

# Werken met git

## Setup

We gaan eerst oefenen met het aanmaken van een nieuwe git repo.

Ga naar je terminal, navigeer naar de locatie waar je dit wilt doen (bijvoorbeeld in ~/code) en maak een nieuwe, lege directory aan met `mkdir gittest`

Vervolgens navigeer je naar die nieuwe directory, en doe je `git init`  
Je initialiseert nu een nieuwe git repository.

```
λ MacBook-Pro-van-Raoul code → mkdir gittest
λ MacBook-Pro-van-Raoul code → cd gittest
λ MacBook-Pro-van-Raoul gittest → git init
Initialized empty Git repository in /Users/rgrouls/code/gittest/.git/
λ MacBook-Pro-van-Raoul gittest → λ git main →
```

Je ziet dat je in de main branch zit.

## Add files

Nu ga je aan het werk, en je maakt een eerste file aan. Je kunt eenvoudig een lege file maken met `touch script.py`

Er verschijnt nu een nieuwe file, en git merkt dat op. Je ziet dat er een sterretje achter main staat, om aan te geven dat er wijzigingen zijn.

Je kunt de status bekijken met `git status`

```
λ MacBook-Pro-van-Raoul gittest → λ git main → touch script.py
λ MacBook-Pro-van-Raoul gittest → λ git main* → git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
   script.py

nothing added to commit but untracked files present (use "git add" to track)
λ MacBook-Pro-van-Raoul gittest → λ git main* →
```

Je ziet in de main branch, je hebt geen commits, en er is een untracked file. We besluiten dat we ons werk willen gaan toevoegen aan het versiesysteem van git. Dat doen we door `git add script.py`

Als we nu weer de status checken, zien we dat er iets veranderd is: er is een "change to be committed".

```

λ MacBook-Pro-van-Raoul gittest → λ git main* → git add script.py
λ MacBook-Pro-van-Raoul gittest → λ git main* → git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
       new file:   script.py

λ MacBook-Pro-van-Raoul gittest → λ git main* → █

```

## Commit changes

**Een commit** is het maken van een snapshot van je code. Vuistregels zijn: doe dit vaak en voor kleine veranderingen. Dus niet: 2 uur werken, en dan alles in 1x toevoegen met als commit boodschap “allerlei dingen”. Dat is te onduidelijk. Je wilt een geheugensteuntje zodat een ander (en jijzelf, over 6 maanden) nog weet wat er in die commit gebeurde.

We hebben dus een add gedaan, en gaan dit nu commiten. Dat doen we met `git commit -m "script toegevoegd"`

Het stuk tekst tussen aanhalingstekens is de message (aangegeven door de -m). Hierdoor kun je later makkelijk een overzicht zien van veranderingen. Zie het als een titel voor je werk.

Je ziet nu dat de status weer is veranderd: je working tree is clean, en er is niks meer om voor een commit over.

```

λ MacBook-Pro-van-Raoul gittest → λ git main* → git commit -m "script toegevoegd"
[main (root-commit) d1f9082] script toegevoegd
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 script.py
λ MacBook-Pro-van-Raoul gittest → λ git main → git status
On branch main
nothing to commit, working tree clean
λ MacBook-Pro-van-Raoul gittest → λ git main → █

```

Wat zijn we nu aan het doen? En waarom? Om wat beter te begrijpen wat we doen, gaan we eerst wat meer “werk” verrichten.

Dus:

- voeg een tweede commit toe met het bestand `script2.py` en de boodschap “tweede script toegevoegd”
- voeg een derde commit toe waarin je twee bestanden maakt en vervolgens toevoegt, namelijk `main.py` en `README.md` met als boodschap “main en readme”

## De log bekijken

Als je nu het commando `git log --oneline --graph` invoert, zie je als alles goed is gegaan het volgende overzicht:

```

* 870a2ae (HEAD -> main) main en readme
* 7316d30 tweede script toegevoegd
* d1f9082 script toegevoegd
(END)

```

Druk op de q om hier weg te gaan. We zien hier een aantal dingen:

- Er zijn drie snapshots gemaakt. Dit zijn drie verschillende “toestanden” van je code.
  - Elke snapshot begint met een hash-code, een unieke identifier van letters en cijfers
  - Elke snapshot heeft een korte beschrijving die je zelf hebt meegegeven
  - Je ziet een (HEAD -> main). HEAD is waar je je nu bevindt. Je bevindt je dus in de main branch.
- Je kunt ook `git log` als commando geven, dan krijg je wat meer informatie, of `git log --stat` waarmee je nog meer informatie krijgt.

