Git branches

Werken met branches

In git kun je werken in losse *'branches*'. Takken of vertakkingen dus. Dit stelt je in staat om aan verschillende delen van je code te werken zonder dat wijzigingen elkaar in de weg zitten.

Wanneer je een nieuwe git repository aanmaakt, bevind je je in de default branch die in de meeste gevallen *master* of *main* heet. Github heeft in 2021 om ethische redenen als default *main* gekozen en voor de rest van deze handleiding gebruiken we ook *main* om de hoofd-branch aan te duiden. Om een *master* branch te hernoemen naar *main* gebruik je vanuit de *master* branch:

```
git branch –m main
```

Alle wijzigingen die je commit worden toegevoegd aan de branch die je op dat moment 'uitgechecked' hebt. Vanuit een branch kun je een nieuwe branch aanmaken. De wijzigingen die je in de nieuwe branch commit bevinden zich enkel in die branch en niet in de *main* branch. Wanneer je klaar bent met het werken in een branch, kun je deze code toevoegen aan (mergen met) een andere branch.

N.B.: helemaal achteraan staat een 'cheat list' met alle commando's uit dit document.

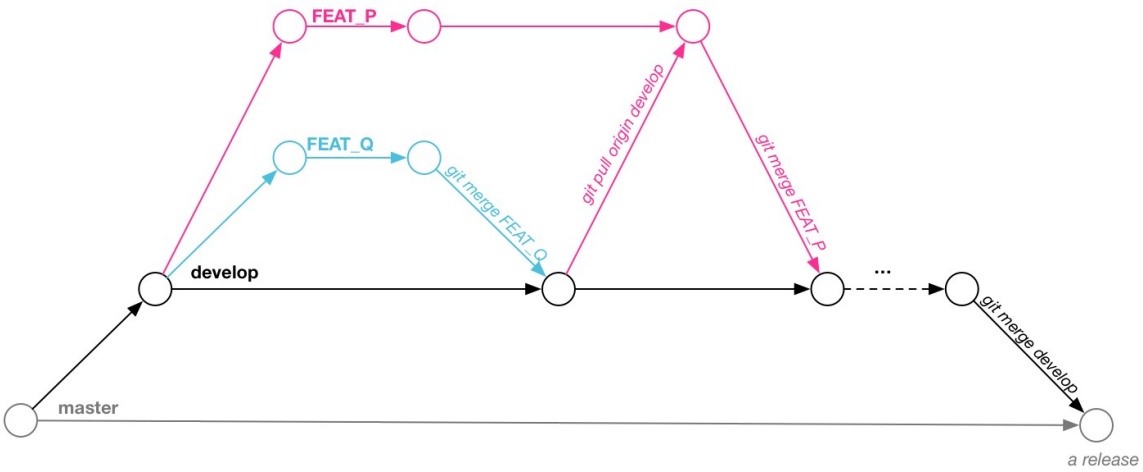
Releases, features en ontwikkeling

In projecten wordt de *main* branch vaak enkel gebruikt voor releases van het project, bijvoorbeeld om een werkende versie te kunnen testen met gebruikers. Naast het werken in de *main* branch, wordt een extra branch aangemaakt genaamd '*develop*'. Deze blijft het hele project bestaan.

Vanuit de *develop* branch worden losse branches aangemaakt. In deze losse branches worden features toegevoegd, wordt code anders gestructureerd, worden bugs gefixt… etc. Het is nuttig om een duidelijke naamgeving te hanteren voor branch-namen, zodat je de inhoud (wijzigingen /toevoegingen e.d.) goed kunt afleiden a.d.h.v. de naam van een branch.

Een development team maakt vaak afspraken over de naamgeving van branches. Zo gebeurt het toevoegen van nieuwe functionaliteiten (features) vaak in een branch genaamd *FEAT_<naam_functionaliteit>* en bijvoorbeeld een bugfix in een branch genaamd *BUGFIX_<issue_code>*.

In de onderstaande afbeelding zie je hoe er gewerkt kan worden vanuit de *develop* branch in plaats van de *main* branch.



Pull requests a.k.a merge requests en het mergen van branches

Wanneer je in een losse branch hebt gewerkt aan een afgebakende functionaliteit en wanneer dit onderdeel klaar en getest is dan ben je klaar om de wijzigingen in deze branch toe te voegen aan de *develop* branch. Dit heet **merge**. Dit kun je doen vanuit de *command line*, maar ook in de remote repository d.m.v. een zogenaamde '*merge request*' (op github heet dit een '*pull request*'). Het voordeel van het maken van een *merge request* is dat je een teamgenoot kunt vragen om online de wijzigingen eerst te reviewen, een zogenaamde *code review,* voordat jouw wijzigingen aan de bestaande code base toegevoegd worden. Het kan zijn dat je teamgenoot je naar aanleiding van een code review vraagt om nog iets aan te passen, voordat hij/zij de *merge request* wil accepteren.

