fact) I've been working at is to make that process really _efficient_. If it takes half a minute to apply a patch and remember the changeset boundary etc (and quite frankly, that's _fast_ for most SCM's around for a project the size of Linux), then a series of 250 emails (which is not unheard of at all when I sync with Andrew, for example) takes two hours. If one of the patches in the middle doesn't apply, things are bad bad bad.

Now, BK wasn't a speed deamon either (actually, compared to everything else, BK _is_ a speed deamon, often by one or two orders of magnitude), and took about 10-15 seconds per email when I merged with Andrew. HOWEVER, with BK that wasn't as big of an issue, since the BK<->BK merges were so easy, so I never had the slow email merges with any of the other main developers. So a patch-application-based SCM "merger" actually would need to be _faster_ than BK is. Which is really really really hard.

So I'm writing some scripts to try to track things a whole lot faster. Initial indications are that I should be able to do it almost as quickly as I can just apply the patch, but quite frankly, I'm at most half done, and if I hit a snag maybe that's not true at all. Anyway, the reason I can do it quickly is that my scripts will _not_ be an SCM, they'll be a very specific "log Linus' state" kind of thing. That will make the linear patch merge a lot more time-efficient, and thus possible.

https://marc.info/?l=linux-kernel&m=111288700902396

> If so, then your mailbox (or patch queue) becomes a natural
> serialization point for the changes, and the need for a tool that can
> handle a complex graph of changes is much reduced.

How it's going

Git is a way to keep track of code.

From your computer to a remote server.

The basic workings of git.

Git

Basics

- 🌳 What is git?
- 🏷️ What is a branch?
- 🧩 What is a commit?

- ⚠️ **Never ever use git in Google Drive, OneDrive, iCloud Drive, etc.!**

🌳 What is git?

Git fundamentals

- 🤳 Snapshots, not differences
- 🏠 Nearly every operation is local
- 🔐 Git has integrity
- 📈 Git generally only adds data

- https://git-scm.com/book/en/v2/Getting-Started-What-is-Git?

What is a branch?

Git fundamentals

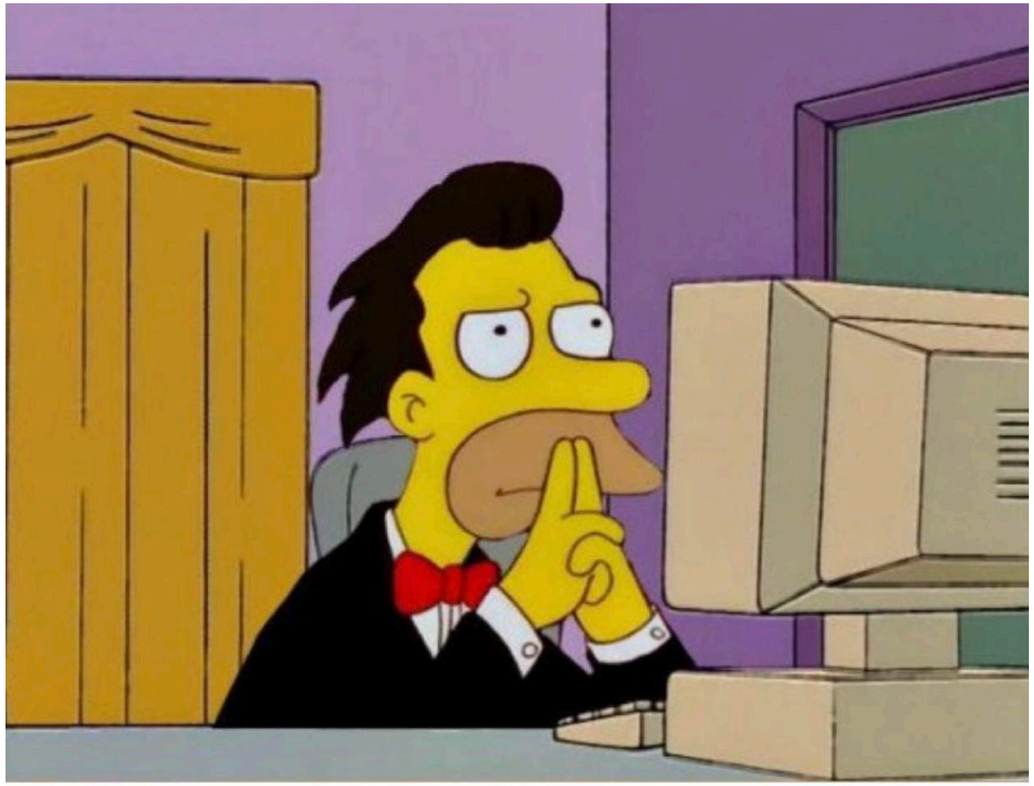
- A place to safely diverge from the codebase.
- An automatic timeline that keeps track of changes (develop, main, etc.).

When your coworker asks you which git branch you're currently working on

🧩 What is a commit? - I

Git fundamentals

- The single most important aspect of git (imho).
- The (little) piece of code that is being added to the whole.
- Commit often and in chunks that belong together.
  (I know this is vague...)
- Use a good descriptive name for each commit↷.



```
% git add .
% git commit -m '
```

🧩 What is a commit? - II

Git fundamentals

- I love to use prefixes to add some context to commits:
  - **feat**: a new feature for the user, not a new feature for a build script
  - **fix**: bug fix for the user, not a fix to a build scripts
  - **refactor**: refactoring production code
  - **chore**: updating config-files etc.; *no production code change*
  - **docs**: changes to documentation
  - **perf**: code improved in terms of processing performance
  - **vendor**: update version for dependencies, packages
  - **test**: adding missing tests, refactoring tests; *no production code change*

Emoji	Purpose
💬	Generic message
📐	Improve the format / structure of the code / fi
⚡	Improve performance
🚀	Improve something (anything)
📝	Write docs
💡	New idea
🎉	New feature
➖	Remove a feature
🔰	Tested it out!
🩹	Add new code / file(s)
🔥	Remove code / file(s)
✏️	Rename / move a file(s) / folder(s)
❓	Ask a question
🚧	Work in progress
📚	Storybook
🆕	Critical changes
🔜	When something is to be released / deployed
🔚	When something is to be disabled, not be ava support of it
🔐	When dealing with security
💪	Fix bug
🔊	Add logging
🔇	Reuce logging
🌍	Deploy
🤖	Dependabot update `yarn.lock` and `packag` with `yarn dependabot` (`git pull; yarn`)
📦	Add / update a new dependency
👐	Remove a dependency
⤴️	Upgrade dependency
⤵️	Downgrade dependenciy
⚠️	Warning (Draw attention to sth)
✅	Add tests (TDD)