## Branchen

Tot zover lijkt dit heel erg op versiebeheer van dingen als google docs. Je kunt versies opslaan, en je kunt (zoals je waarschijnlijk al vermoedt) terugkeren naar eerdere versies.

Waar git afwijkt van de standaard versiebeheer, is dat je branches kunt maken. Dit zijn parallelle aftakkingen. Stel je voor dat je nu in de main branch bent, en dat alles werkt zoals verwacht. Je wilt nu graag iets nieuws gaan toevoegen, maar wilt niet de huidige toestand tijdelijk onwerkbaar maken. Dit doe je door een branch te maken. Stel, we willen een inlog feature maken. We kunnen dan doen:

```
git branch feature/inloggen
```

Er lijkt nu niks veranderd te zijn, maar als we nu `git branch --list` doen, zien we dat er iets is veranderd: een tweede branch verschijnt in een overzicht.

```
λ MacBook-Pro-van-Raoul gittest → λ git main → git branch feature/inloggen
λ MacBook-Pro-van-Raoul gittest → λ git main → git branch --list
```

We gaan nu naar die andere aftakking, die op het moment een exacte copy van onze main branch is. We doen dat met `git checkout feature/inloggen` (tip, je kunt de TAB toets gebruiken om lange branchnamen te voltooien nadat je de eerste paar letters hebt getypt)

We gaan nu in deze branch iets veranderen. Voeg als eerste regel in `main.py` een import toe:  
`import numpy as np`  
Als je na het opslaan de status opvraagt, zie je dat `main.py` inderdaad is veranderd.

```
λ MacBook-Pro-van-Raoul gittest → λ git feature/inloggen* → git status
On branch feature/inloggen
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
       modified:   main.py

no changes added to commit (use "git add" and/or "git commit -a")
λ MacBook-Pro-van-Raoul gittest → λ git feature/inloggen* →
```

We zijn tevreden, en gaan dit weer vastleggen:  
`git add main.py`  
`git commit -m "numpy import"`

Nu kun je iets interessants zien, als je `git log --oneline --graph` gebruikt.

```
* 4c1da59 (HEAD -> feature/inloggen) numpy import
* 870a2ae (main) main en readme
* 7316d30 tweede script toegevoegd
* d1f9082 script toegevoegd
(END)
```

We zien dat de main branch nog steeds bij “main en readme” staat, maar dat we een nieuwe branch hebben die een extra commit heeft (“numpy import”), en daarmee een commit voorloopt.

Je kunt nu eenvoudig switchen tussen de twee branches. Geef afwisselend het commando `git checkout main` en `git checkout feature/inloggen` en observeer hoe je heen en weer switched tussen de twee toestanden in `main.py` zelf.

In VS code heb je de plugin “git graph”, waar het allemaal wat overzichtelijker wordt weergegeven, maar het principe is hetzelfde. Stel nu dat ik switch naar de main branch, en daar nog een commit doe die “add list” heet, dan ziet dat er in VS code zo uit:

Graph	Description	Date	Author	Commit
	<b>main</b> add list	3 May 202...	R.Grouls	d952e689
	feature/inloggen numpy import	3 May 202...	R.Grouls	4c1da597
	main en readme	3 May 202...	R.Grouls	870a2aec
	tweede script toegevoegd	3 May 202...	R.Grouls	7316d30b
	script toegevoegd	3 May 202...	R.Grouls	d1f90822

Je ziet dat er nu twee versies zijn, die van elkaar zijn afgetakt. `feature/inloggen` heeft een commit “numpy import”, maar `main` heeft ondertussen ook al “add list”.

## Mergen

Nu kun je je werk ook weer samenvoegen. Stel dat wat we nog wat meer werk hebben gedaan in de `feature/inloggen` branch, en we zijn tevreden. We willen de twee stromen weer samenvoegen, en dat heet in git termen een merge.

Dit gaat moeiteloos wanneer je op verschillende regels of in verschillende bestanden hebt gewerkt. Maar als je op exact dezelfde regel dingen veranderd, gaat git om hulp vragen: welke van de twee versies wil je behouden?