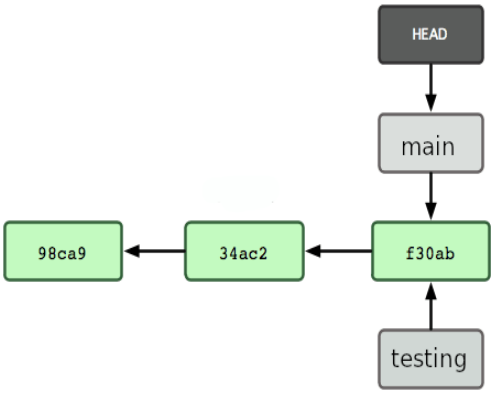
Wanneer je een *merge request* wilt aanmaken terwijl de develop branch in de tussentijd is aangepast, is het belangrijk om zelf de nieuwe code uit de develop branch in je eigen branch binnen te halen (`git pull origin develop`). Het kan namelijk zijn dat zich anders *merge conflicts* voordoen. Als er *merge conflicts* ontstaan dan kom je die al tegen bij het binnenhalen van de nieuwe code uit de *develop* branch. Omdat jij het beste op de hoogte bent van de wijzigingen in jouw branch, is het efficiënter (en sympathieker) wanneer jij de *merge conflicts* oplost, dan wanneer je dat overlaat aan je teamgenoot.

Ook in een project waarin je alleen werkt is het prettig om te werken in losse branches. Je kunt zo werken aan verschillende functionaliteiten zonder dat de afzonderlijke wijzigingen elkaar in de weg zitten. Je kunt dan tussendoor eenvoudig switchen tussen branches. Het gebruik van online *merge requests* is ook prettig in een solo-project. Dit creëert namelijk een moment waarop je kritisch naar je eigen wijzigingen kunt kijken, in plaats van aan te nemen 'dat het wel goed zal zijn'.

1. bron: https://git-scm.com/book/nl/v2/Branchen-in-Git-Branches-in-vogelvlucht

Een nieuwe branch maken

Git gebruikt HEAD om te wijzen naar de plek in de repository die gekoppeld is aan je working folder, zeg maar waar je op dat moment in de repository naar kijkt, of 'je in bevindt'. HEAD wijst dus naar een specifieke commit binnen een gegeven branch. In het plaatje hiernaast wijst HEAD naar commit f30ab binnen de branch *main*.



Een nieuwe branch wordt aangemaakt vanuit de branch waar je je op dat moment in bevindt. Zie dezelfde afbeelding, waarin vanuit de *main* branch een nieuwe branch *testing* wordt aangemaakt.

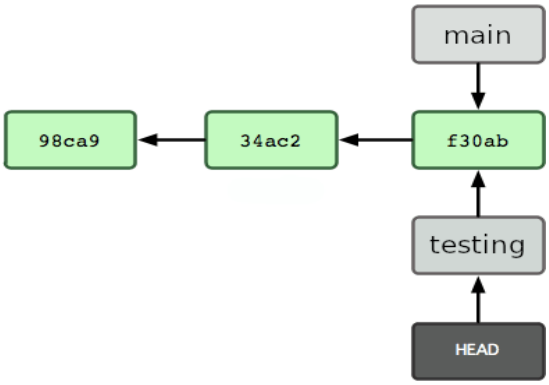
Na het aanmaken van een nieuwe branch wijst je HEAD nog naar dezelfde branch als daarvoor. Je 'bevindt' je dan dus nog niet in de nieuwe branch.

```
git branch <branch-name>
```

Een bestaande (of net gemaakte) branch uitchecken

Om de HEAD naar de nieuwe branch te laten wijzen moet je deze nieuwe branch 'uitchecken'. Zie de afbeelding[1] hieronder. HEAD wijst nu naar de nieuwe branch.

```
git checkout <branch-name>
```



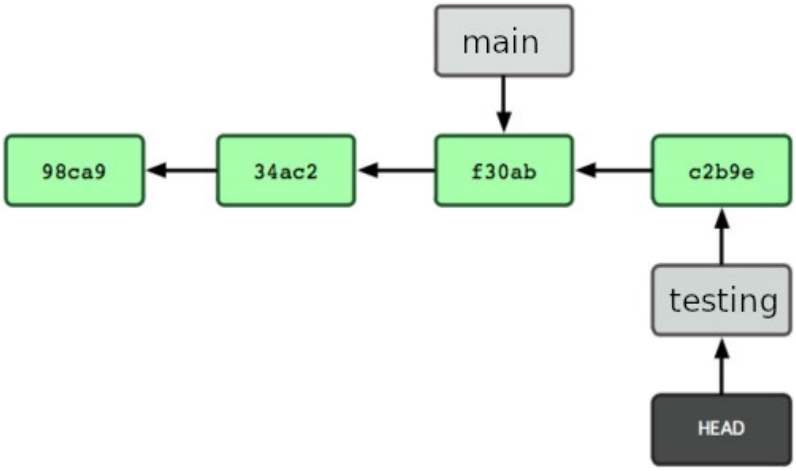
Het aanmaken van een branch en het direct uitchecken hiervan kan ook in één keer:

```
git checkout -b <branch-name>
```

