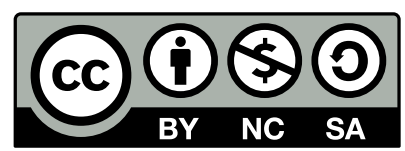
Git voor beginners

Ridan Vandenbergh Gilles Coremans

ULYSSIS

24 November 2021



Installatie

Windows `https://www.git-scm.com` en Git Bash

Mac `https://www.git-scm.com` en Terminal

Linux Package manager en uw favoriete terminal

- `git config --global user.email "email@example.org"`
- `git config --global user.name "Uw Naam"`
- `git config --global core.editor nano`
- `git config --global init.defaultBranch master`
- `git config --global core.pager "less -RXF"`
- `git config --global core.autocrlf false`

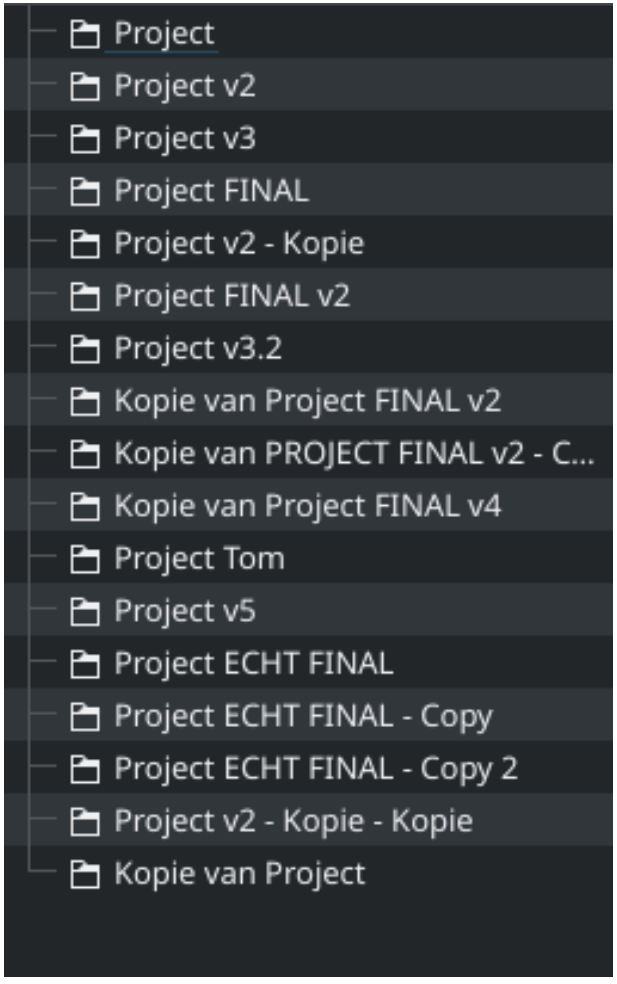
Wat is git?

- Versiecontrolesysteem
- Door Linus Torvalds (auteur van Linux!)
- Vanwege onvrede met bestaande systemen:
  - Git is sneller
  - Git is gedistribueerd
  - Git is heel krachtig
- NIET GitLab of GitHub!

