De samenvatting

1. *Fork* de DEF-D *repo* naar je eigen GitHub-account.
2. Nodig de hoofdTA's uit als *collaborators*.
3. Maak een *branch* van `main` naar je huidige opdracht.
    - werk aan opdracht (in de juiste *branch*!) totdat je klaar bent, met *commits* tussendoor.
4. *Push* je werk naar GitHub
5. Vraag een *Pull Request* aan van jouw huidige opdracht *branch* naar `main` en laat jouw hoofdTA (dit verschilt per opdracht) de *reviewer* zijn
    - Maak ook een *PR* aan naar de `main` *branch* van je groepsgenoten.
    - Verbeter eventueel nog je werk met nieuwe *commits&pushes* en vraag weer een *review* aan!
6. De opdracht is goedgekeurd, de TA doet dit voor je:
    - *Squash en merge* je huidige *branch* met `main` en verwijder de huidige opdracht *branch*.

NOTE: als je dan via je *fork* terug naar de originele *repo* een *PR* aanmaakt is dit precies hoe grote open-source projecten werken!

Verder met Git

Voor DEF-D gaan we iets verder in op werken met *Git*. We gaan nu echt werken met *version control*; dit betekent dat er een aantal nieuwe dingen geleerd moeten worden! Namelijk: - Een *repository forken* - *Branch* aanmaken - *Collaborators* toevoegen - *Committen* en *pushen* - *Branches mergen*: *pull requests*

Ook hebben we hier een link met wat jullie hebben geleerd bij Inleidend Practicum, daar gaan we op verder bouwen. Elk onderdeel gaat ook een voorbeeld hebben van de introductie opdracht.

Forking

Een *repository* (*repo*) *clonen* is handig om er zelf op jouw laptop aan te werken, maar hoe deel je nu wat jij hebt gedaan met je groepsgenoot? Dit gaan we oplossen door de opdrachten eerst te *forken* naar je eigen GitHub-account. In essentie kopiëer je dus eerst de *repo* naar jouw account en van daar *clone* je hem naar je eigen laptop. Dit betekent dat een kopie van de *repo*, met alle notebooks, daadwerkelijk op de harde schijf van jouw laptop staat en je eraan kan werken als je laptop niet met internet verbonden is. Je kan er dan aan werken in welke omgeving je wilt, veel van jullie gebruiken daarvoor VSCode.

Waarom? Nu kan je samen met je groepsgenoot werken aan de opdracht. Jullie hebben dezelfde *fork*, maar werken in je eigen *clone*. Zodra je een stuk werk af hebt kan je dat via GitHub makkelijk delen, want de github.com website houdt een versie van jullie *fork* bij. Het is natuurlijk niet de bedoeling dat jullie jullie werk *pushen* naar de DEF *repo* ;)

***Forken* doen we maar 1 keer!** Dit zal voor de meeste op de kickoff dag gebeuren.

Voorbeeld

We beginnen op de DEF-D opdracht repo, deze gaan jullie *forken*. *Forken* staat rechtsbovenin aangegeven:

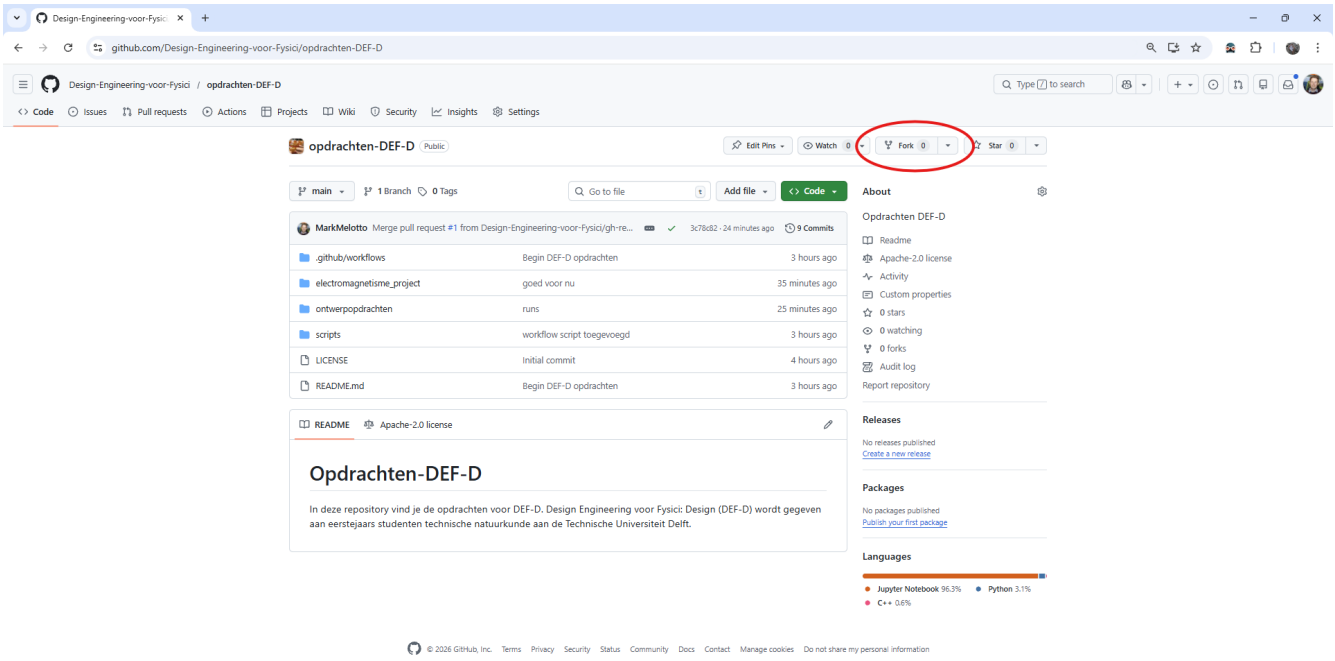


Figure 1: *Fork*: waar doe je dat.

Klik op `Fork` dan kom je hier.