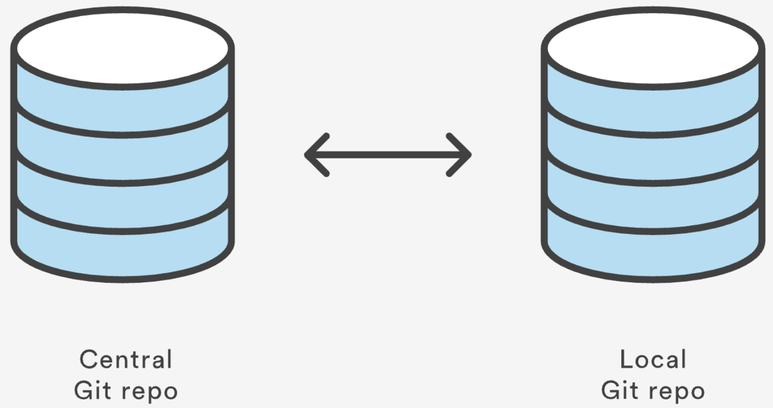
Basics: working with git

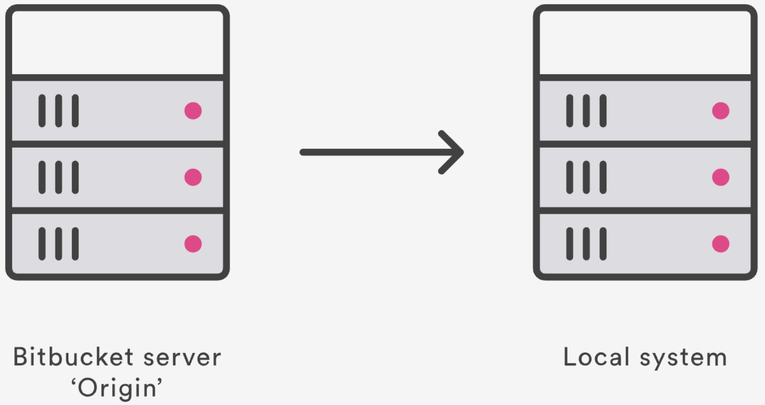
Just you

We are working on code (on our own).

We want to save our code to a provider. (eg. GitHub) in the cloud.

```
git init
git clone
git remote add ...
```





Basics: some changes

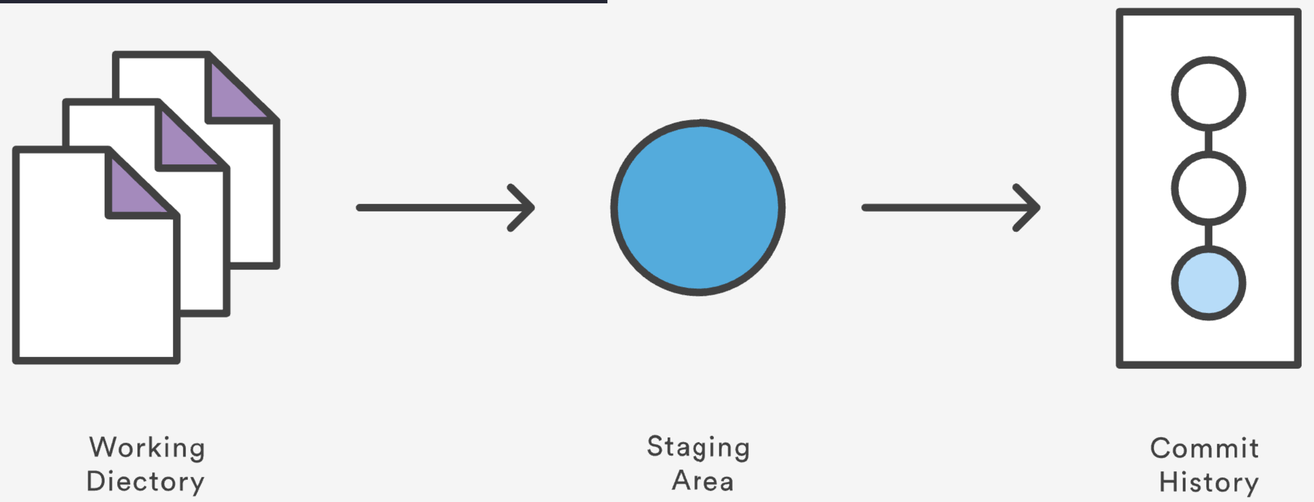
Changing stuff

We do our first changes in the code.

What do we do with these changes?

```
git add .
git commit -m "fix: removed bug with database"
```