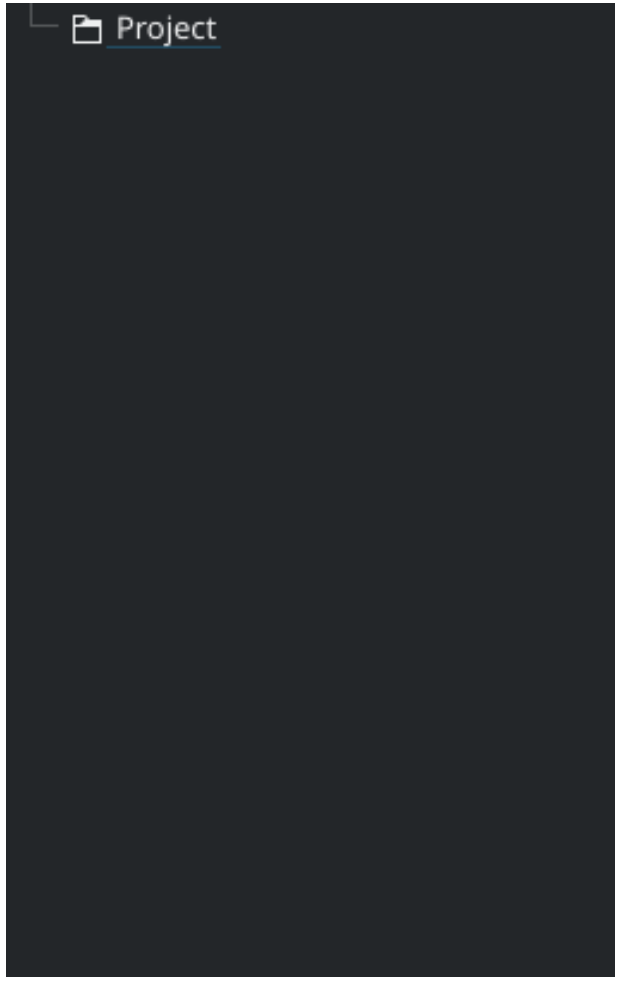
Waarom git?

ULYSSIS

Zonder git:



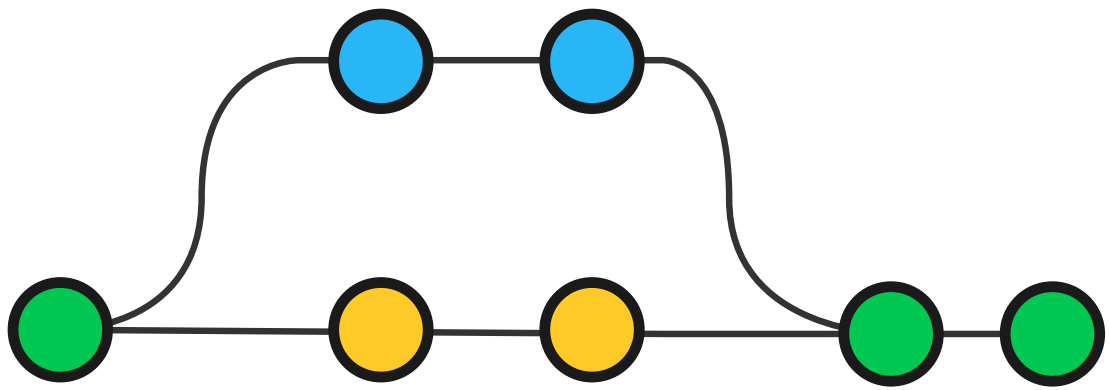
Met git:



Naamgeving

ULYSSIS

- Een project in git is een *repository*
- Veranderingen groeperen in een *commit*
- Commits volgen elkaar op in een boom
- Een tak van de boom is een *branch*
- Een branch kan op elk punt ontstaan of samenkomen



Naamgeving

Een file in een repository kan zich in vier verschillende "staten" bevinden:

Untracked Git volgt deze file (nog) niet

Staged De veranderingen in de file zijn klaar voor de *volgende* versie (commit)

Unmodified De file is hetzelfde als in de laatste versie (als in `HEAD`)

Modified De file is veranderd sinds de laatste versie

- `git status` toont in welke staat elke file zit!

Een eerste commit

- Voor we beginnen: enkele basiscommando's
  - `ls`, `cd`, `mkdir`, `nano <file>`

Een eerste commit

- Voor we beginnen: enkele basiscommando's
  - `ls`, `cd`, `mkdir`, `nano <file>`
- Repository maken: `git init`
  - Nieuwe lege repository
  - Enkel lokaal

Een eerste commit

- Voor we beginnen: enkele basiscommando's
  - `ls`, `cd`, `mkdir`, `nano <file>`
- Repository maken: `git init`
  - Nieuwe lege repository
  - Enkel lokaal
- `git add <file>`: Files toevoegen aan de staging area (volgende commit)

Een eerste commit

ULYSSIS

- Voor we beginnen: enkele basiscommando's
  - `ls`, `cd`, `mkdir`, `nano <file>`
- Repository maken: `git init`
  - Nieuwe lege repository
  - Enkel lokaal
- `git add <file>`: Files toevoegen aan de staging area (volgende commit)
- `git commit`: Alle veranderingen in de staging area groeperen in nieuwe *commit* (versie) die volgt op `HEAD` (vorige versie)

Wat hebben we gedaan?