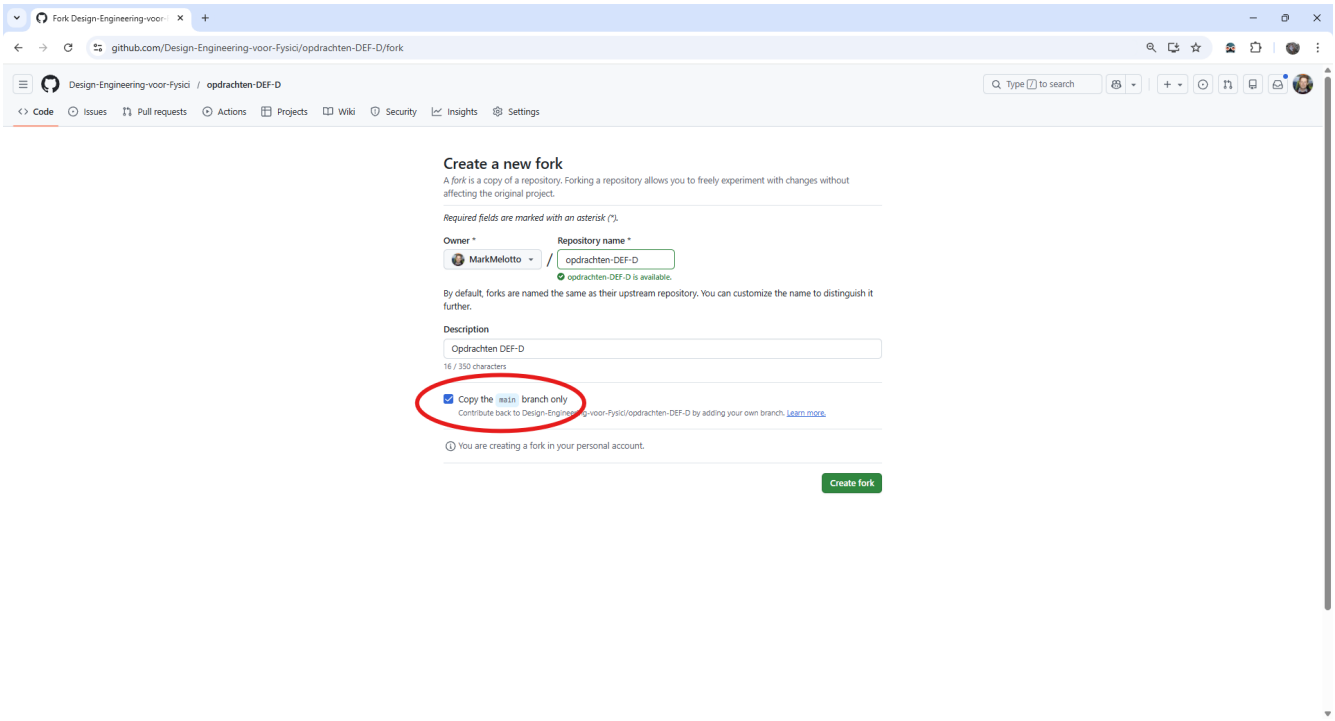


Figure 2: De *fork* starten

Let op dat je **alleen** de main *branch forked* naar jouw account, verzin een *repo* naam en `Create fork`, ik hou de naam hetzelfde: ‘opdrachten-DEF-D’.

Nice, je hebt nu de DEF *repo* op je eigen account! Het zou er nu zo uit moeten zien, dan kan je door naar het volgende onderdeel.

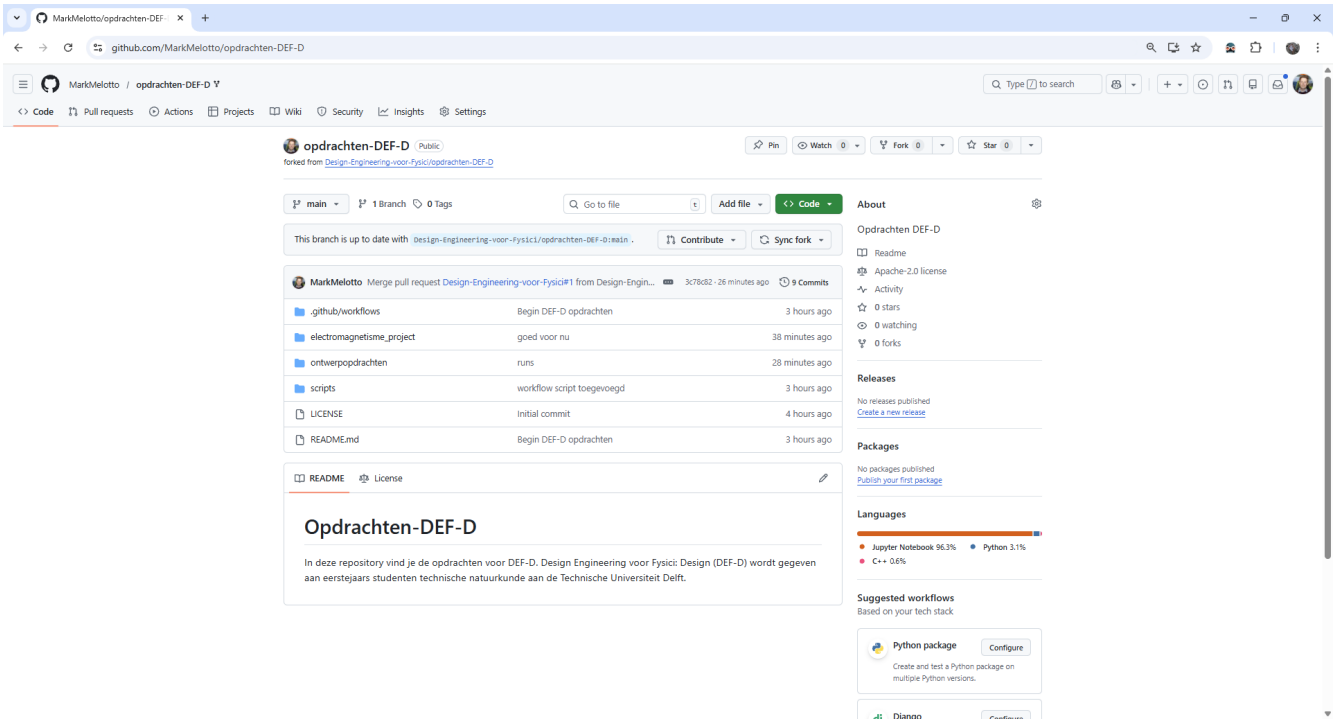


Figure 3: *Forken* is gelukt.