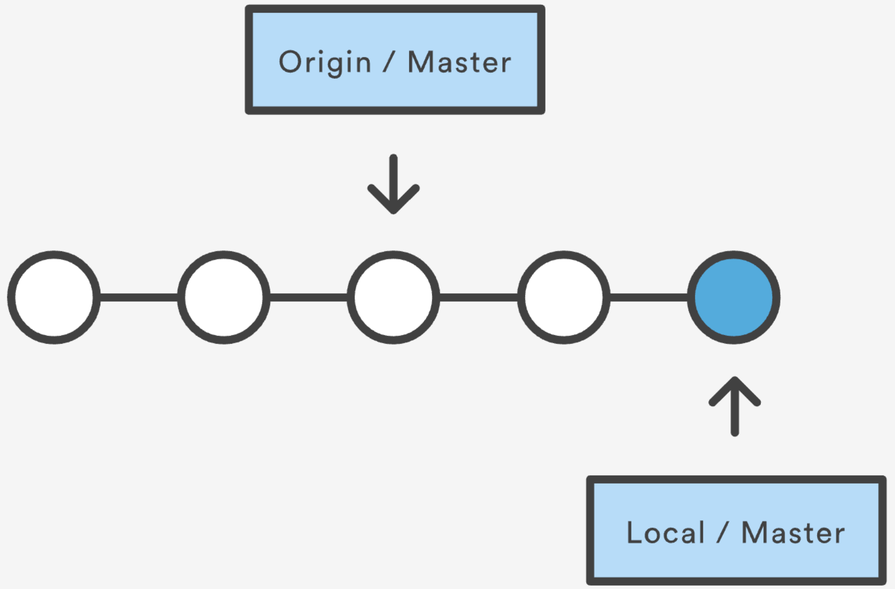
```
git push -u origin
```



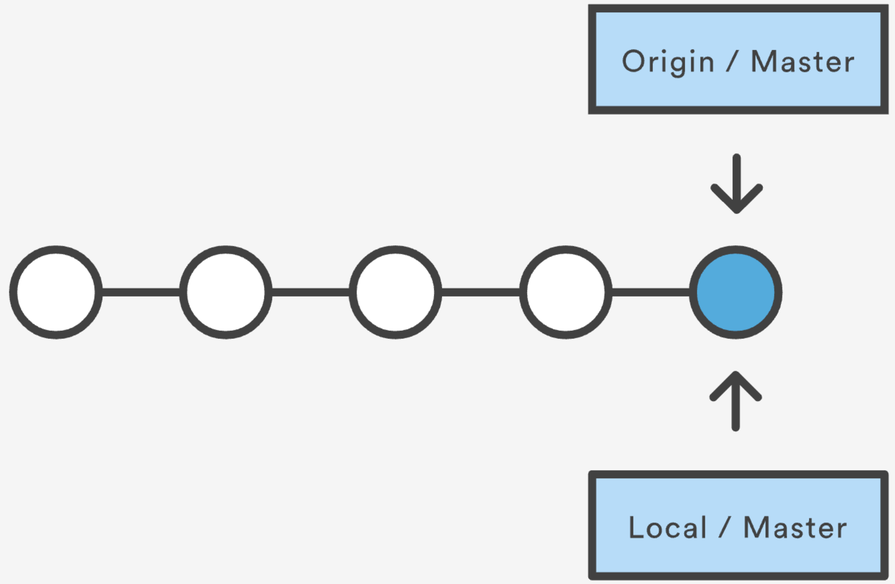
Basics: pushing to remote

Sigh of relief!

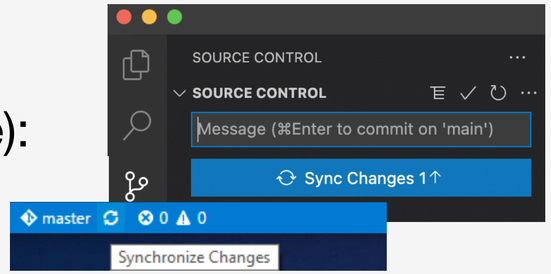
- Your local code will be ahead of the remote code: you'll have to merge.



- How to merge safely?
    1. **Commit** your code (don't push yet).
    2. **Pull the external branch**.
       At this point an automatic merge can happen.
    3. **Push** to the remote.



- Or: **git sync** (in VS Code):



Basic git commands.

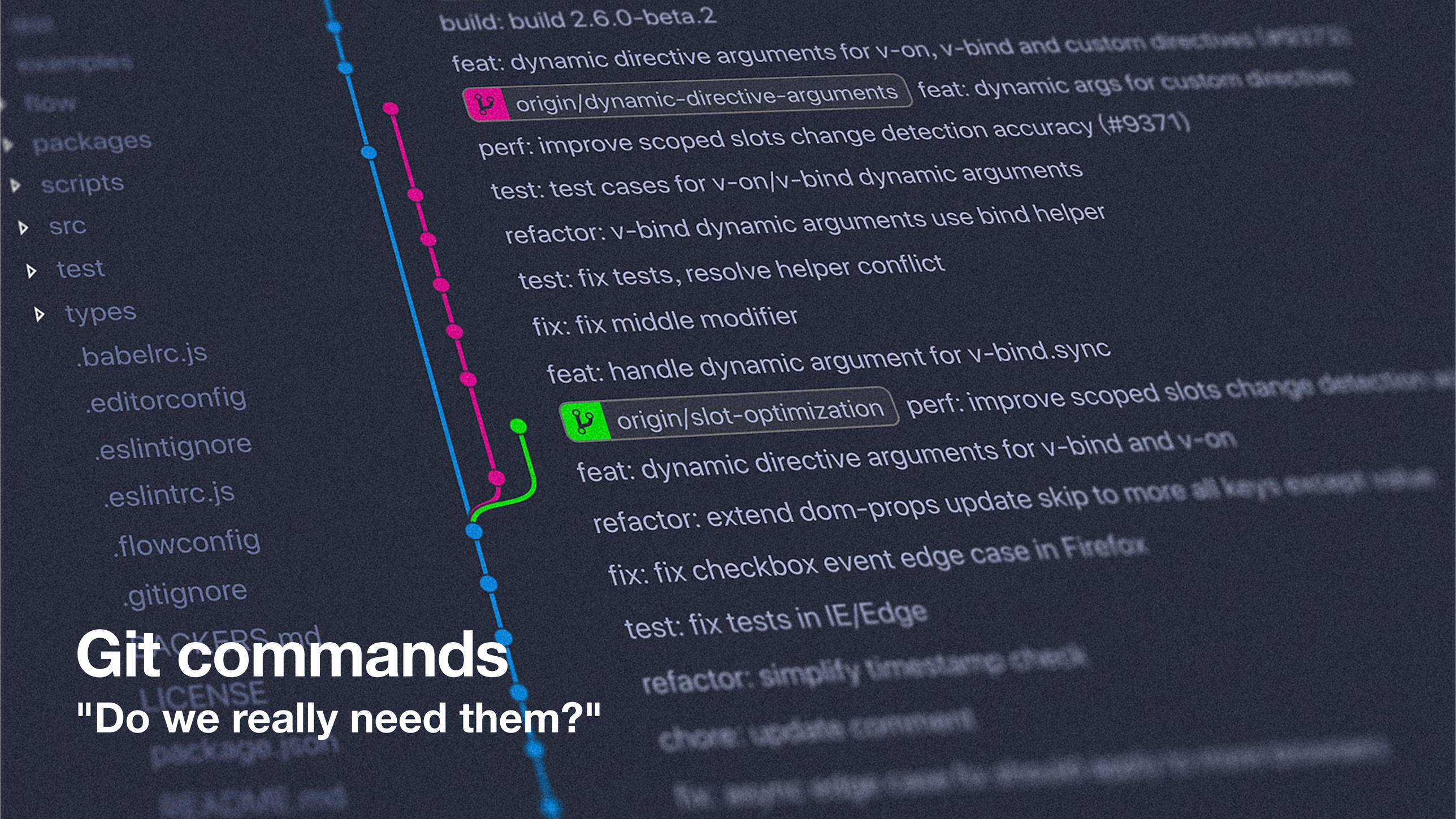
Git commands

That are common

`git init`	Initialize a repository
`git clone <url>`	Clone a repository
`git status`	Check repo status
`git add .`	Stage all changes
`git commit -m "message"`	Commit changes
`git push origin main`	Push changes
`git pull origin main`	Pull latest changes
`git branch new-branch`	Create a branch
`git checkout new-branch`	Switch to a branch
`git merge new-branch`	Merge a branch

🧐 Optional

git fetch: getting all remote branches without updating



Git commands

"Do we really need them?"

Gitflow: working in a team with git.

Working in a team

On a shared codebase.

- Often the project is set up so each push to the main-branch triggers a production pipeline.
  - Code can be build, tested and deployed.
- We need a set of agreements to work efficiently: this can be done with gitflow.