- `git log`: Alle commits tonen
- `git show`: Veranderingen in laatste commit bekijken

`git add`

- Ook mappen als argument
- `git add -u`: Voeg elk **tracked** bestand toe aan de staging area.
- `git restore --staged <file>`: Haal `<file>` uit de staging area
- `.gitignore`:
  - Bestand, meestal in de bovenste map van de repository
  - Vertelt git welke bestanden en mappen te negeren
  - Ondersteunt `*` als wildcard

`git commit`

ULYSSIS

- `git commit -m "Commit message"`: Voeg inline een commit message toe
- `git commit -a`: Voeg alle **tracked** files toe aan de staging area, en maak dan een nieuwe commit
- `git commit -a -m "Commit message"`: Combinatie van `-a` en `-m`!

Commits bekijken

ULYSSIS

- `git show <commit>`: Details en veranderingen van één commit
- `git log <commit>`: Geschiedenis van de commit
  - `git log --oneline`: Eén regel per commit
- `git diff`: Veranderingen tussen staging area en modified bestanden
  - `git diff --staged`: Veranderingen tussen staging area en `HEAD`
  - `git diff <commit1> <commit2>`: Veranderingen in `<commit2>` ten opzichte van `<commit1>`

Pauze

Remotes

- Tot nu toe: alles lokaal
- Waar is remote?
  - `git remote add origin <url>`: Handmatig URL toevoegen
  - `git clone <url>`: Download een repository, git stelt remotes automatisch in

Pushen en pullen

ULYSSIS

- `git fetch`: Haal veranderingen van remote, maar voer ze niet door
- `git pull`: Veranderingen van remote ophalen en direct doorvoeren
- `git push`: Veranderingen pushen naar remote

Branches

- Branches zijn verwijzingen naar een commit (en `HEAD` wijst gewoon naar de huidige branch)
- `git branch`: Branches beheren
- `git switch <branch>`: Wissel naar een branch
- `git switch -c <branch>`: Creëer en wissel naar een nieuwe branch
- `git log --all`: Toon de geschiedenis van **alle** branches
- `git log --graph`: Toon de verbindingen tussen commits