Dit is een andere manier om de *repo* te *clonen* van wat je eerder hebt geleerd bij IP.

LEAVE NOOIT DE FORK

Collaborators toevoegen

Nodig nu je groepsgenoot uit voor jouw *repo*, of accepteer die van je groepsgenoot, mits dit nodig is. **Nodig ook de 3 hoofdTA's uit**, dit is belangrijk voor nakijken later! Hoe dit moet staat hier, bij `settings` zie je een tab *collaborators*.

3 hoofdTA GitHub accounts

- Martijn Sonneveld: MartijnSonneveld
- Emma Aspeslagh Nielen: Enielen
- Sjoerd Hoogeman: SHoogeman

Als je het niet zeker weet, dan kan je hier de hoofd TA's zien.

Actions aanzetten

We hebben een aantal GitHub *actions* gemaakt die jullie gaan helpen: automatische checks. Deze moet je wel even aanzetten op je eigen Github page, zie dit screenshot:

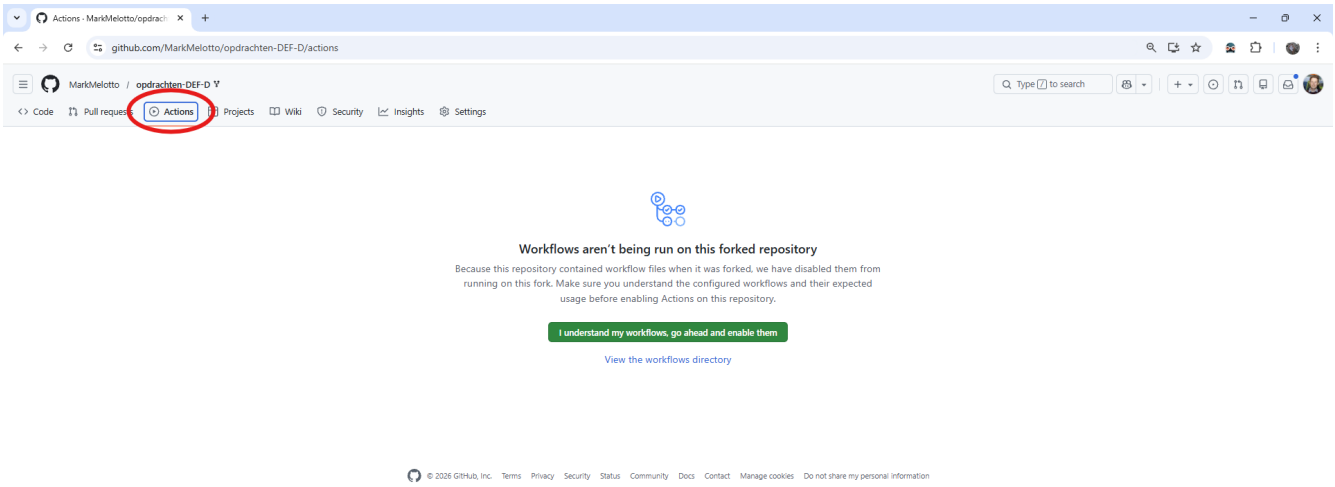


Figure 4: GitHub actions.

En enable de workflows.

Protecting main

Een goede eigenschap is om je main *branch* te *protecten*. Zo bescherm je je werk waarvan je zeker weet dat het werkt! (Denk aan AWS die laatst omviel…) Ook helpt het om kleine foutjes te spotten die jou zijn ontgaan.

Dit zorgt er ook voor dat je met *branches* moet werken! Hiervoor gaan we op de GitHub page naar `settings --> branches --> Add branch rule`.

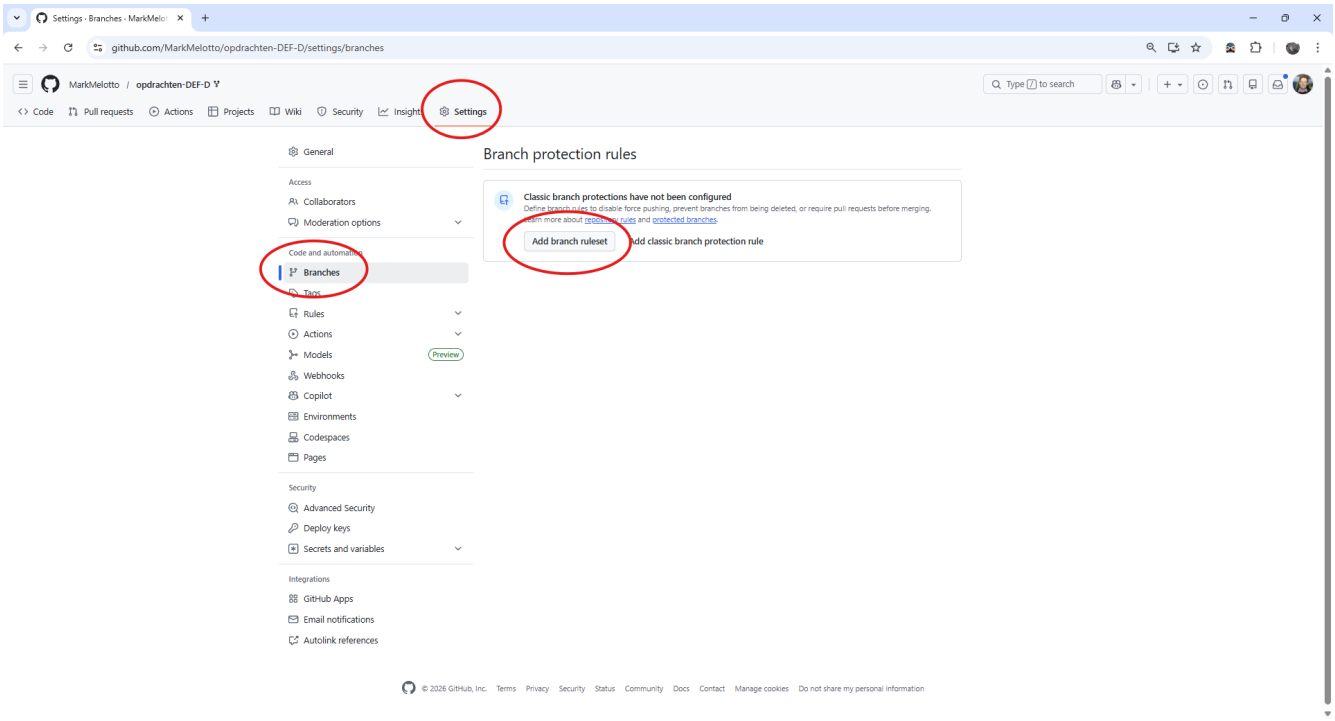


Figure 5: Start van: protecting main.