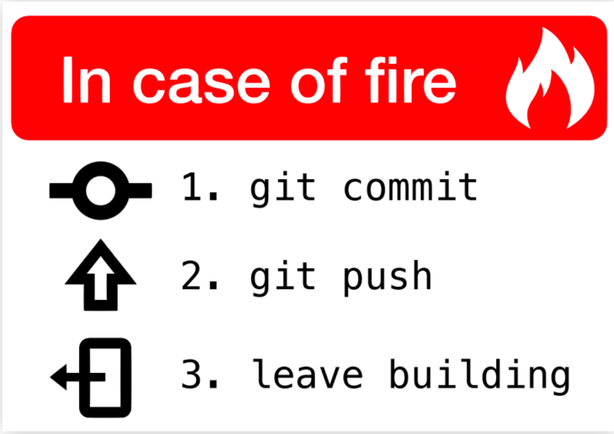
- Otherwise, you'll get: ...


GIT MERGE

Gitflow

A workflow for git.

- We want to **get software quickly to end users**.
- **Only defines branches**, commits remain the same (!):
  - Use logical descriptions for commits
  - Avoid too large commits
  - Commit as often as possible!
- Use a good .**gitignore**-file!
  https://gitignore.io/



✌🏼 Work with two branches

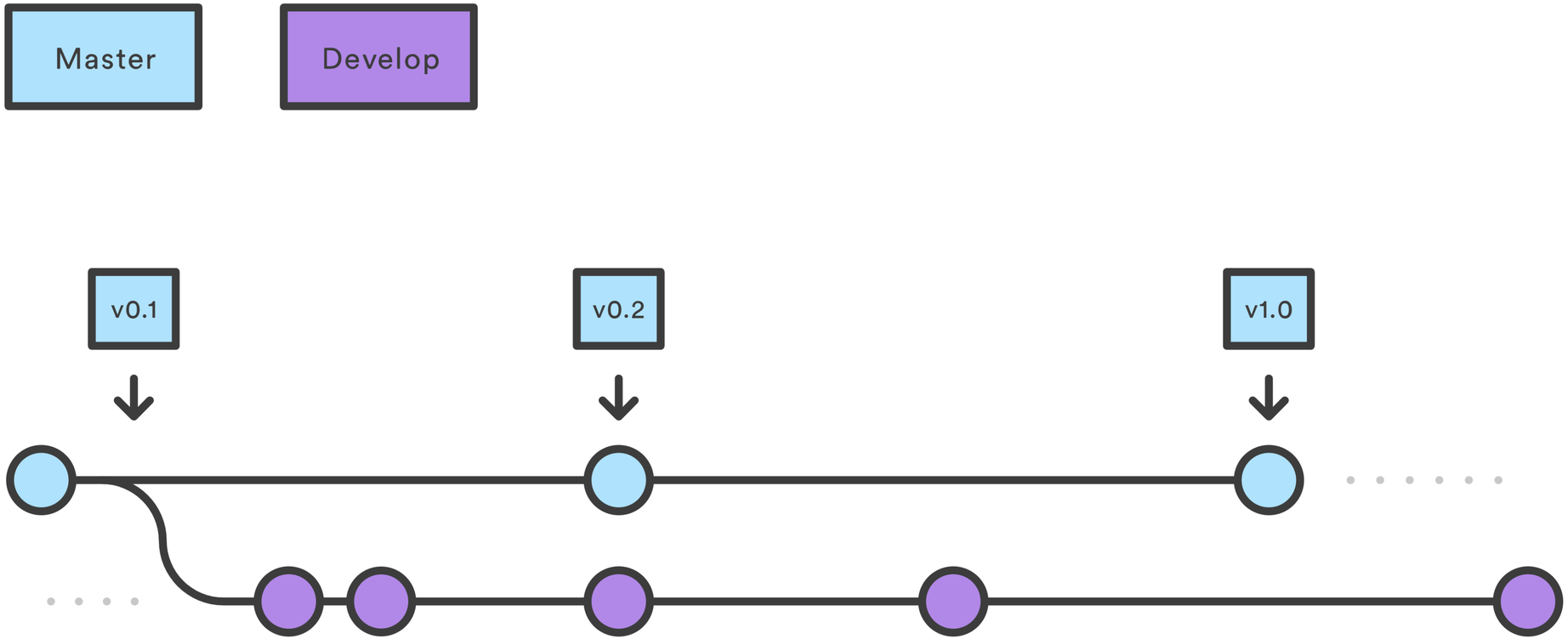
Main & develop

- **Master / main branch:**
  - Everything that is finished:
    - the end user can see it,
    - this can be released.
  - Get a version for each release.

- **Develop branch:**
- All work-in-progress
  - Features ready to go to production.
- This has to be finished: we need to be able to go to production.
- Code we don't want to release yet.

✌🏼 Work with two branches

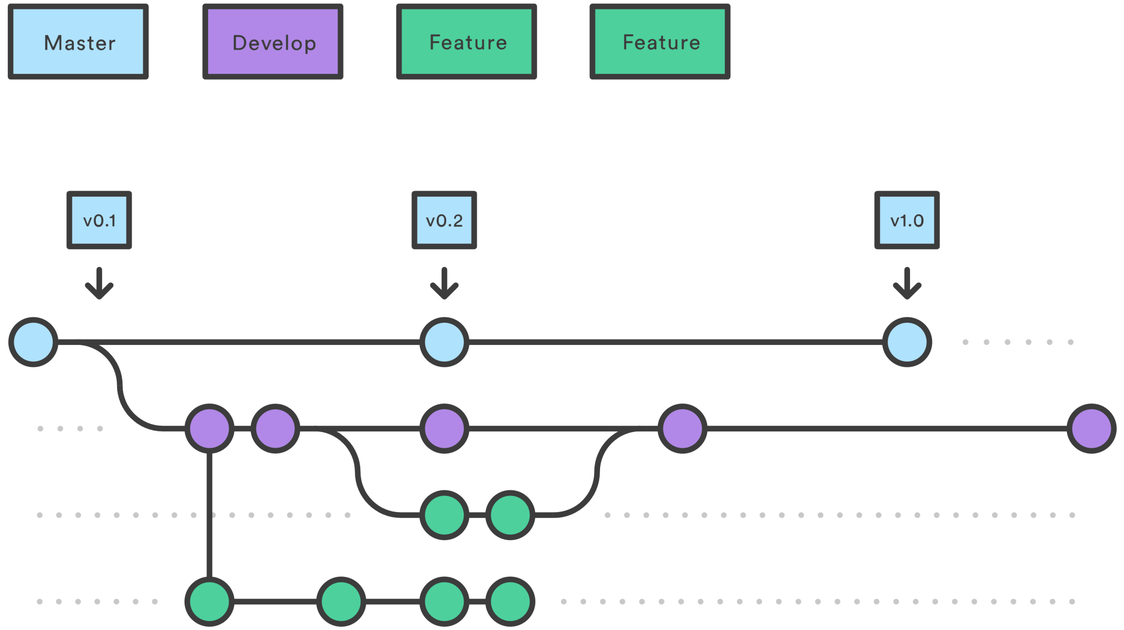
Main & develop



👷 But... we need to be able to work on our own

Getting things done.