1. bron: https://git-scm.com/book/nl/v2/Branchen-in-Git-Branches-in-vogelvlucht

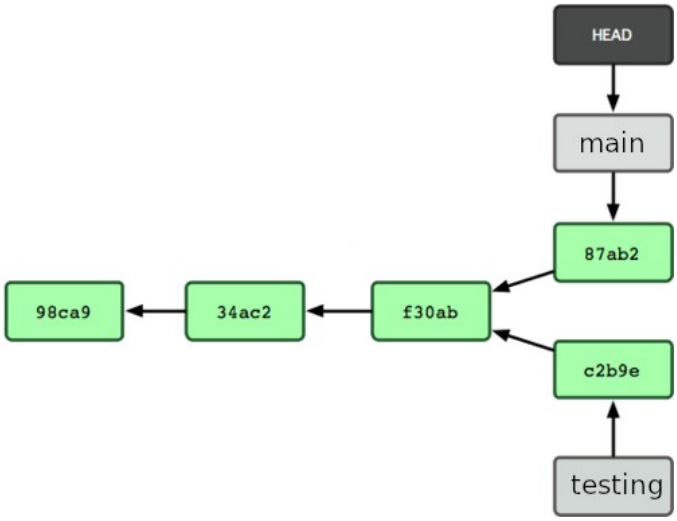
Werken in een branch

Wanneer je wijzigingen aan de code commit, gaat de HEAD mee vooruit in de huidige branch. Dit zie je in de volgende afbeelding[1] waar in de *testing* branch een nieuwe commit c2b9e is toegevoegd. Hierin loopt de tijd van links naar rechts, dus de oudste commit staat links, de nieuwste rechts.



Wanneer je daarna de main branch uitcheckt (`git checkout main`) is deze wijziging niet te zien, omdat die was uitgevoerd en gecommit in de branch *testing* maar (nog) niet in de *main* branch.

Hetzelfde geldt voor een wijziging en commit in de *main* branch: deze is niet aanwezig in de branch *testing*.



1. bron: https://git-scm.com/book/nl/v2/Branchen-in-Git-Branches-in-vogelvlucht

Branches mergen

Wanneer je klaar bent met het werken in een losse branch dan kun je je branch mergen zodat je de nieuwe features, bug fixes etc. in je *main* branch introduceert. Dit kun je online doen d.m.v. een *merge request,* of in je lokale repository kun je het *merge* commando gebruiken. Check dan eerst de branch uit waar je naartoe wilt mergen, bijvoorbeeld de *main* branch. Je bevindt je dan als het ware in de *main* branch en de ontwikkelingen in de tijdelijke branch trek je dan naar de *main* branch toe.

```
git merge <branch-name>
```

Om dit te kunnen doen moet je weten welke branches in je repository beschikbaar zijn. Dat kun je aan git vragen.

Bestaande branches tonen

Met de onderstaande commands kun je de branches tonen. Bij het tonen van lokale branches staat er een * voor de branch waar op dat moment de HEAD heen wijst.

`git branch` *toont de lokale branches.*

`git branch -r` *toont de remote branches.*

`git branch -a` *toont zowel de lokale als remote branches.*

Bestaande branches verwijderen

Wanneer je klaar bent met een branch kun je deze verwijderen. Let op: om een lokale branch te verwijderen, moet je je buiten die branch bevinden. De HEAD moet dus naar een andere branch wijzen.

`git branch -d <branch-name>` *verwijdert de lokale branch*

`git push origin --delete <branch-name>` *verwijdert de remote branch*

De onderstaande site bevat een duidelijke uitleg over git branches:
https://git-scm.com/book/nl/v2/Branchen-in-Git-Branches-in-vogelvlucht

1. bron: https://git-scm.com/book/nl/v2/Branchen-in-Git-Branches-in-vogelvlucht

Git branching cheat list

Rename a branch: git branch –m <new-branch-name>

Create new branch: git branch <branch-name>

Check it out: git checkout <branch-name>

Create & check out: git checkout –b <branch-name>

Show local branches : git branch

Show remote branches : git branch –r

Show local and remote branches : git branch –a

Merge into current branch: git merge <branch-name>

Remove local branch: git branch –d <branch-name>

Remove remote branch: git push origin --delete <branch-name>

Pull from remote branch: git pull <remote> <branch-name>

1. bron: https://git-scm.com/book/nl/v2/Branchen-in-Git-Branches-in-vogelvlucht