Druk op `Add branch rule`, geef de goede 'Ruleset Name', zet hem op active en geef de juiste branch targeting criteria, zet deze op default:

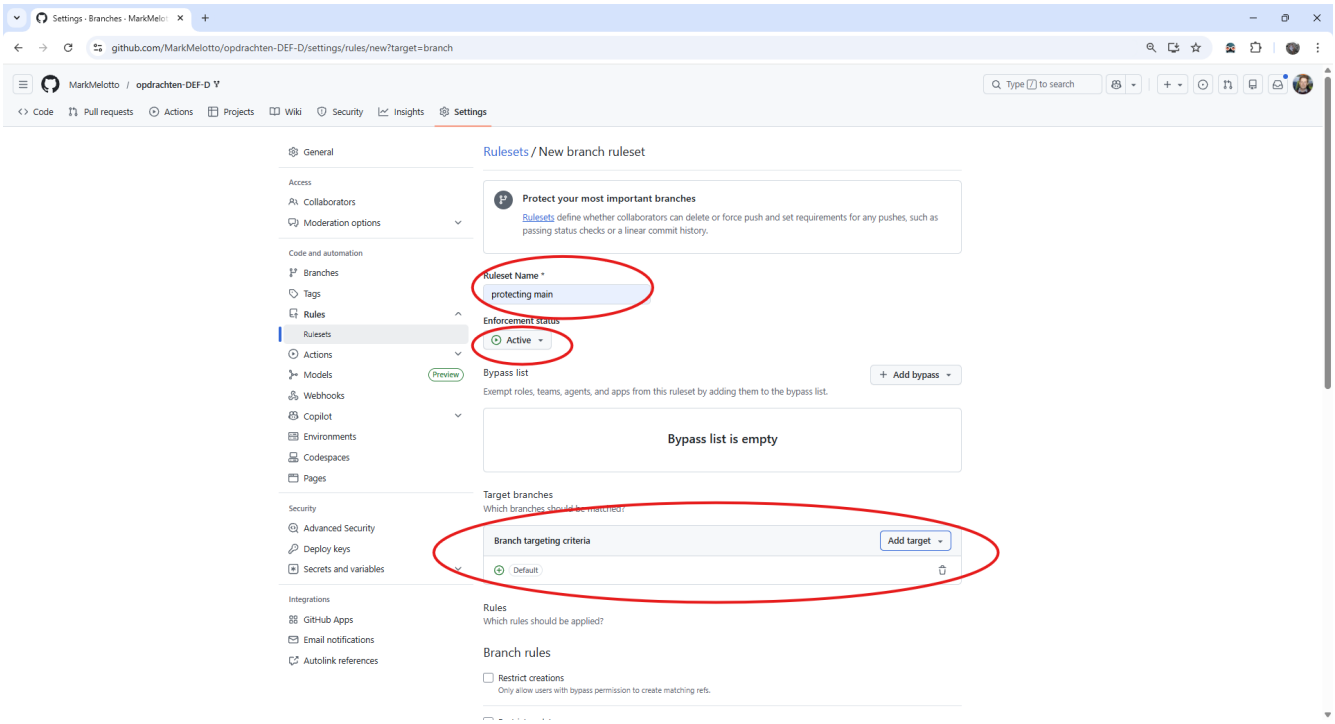


Figure 6: Protecting main aanmaken.

Scroll dan naar beneden en vink aan: ‘Require a pull request before merging’ en zet Required approvals op 1:

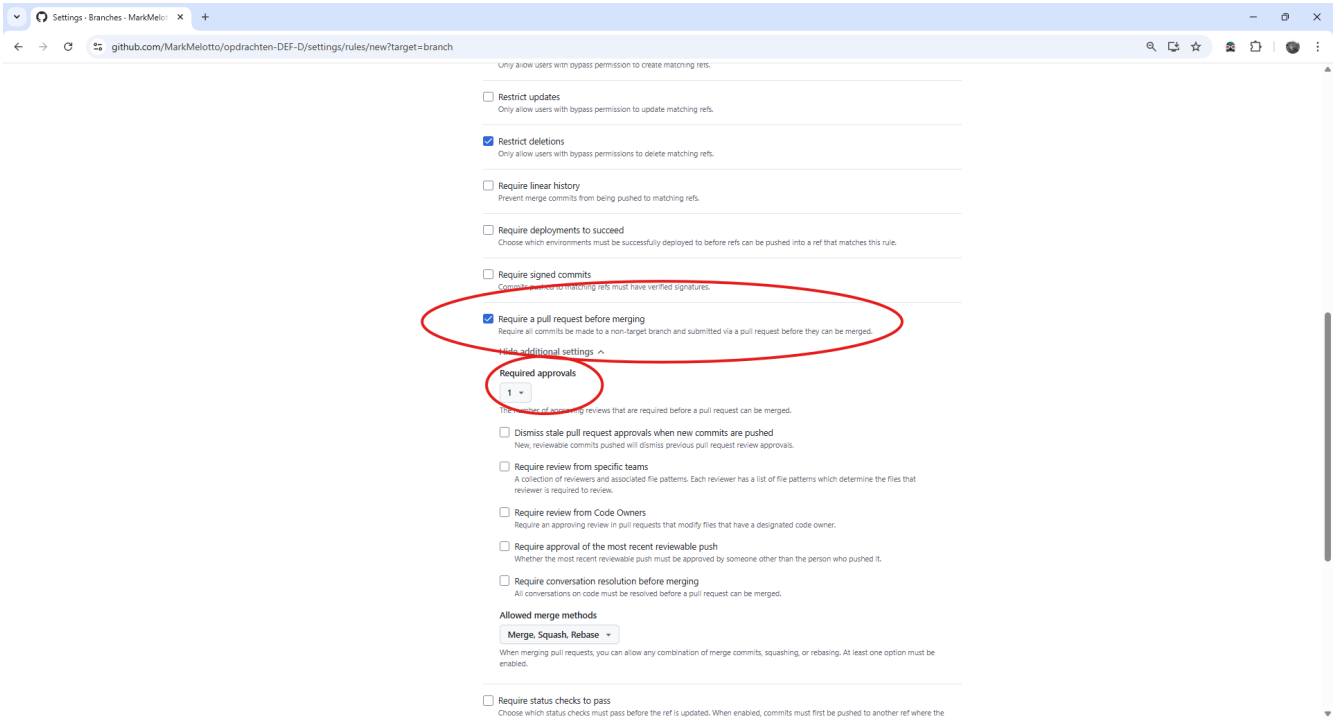
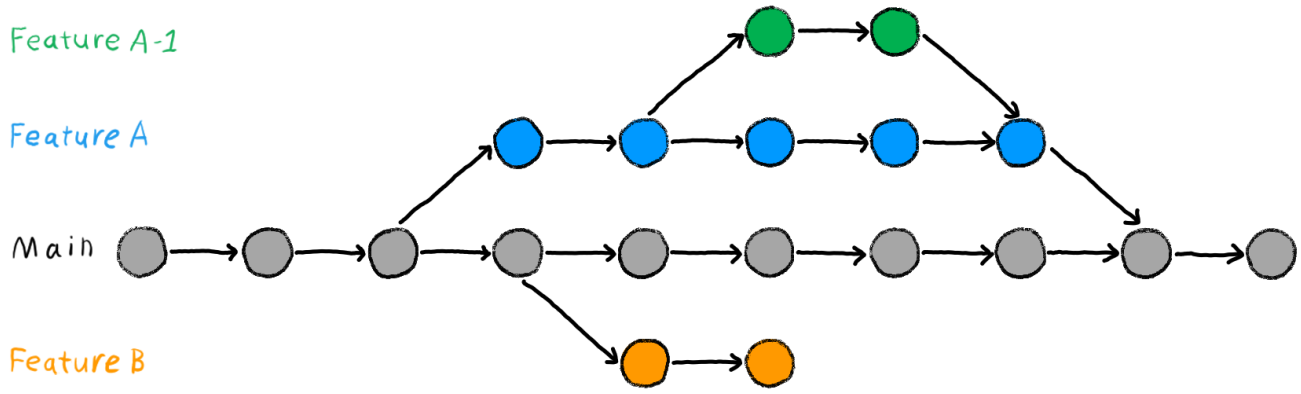


Figure 7: Protecting main settings.

Druk dan op `Create`!

Git Branches

Als je de *repo* met de opdracht hebt *geforkt* dan kan je ook alle *branches* mee krijgen, voor DEF-D heb je alleen de *branch main*. Een *branch* is een kopie van de *repo*, dit kan een kopie zijn van *main*, maar ook weer van een andere *branch*. Jullie gaan een kopie van *main* maken. Een *branch* wordt gebruikt om aan verschillende functionaliteiten te werken, voor ons betekent dat: een *branch* voor elke opdracht! Jullie eerste *branch* zou iets zijn als: `introductie`. Maak de *branch* voor de opdracht pas aan als je er ook echt aan gaat beginnen.



An illustration of branching in Git. Taken from here. Illustration by Scriberia. Used under a CC-BY 4.0 licence. DOI: The Turing Way Community & Scriberia (2024).

Voorbeeld: Branch maken

Nu je de *repo* op je eigen account hebt *geforkt*, klikken we op de dropdown van de *branches*:

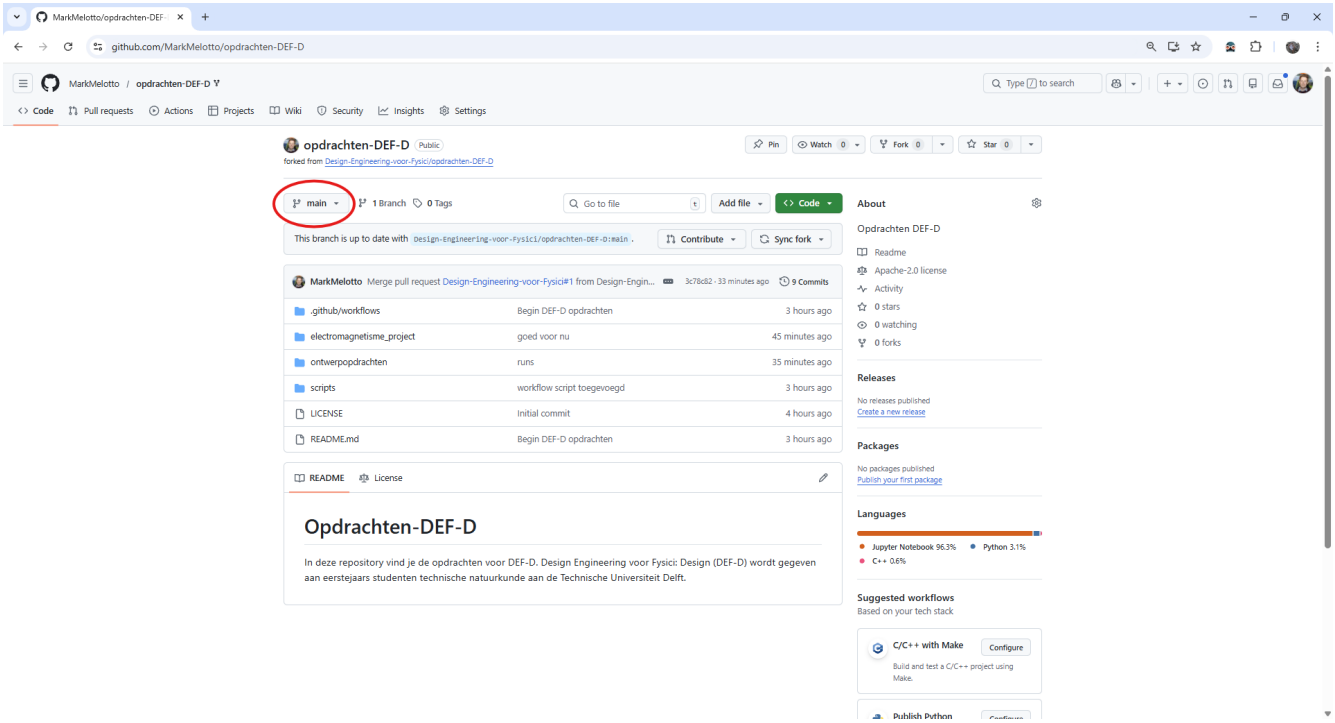


Figure 8: Branch plek.

Dan om de nieuwe `opdracht` branch aan te maken typen we in de dropdown: ‘opdracht’ en klikken we: ‘create branch’.

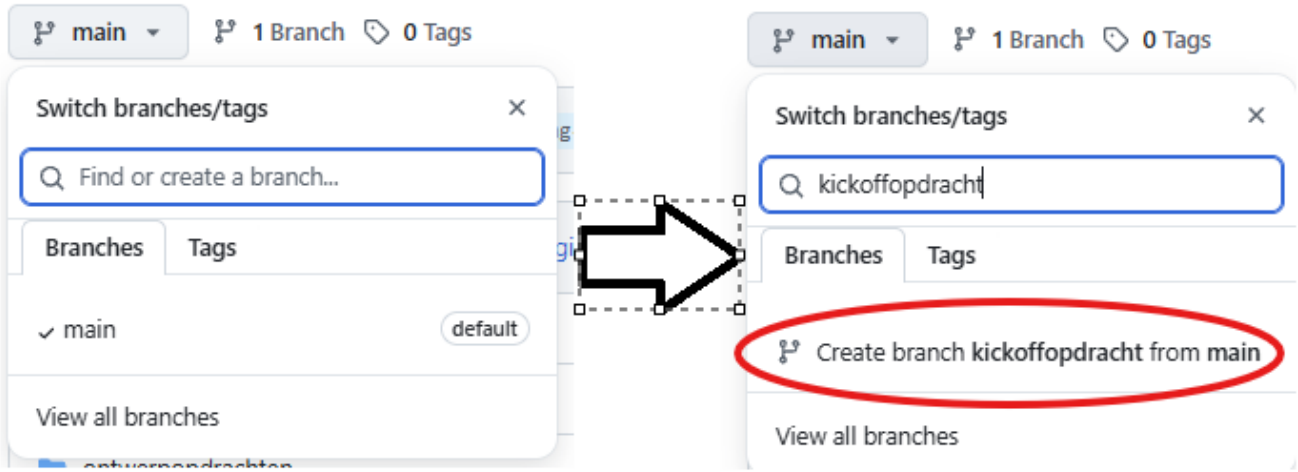


Figure 9: Branch aanmaken.

In hetzelfde dropdown menu als eerst kan je nu switchen tussen de *branches*.

Voorbeeld: naar je branch switchen in VSCode

Om dit in VSCode te doen, gaan we eerst jullie *repo clonen* naar VSCode en dan de goeie *branch* in. In de terminal kan je dit typen nadat je *gecloned* hebt: `git checkout <opdracht>`.

Het kan ook via deze manier:

1. ga naar het menu van de *repo* in VSCode:

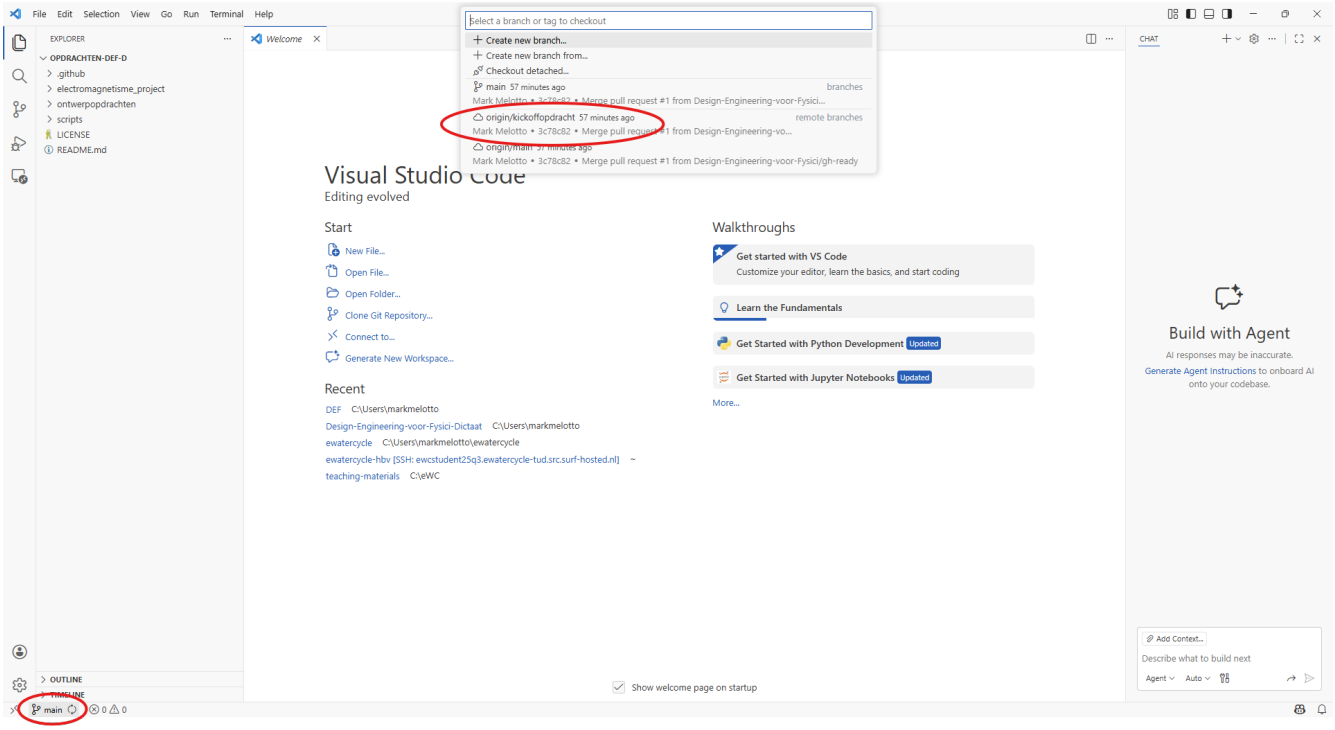


Figure 10: Switchen naar de goede branch

2. klik dan op de nieuwe `opdracht` branch.