- Each developer needs to be able to work on code without messing someone else's code up.
- This happens in a feature-branch:
  - Creates a local branch with your current work, starting from develop.
  - Your changes of files and packages and so on are in this branch.
- When a feature is finished[1], we can finish it and merge it into the develop-branch.
- PS: a feature starts from the develop-branch.

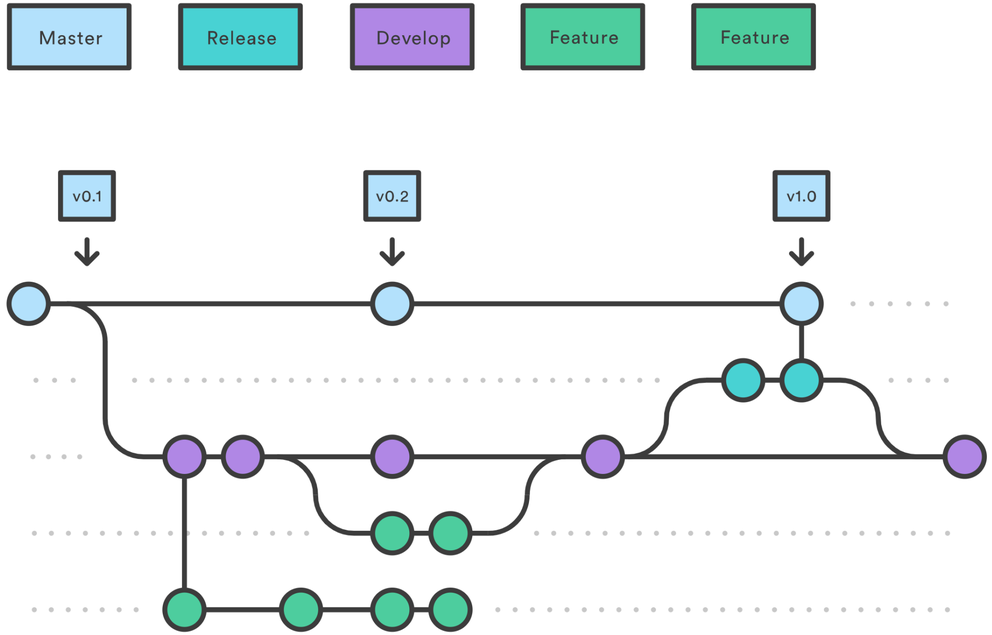


1. The *definition-of-done* depends on your team (agile work).

👌 Develop has enough features for a new release

Good job, team

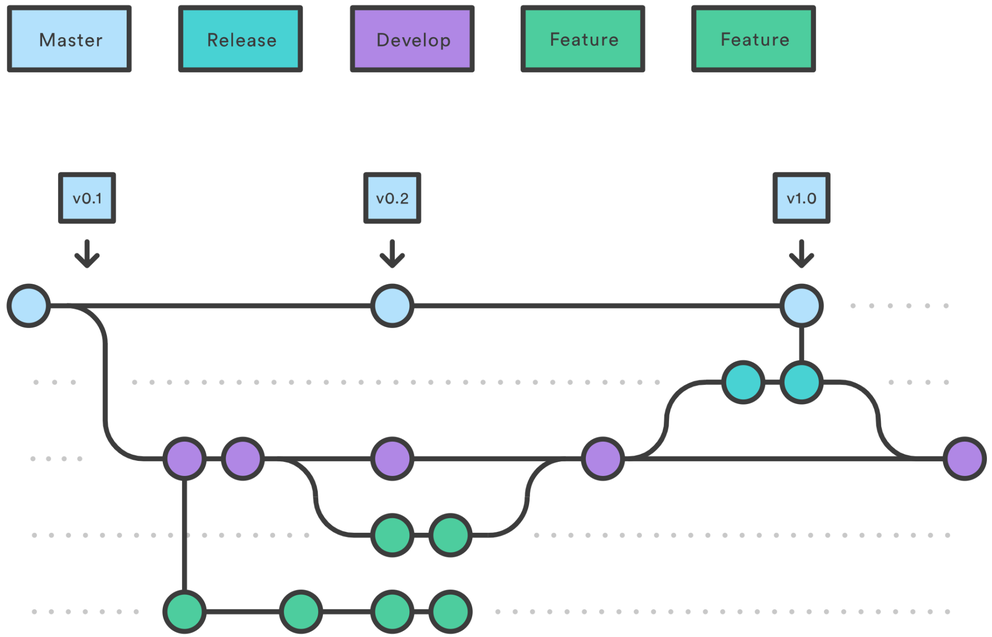
- A fork from the develop-branch becomes a release-branch.
  Here go commits of:
  - Last minute bug-fixes for the release.
  - Documentation
  - Versions, updates, etc.
- Don't add new stuff! Use another feature-branch is you need to.



🎉 The release-branch is ready!

Getting it out there

- The release-branch get merged with:
  - Develop:
    we want every last minute change also applied to the current work-in-progress (develop).
  - Main:
    this triggers a new build and deployment (CI/CD)!
  - This merge will also need a tag: eg.: 0.0.1.