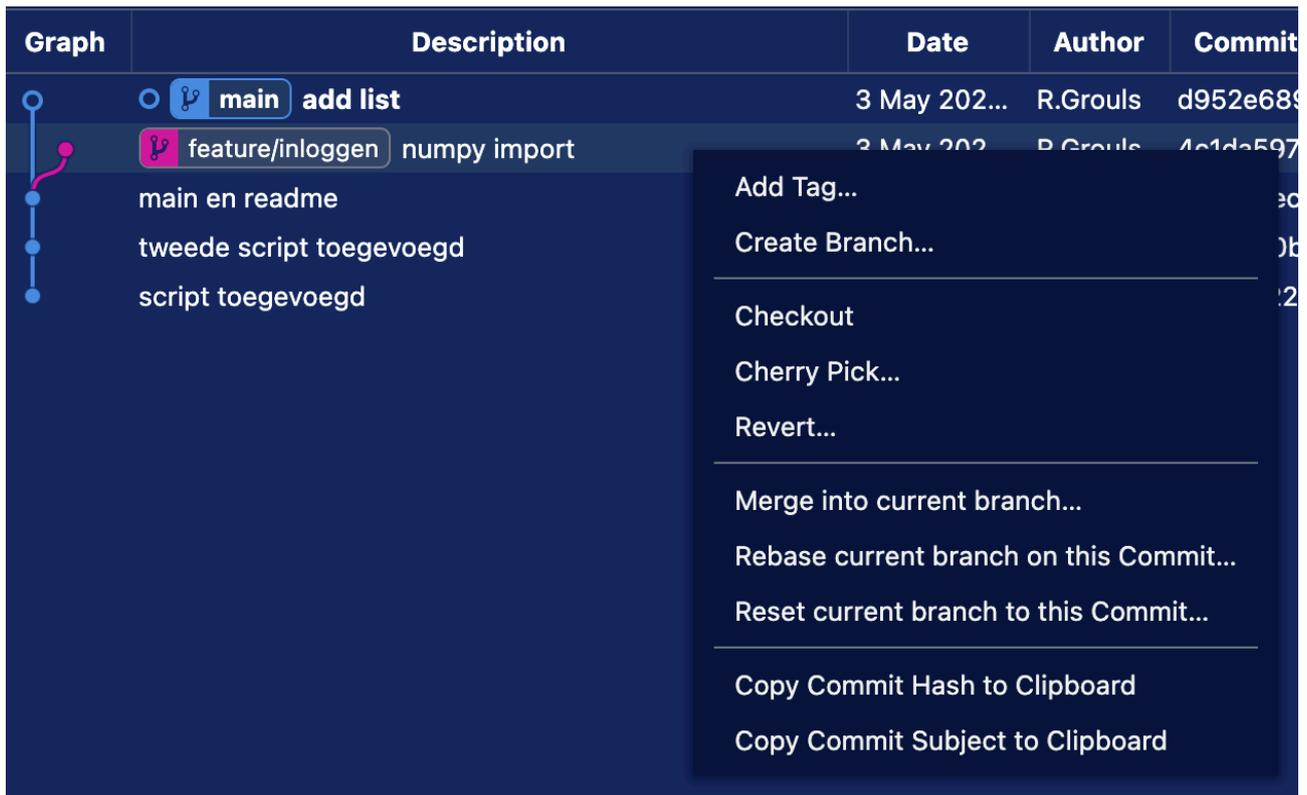
In VS code gaat dit vrij overzichtelijk. Zorg dat je je in de `main` branch bevindt, klik met de rechtermuisknop op de “numpy import” commit, en kies dan “merge into current branch...” in het dropdown menu. Je zou dit ook via de terminal kunnen doen:

```
git checkout main
git merge features/inloggen
```

zijn dan de twee commando's die je achtereenvolgens moet geven.

Als je dan kiest voor “create a new commit” in VS code, dan krijg je mogelijk een merge conflict (als je op dezelfde regels iets gewijzigd hebt)

```
λ MacBook-Pro-van-Raoul gittest → λ git main → git merge feature/inloggen
Auto-merging main.py
CONFLICT (content): Merge conflict in main.py
Automatic merge failed; fix conflicts and then commit the result.
```



## Merge conflicten oplossen

Er zijn meerder manieren om een conflict op te lossen. De meest simpele (maar waarschijnlijk minst bruikbare) manier is om de merge af te breken. Je kunt dit doen met `git merge --abort` in de command line.

Een betere manier is meestal om te gaan kijken wat er fout gaat. Als we in VS code op het bestand klikken waar het conflict is, zien we duidelijk wat er aan de hand is:

```

1 <<<<<<< HEAD (Current Change)
2 a = [1, 2, 3]
3 =====
4 import numpy as np
5 >>>>>> feature/inloggen (Incoming Change)
6

```

We zitten in de `main` branch, en in de HEAD van die branch staat op de eerste regels een lijst

a = [1, 2, 3] toegevoegd. Maar op dezelfde locatie hebben we in de inkomende branch (feature/inloggen) ook een wijziging gedaan: import numpy as np. Git wil nu weten: wat is hier de bedoeling: wil je een van beiden overschreven laten worden? Of wil je beide wijzigingen behouden?