Pas als je in de goede *branch* zit, ga je werken aan je opdracht!

Terminal (optioneel)

Je kan als je dat wilt ook werken met de command line (ook wel terminal genoemd). In dat geval zijn dit de commando's die je gebruikt:

Branch maken en meteen ernaar toe switchen: `git checkout -b <naam van je nieuwe branch>`.

Anders kan je `git checkout <branchnaam>` gebruiken.

Git Push

Nu je de opdrachten allemaal op je eigen GitHub-account hebt en een *branch* voor je opdracht hebt, en je weet hoe je dit op VSCode krijgt, zijn jullie klaar om opdrachten te maken.

Als je lokaal (dus op je laptop *gecloned*) je opdracht hebt gemaakt met de juiste *commits*, gaan we *pushen* naar GitHub. Als het goed is, is dit al bekend, zo ja: ga dan verder naar het samenwerken. Zo niet, dan staat het hier ook onder bij het voorbeeld.

LET OP, voordat je inlevert, moet je `restart & run all` hebben gedaan in de notebook. Dit moet zodat alles goed staat en plaatjes goed inladen!

Voorbeeld

Ik heb hier de kickoff opdracht gedaan:

1. *stage* mijn verandering, dus eerst de vernieuwde notebook:

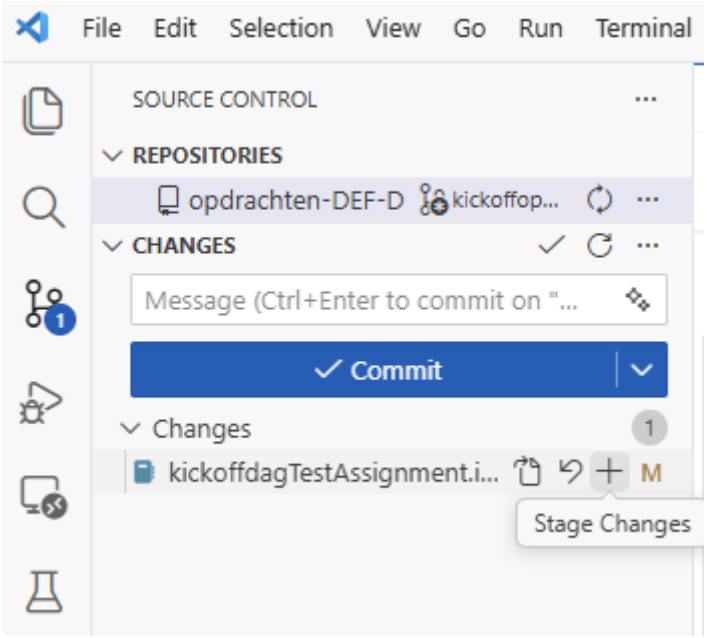


Figure 11: Geüpdate files stagen.

2. Dan stage ik ook de foto en dan schrijf ik de *commit message*:

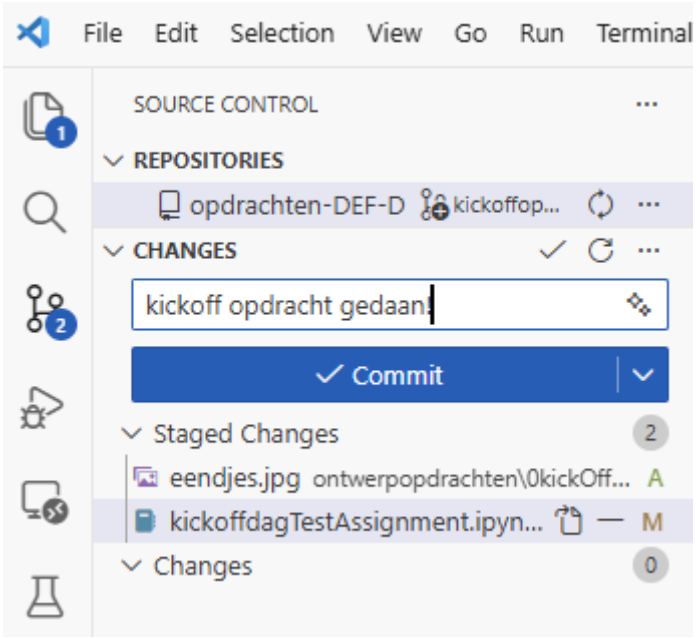


Figure 12: Geüpdate files comitten.

- Hoe je dit deed staat hier.

3. Dan *sync (push)* ik mijn VSCode naar GitHub:

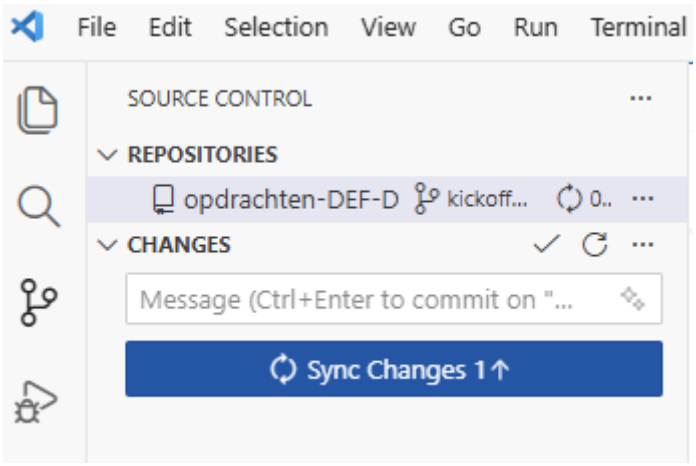


Figure 13: Geüpdate files pushen/syncen.