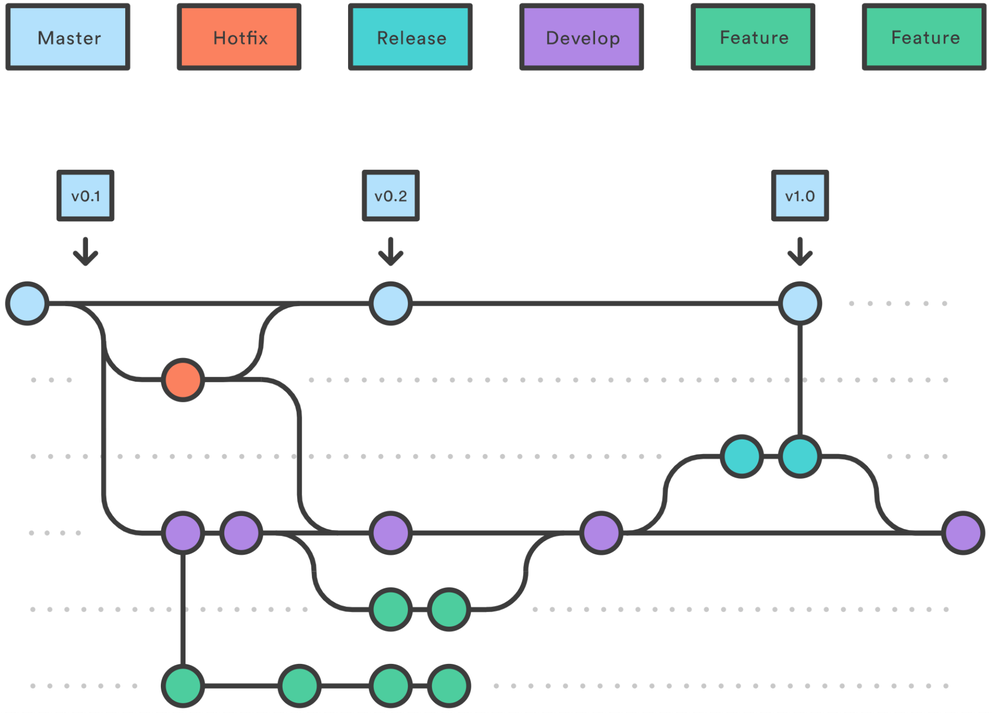


1. Good version guide: https://semver.org/.

🐞 And... there is a bug

Something always goes wrong

- We cry for a short moment 😭, then we:
  - Create a hotfix-branch (from the main).
    (The only fork from the main.)
  - Fix the bug and merge it into main[1] & develop.
- This allows us to do quick updates without:
  - having to wait for a release-branch,
  - breaking the flow (and code) of team members.

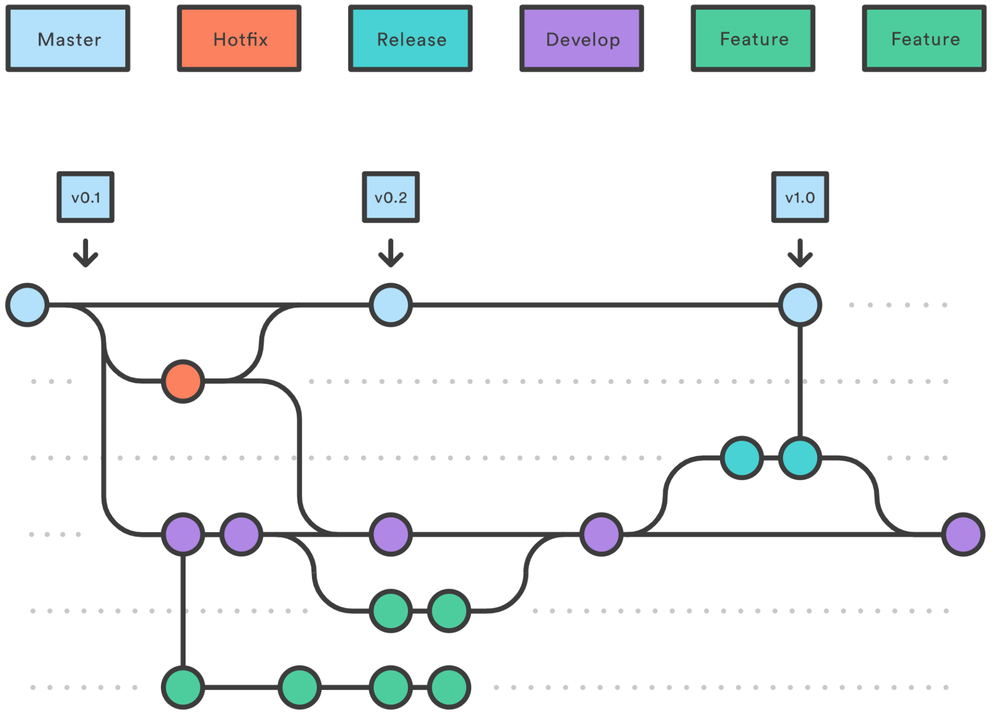


1. Provide a good tag, it is another merge with main.

🚀 Gitflow: create software collaborative

Recap

- **main** → Changes here go live, always provide a tag.
- **hotfix** → Quick bug fixes that go to main & develop.
- **release** → Last touches before release.
- **develop** → The things our team is currently working on (current sprint).
- **feature** → Each function we are working on, can be just of or with multiple developers.



Gitflow in practice

Two options

1. With git in the command line
   There are cli-tools for gitflow:
   🍏 → brew install git-flow
   🖼️ → https://git-scm.com/download/win

2. Or with tools ↷

Tools

Free or paid

- With interfaces voor git:
- 🦑 GitKraken
  https://www.gitkraken.com/git-client (student license available)

  Good tutorial: https://www.youtube.com/watch?v=eTOgjQ9o4vQ (3 min)

- 🍴 Fork
  https://git-fork.com/ free alternative

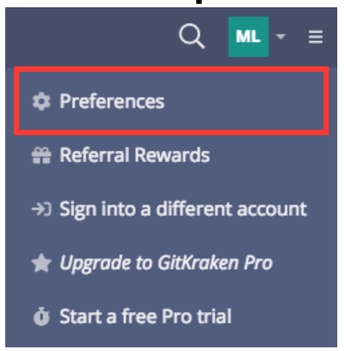
Gitflow setup

Before you start

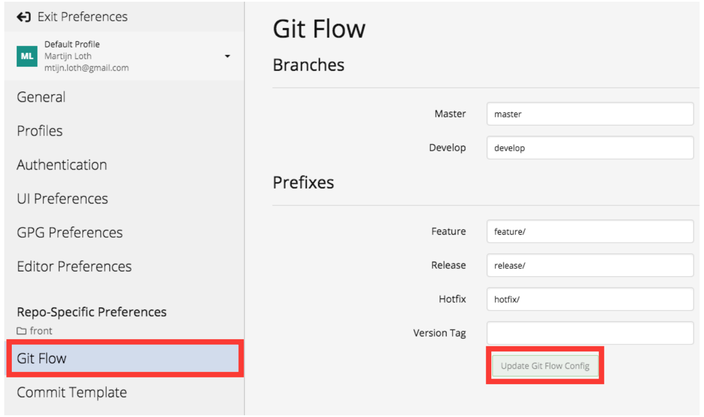
```
git flow init
```

or

- Go to preferences



- Choose:
'Initialize Git Flow'