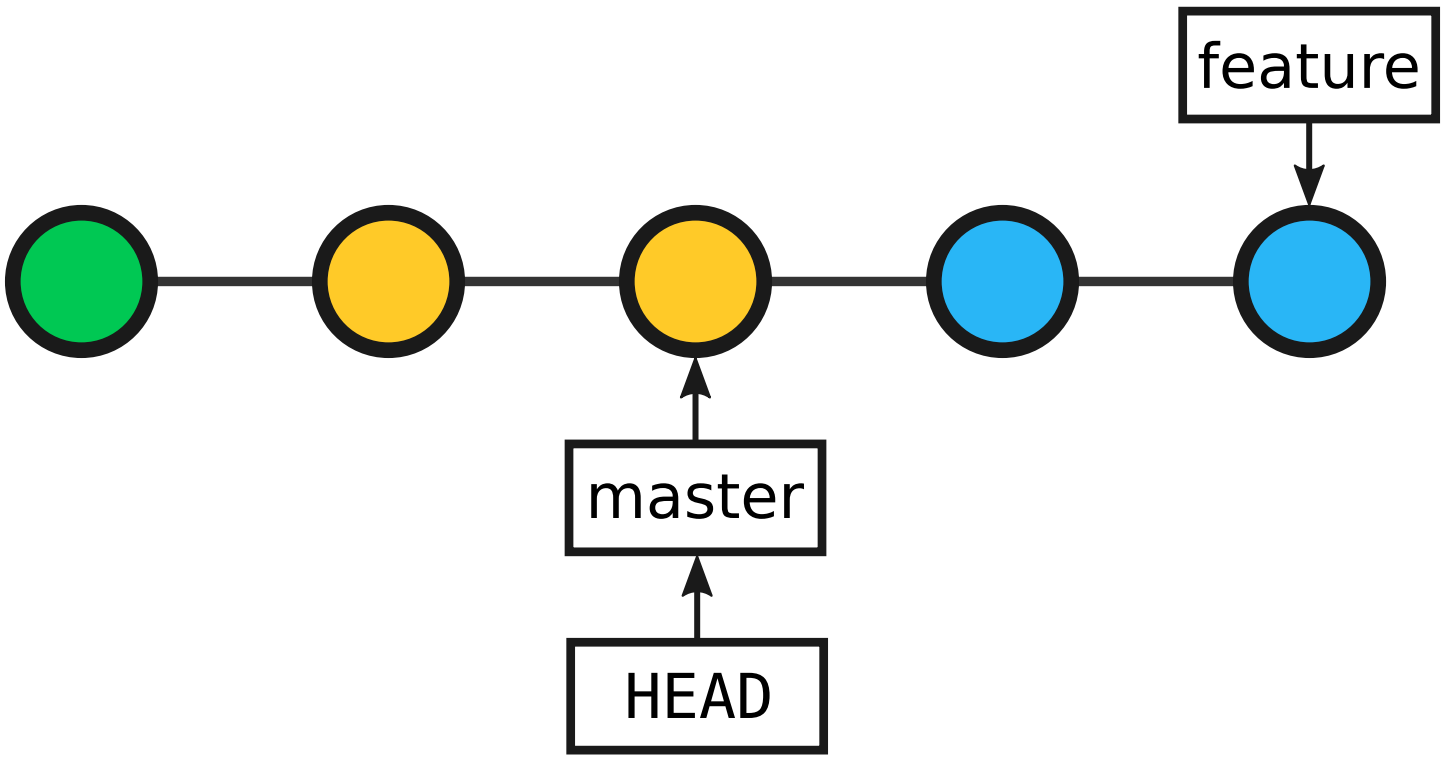
Merging

- `git merge <branch>`: Merget een branch in de huidige branch
- `<branch>` is een voorouder van de huidige branch: doe niks (we hebben alle veranderingen al!)
- `git pull` is gewoon `git fetch` gevolgd door `git merge`!