VS code maakt het makkelijk door bovenin de optie te geven welke change je wilt behouden. Dit kan overigens ook met de commandline, maar in mijn ervaring is VS code toch echt overzichtelijker en simpeler. Ik pas de file aan zoals ik wil, en sla dit resultaat op.

```
λ MacBook-Pro-van-Raoul gittest → λ git main* → git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   main.py

no changes added to commit (use "git add" and/or "git commit -a")
```

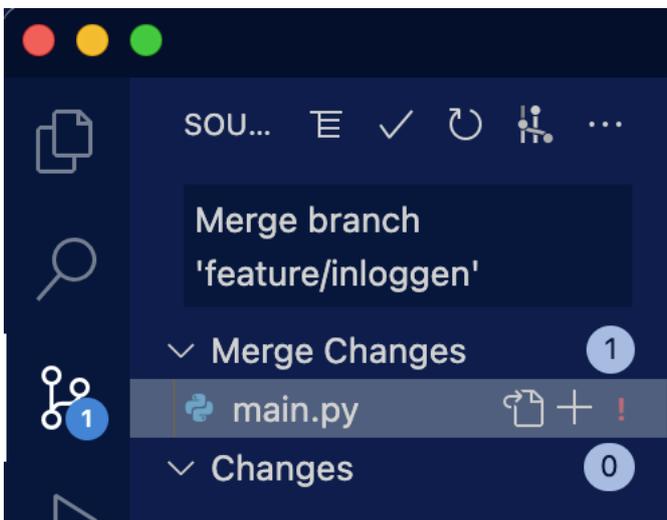
In mijn git status is het nu duidelijk wat er aan de hand is:

- Ik ben bezig met mergen, maar de merge is nog niet voltooid (you have unmerged paths)
- Ik kan alsnog het hele proces afbreken met `git merge --abort`, maar ik kan ook de conflicten oplossen en die committen.

Ik ga dat laatste doen:

```
git add main.py
git commit -m "merge conflict"
```

Of, alternatief, via VS code:



- Kies het branch tabblad links in de kolom
- klik op het + teken achter `main.py`, onder `merge changes`
- `main.py` verhuist naar `staged changes`
- Klik op het vinkje bovenin beeld: er wordt een merge branch commit toegevoegd

Graph	Description	Date	Author	Commit
	<code>main</code> Merge branch 'feature/inloggen'	3 May 2022 15...	R.Grouls	4be1be32
	add list	3 May 2022 15...	R.Grouls	d952e689
	<code>feature/inloggen</code> numpy import	3 May 2022 13...	R.Grouls	4c1da597
	main en readme	3 May 2022 13...	R.Grouls	870a2aec
	tweede script toegevoegd	3 May 2022 13...	R.Grouls	7316d30b
	script toegevoegd	3 May 2022 13...	R.Grouls	d1f90822

Het eindresultaat is dat alle code weer is samengevoegd in de main branch.

## Git push en pull

Als laatste kunnen we nu deze repository ook synchroniseren met de cloud. We kunnen daarvoor bijvoorbeeld [github.com](https://github.com) gebruiken.

- maak een account aan op github
- bij repositories kies je voor New
- geef je repository een naam en maak je keuzes uit de instellingen. De defaults zijn altijd veilig.

Je komt nu met een overzicht van drie situaties. In ons geval moeten we kiezen voor “push an existing repository from the command line”. Het eerste wat we doen, is aan onze lokale git vertellen waar onze remote git zich bevindt:

```
git remote add origin git@github.com:raoulg/gittest.git
```

vervolgens pushen we onze lokale branch. Git suggereert om onze branch te hernoemen naar `main`, met `git branch -M main`, maar dat is in ons geval niet nodig (want de branch heet al `main`). Dus het enige wat we nog hoeven te doen is

```
git push -u origin main
```

de `-u` toevoeging zorgt dat elke lokale branch wordt gekoppeld aan een upstream branch.

`origin` is een shorthand voor de remote repository en wordt gebruikt in plaats van de originele URL.

Schematisch ziet dit er zo uit: we stagen met `git add`, committen met `git commit`, en kunnen van en naar de remote pullen en pushen. Tussen branches kunnen we een checkout of merge doen.