Feature

Work

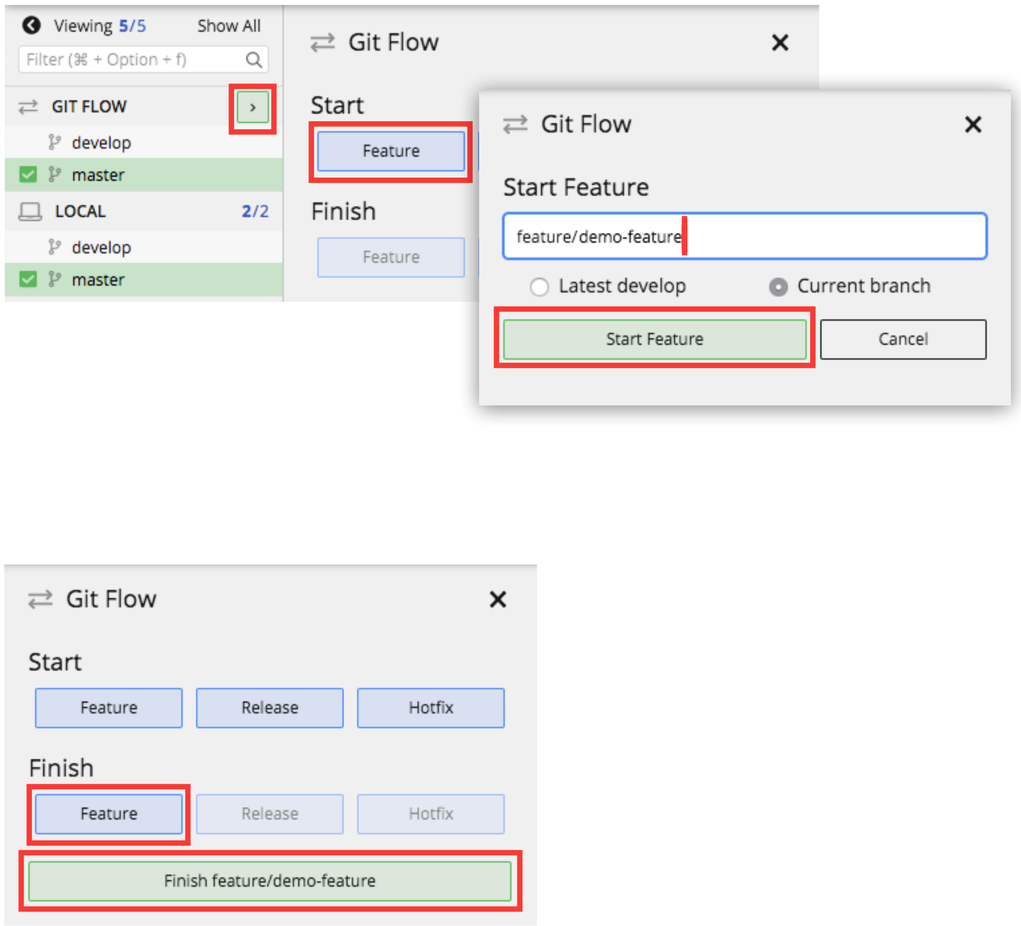
- Start of a feature:

```
git flow feature start feature
```

- Commit all your current work.
- Finish a feature by:

```
git flow feature finish feature
```

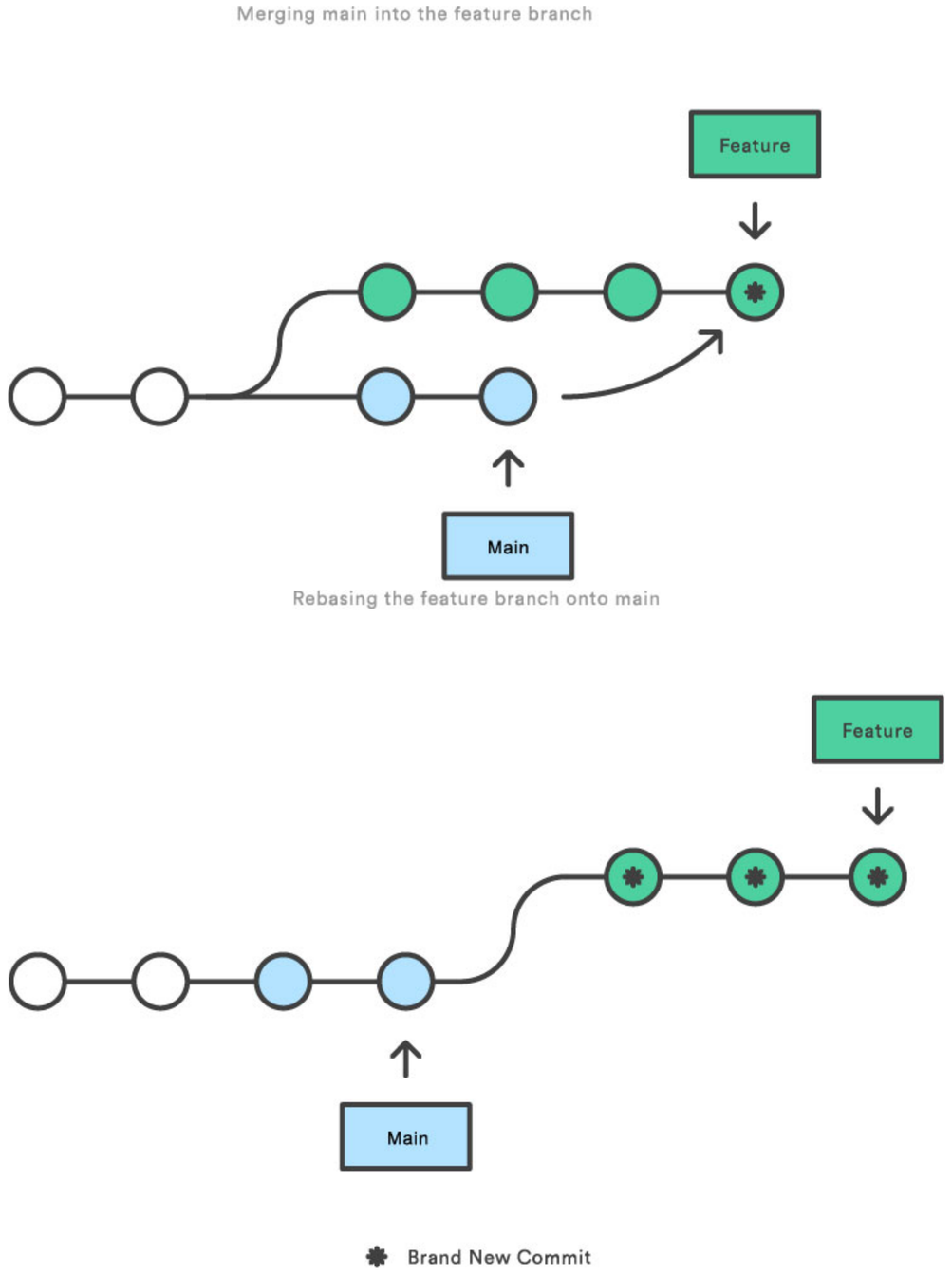
or



Feature

Picking up where we left off

- Sometimes we need to continue work on an 'old' feature.
- You can make sure a branch matches develop again (or sometimes even main):
  - **Merge**: main → feature/whatever.
    Simple, more ugly way.
  - **Rebase**: feature 'becomes' main (copies everything).
    Complexer, more prone to errors.