Merging: grafisch

ULYSSIS

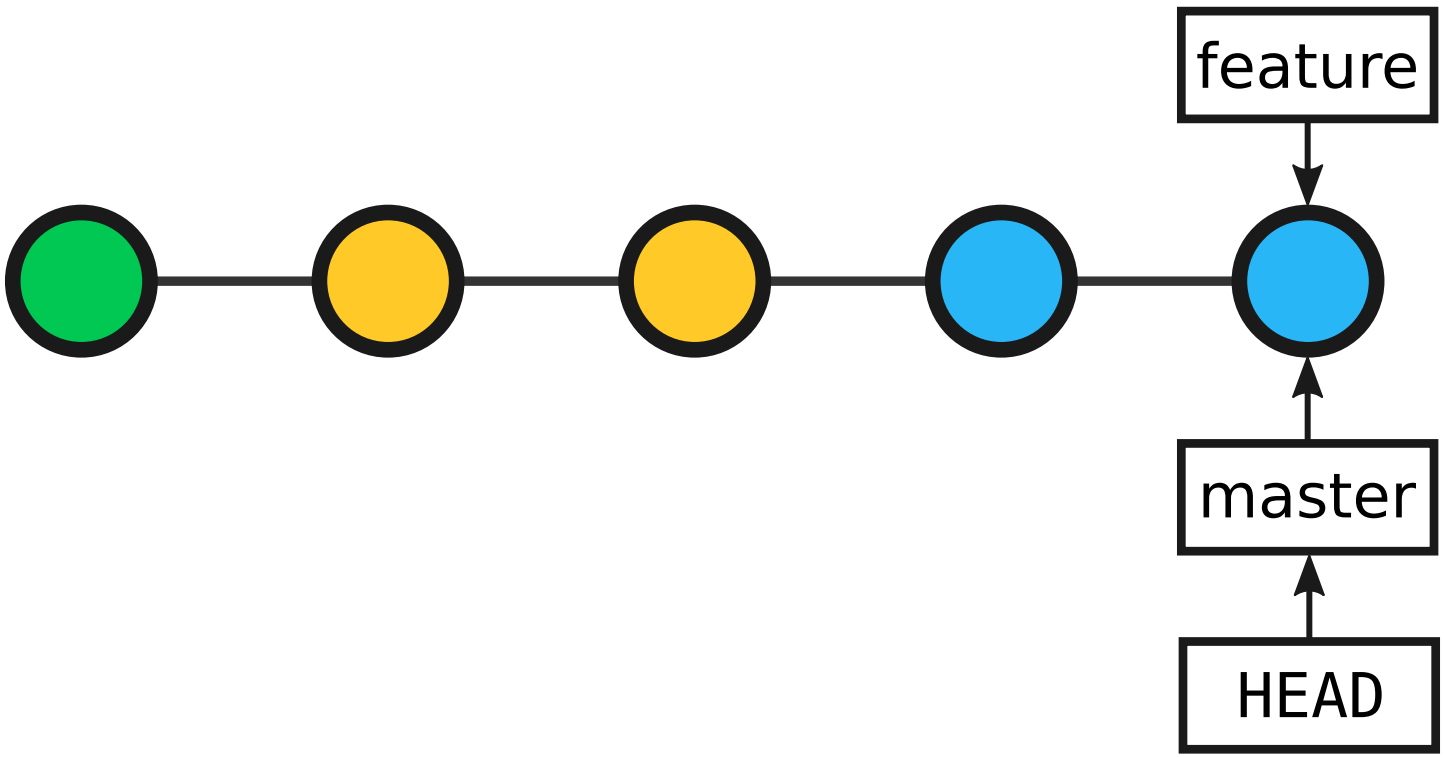
- `<branch>` ligt in verlengde van huidige branch: *fast-forward*



Merging: grafisch

ULYSSIS

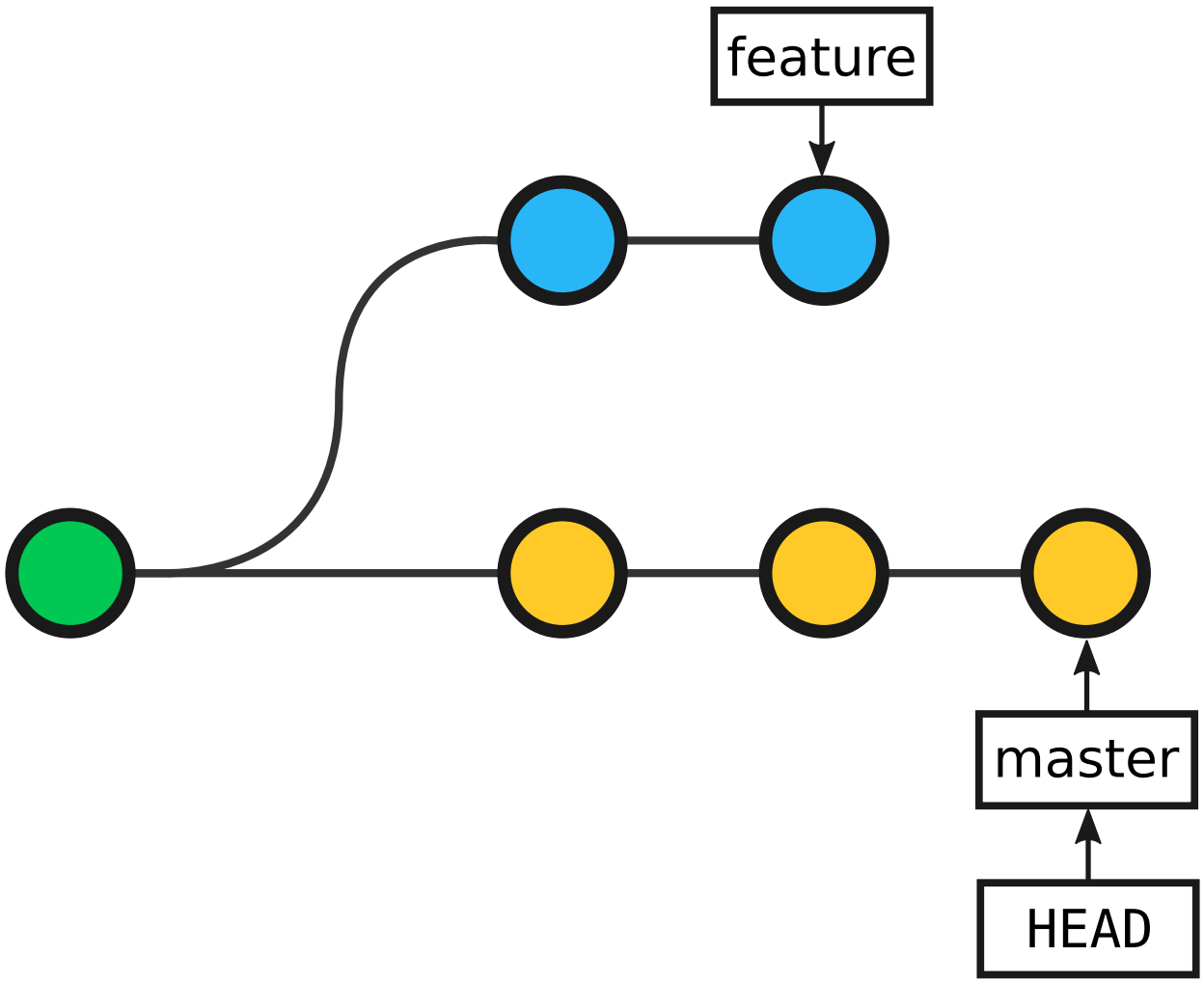
- `<branch>` ligt in verlengde van huidige branch: *fast-forward*



Merging: grafisch

ULYSSIS

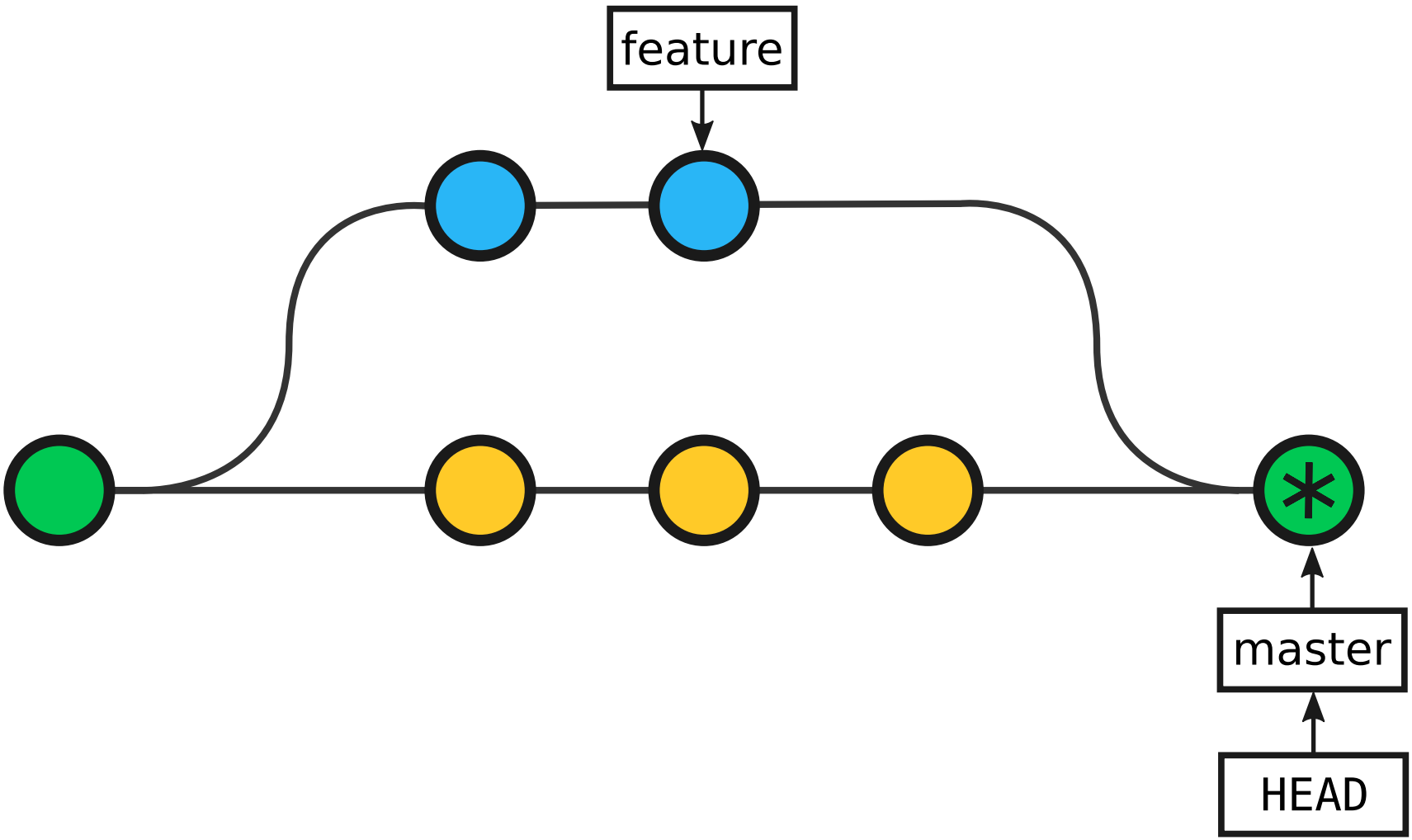
- `<branch>` is afgetakt: maak een *merge-commit* met 2 ouders