Release

Finish

- Starten of a release

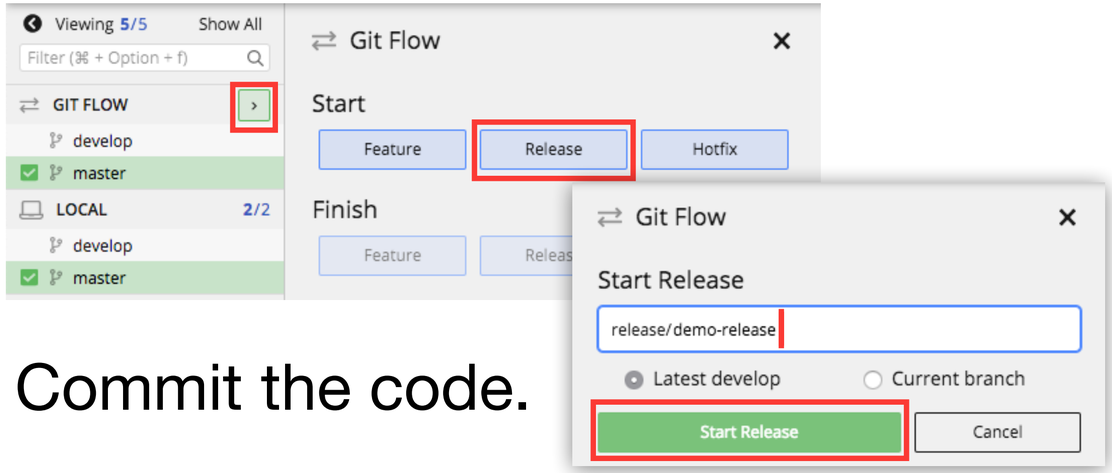
```
git flow release start 0.1.0
Switched to a new branch
'release/0.1.0'
```

- Commit the code.
- Finish a release

```
git flow release finish '0.1.0'
```

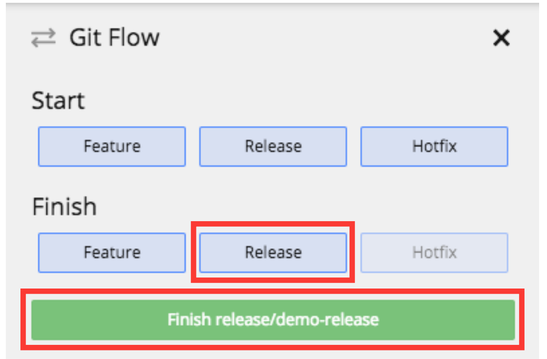
or

- Start a release



Commit the code.

- Finish a release



Hotfix

Just don't create bugs

- Start of a hotfix

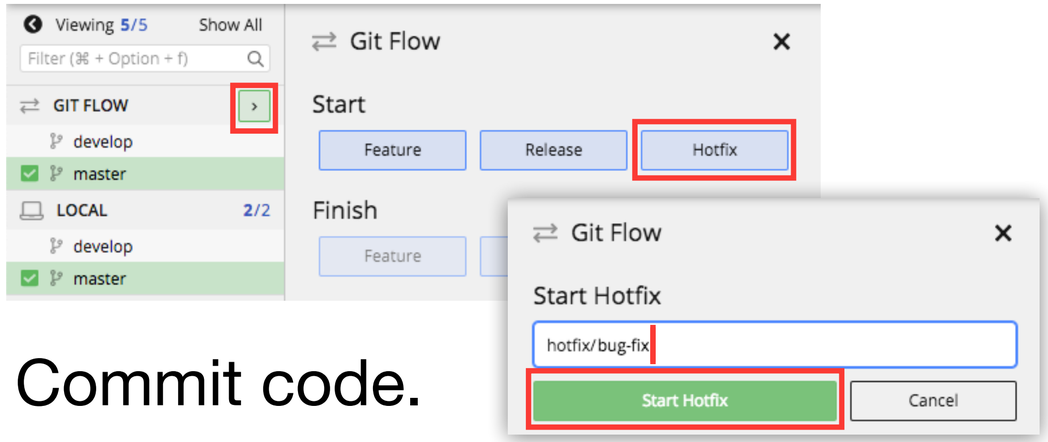
```
git flow hotfix start
hotfix_branch
```

- Commit all your work.
- Finish a hotfix:

```
git flow hotfix finish
hotfix_branch
```

or

- Starten of a hotfix



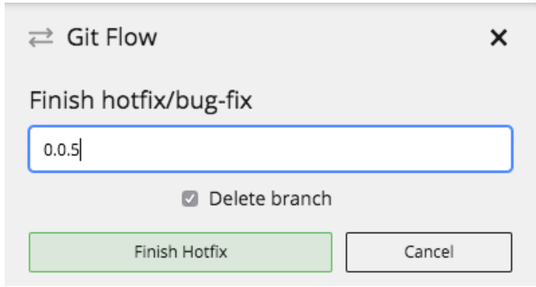
Commit code.

- Finish a feature

Each merge with main

Give a tag

- Asks for a good tag



Recap

Keep in mind

- Gitflow is a branching model around a release pipeline.
- "You build it, you release it." mentality.
- Make sure to create great commits!
- Always use a .gitignore-file.
- Git flow is easier with tools: a CLI or with a UI.

- You'll only really learn this when you do it.

Left out

GitHub: we will see the GitHub integration in the lab

🧠 Exercise

An API

#1 Merging with git

Using CLI & merge it

- Pick a partner for this assignment.
- Check the source files on Notion.

Automatic merge failed; fix conflicts and then commit the result.

#2 We'll create a backend API

Using git & gitflow.

- Pick a partner for this assignment.
- Create a GitHub repository.
- Install GitKraken, **use your student account** for login.
- Have a look at the code provided.
- Start in GitHub, follow your teachers.

Recap

How to work with git?

- Git enables us to smoothly work on code and keep versions.
- Git works in essence with branches & commits.
- Git only adds data as snapshots, works mostly locally and checks integrity.
- You can use prefixes to add context to commits.
- Merging happens locally when you pull another branch (remote or local).
- Gitflow is an approach to use git in a team.

Done.