Merging: grafisch

ULYSSIS

- `<branch>` is afgetakt: maak een *merge-commit* met 2 ouders



Conflicts

- Veranderingen in hetzelfde bestand worden slim samengevoegd
- Veranderingen op zelfde regel $\rightarrow$ conflict!
- Voordat de merge gefinaliseerd wordt: conflict oplossen
- Manueel of met een merge tool: `git mergetool`
- Conflicts opgelost: `git merge --continue`
- Merge aflasten: `git merge --abort`

`git reset`

- `git reset` laat je teruggaan naar een vorige commit
- `git reset <commit>`: Update de huidige branch *en* de staging area naar de gegeven commit, maar laat de working tree met rust
- `git reset --hard <commit>`: Update de huidige branch, de staging area *en* de working tree naar de gegeven commit
- Changes uit de staging area verwijderen, teruggaan naar de huidige commit, laatste commit wissen...
- Verplaatst ook de branch pointer = geschiedenis herschrijven!
- Commit wissen = geschiedenis herschrijven!

`git stash`

ULYSSIS

- Veranderingen in je working directory tijdelijk opslaan, buiten de tree
- `git stash push`: Sla uncommitted veranderingen tijdelijk op en reset working tree
- `git stash apply`: Pas de veranderingen in de stash toe
- `git stash drop`: Verwijder een stash
- `git stash pop`: Hetzelfde als `git stash apply` + `git stash drop`

Making amends

- `git commit --amend`: Voeg changes in de staging area toe aan *vorige* commit
- **Herschrijft geschiedenis!**
- Nuttig voor foute commits die nog niet gepusht zijn

Geschiedenis herschrijven met `git rebase`

`git rebase -i HEAD~n`: Pas de laatste `n` commits aan

- `pick`: gebruik commit
- `edit`: gebruik commit, maar stop na de commit voor aanpassingen
- `squash`, `fixup`: voeg deze commit samen met de vorige commit
- Regel verwijderen: commit wissen

En verder?

- `man git`: ingebouwd handboek!
- `https://git-scm.com/book`: zeer volledig boek over Git
- StackOverflow: praktische oplossingen en uitleg