4. Dat ziet er nu zo uit:

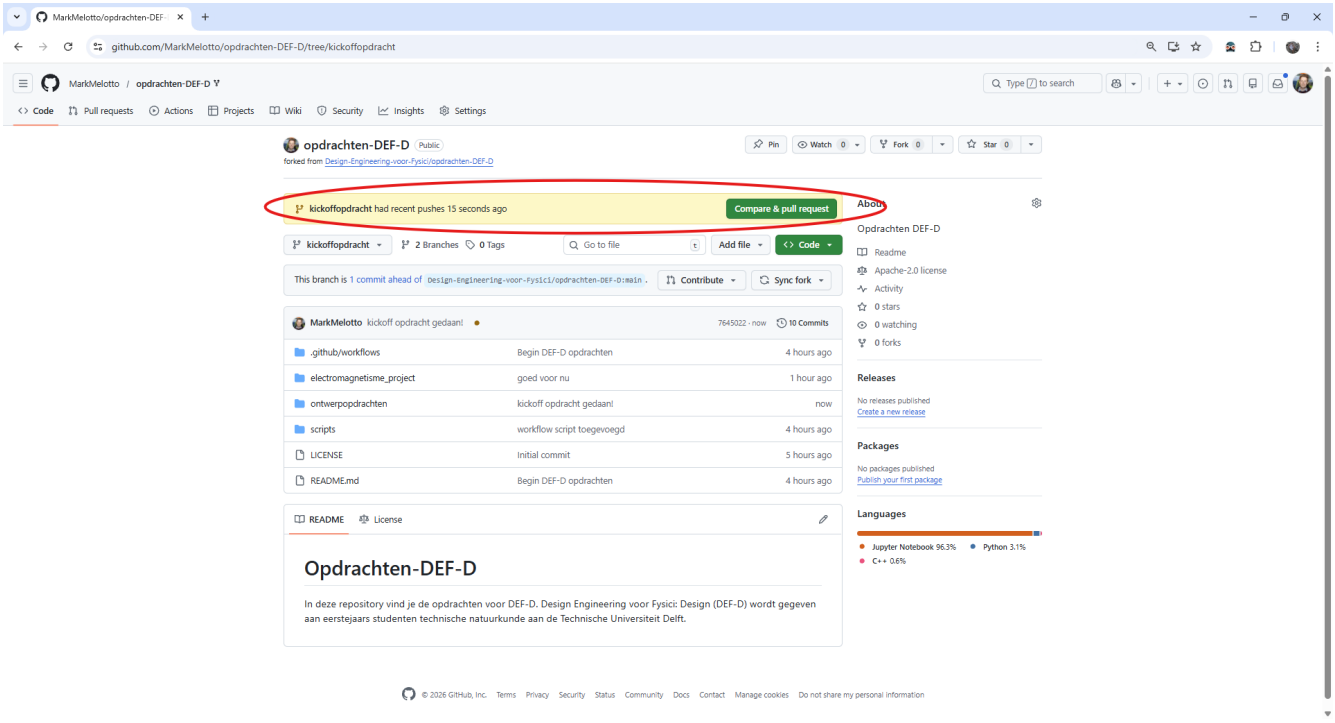


Figure 14: Files zijn gepusht: GitHub ziet er nu zo uit.

Terminal (optioneel)

Je kan als je dat wilt ook werken met de command line (ook wel terminal genoemd). In dat geval zijn dit de commando's die je gebruikt:

- Gebruik `git status` om te zien of er files zijn die moeten worden toegevoegd aan je *staging*.

- `git add <file>` *staged* de file die je hebt aangepast, dit hoef je niet altijd zelf te doen!
- Dan wordt het `git commit -m <jouw commit message>` om alle *staged files* te *committen*.
- Daarna: `git push`

Samenwerken met Git

Voor de woensdagen werken jullie als het goed is op 1 laptop. Dit maakt werken met GitHub wat simpeler; 1 persoon zet alles klaar op zijn GitHub *repo* (maak een nieuwe branch aan!) en past de notebook aan en pusht dit naar de juiste branch.

Voor de thuiswerkopdracht is het belangrijk dat jullie beiden commits maken! Voor branches is het dan gewoonlijk om dit aan te houden: main –> opdracht 1 (elektrostatica) –> student 1, 2. Maar het is ook prima om aan pair programming te doen. Dan hoef je ook niet aparte branches aan te maken voor ieder account, maar werk je op de branch van de opdracht.

Laat dus in je commits ook weten hoe je een sub-opdracht hebt gemaakt!

Git Pull Request

Met een *Pull Request* vraag je aan je samenwerkers (in bij DEF: aan je hoofdTAs) toestemming om de wijzigingen in jouw *branch* toe te voegen aan de *branch* waar de *pull request* naar toe gaat, in ons geval de `main` *branch*. Om nu jullie opdracht na te laten kijken en terug te *mergen* met jullie `main` *branch* gaan we een *pull request* (PR) doen. De *pull request* doe je als je klaar bent met de opdracht. GitHub kijkt dan of alles klopt en runt checks die wij hebben toegevoegd.

Nakijken

Om na te kijken gebruiken wij het review-systeem van Git in de *PRs*. Als de *PR* is aangemaakt kan je rechts bovenin een *reviewer* aanklikken en zet je de link van de *PR* in de bijbehorende assignment op Brightspace. De reviewer moet jullie TA worden, deze gaat dan nakijken en laat wat comments achter en vraagt misschien om verandering als we vinden dat het nog niet helemaal correct is. Als dit zo is, dan moeten jullie aanpassingen gaan maken, dit kan gewoon weer in dezelfde *branch* met nieuwe *commits* (vb: commentaar verwerken). Zodra je alles hebt verbeterd, vraag je opnieuw de review aan van je TA.

Op Brightspace houden we uiteindelijk bij of de opdracht namelijk succesvol is afgerond

Voorbeeld

Nu alles is gesynct op mijn GitHub account ga ik een *pull request* doen van mijn `opdracht` branch naar `main`.

1. Ik klik op ‘Compare & pull request’:

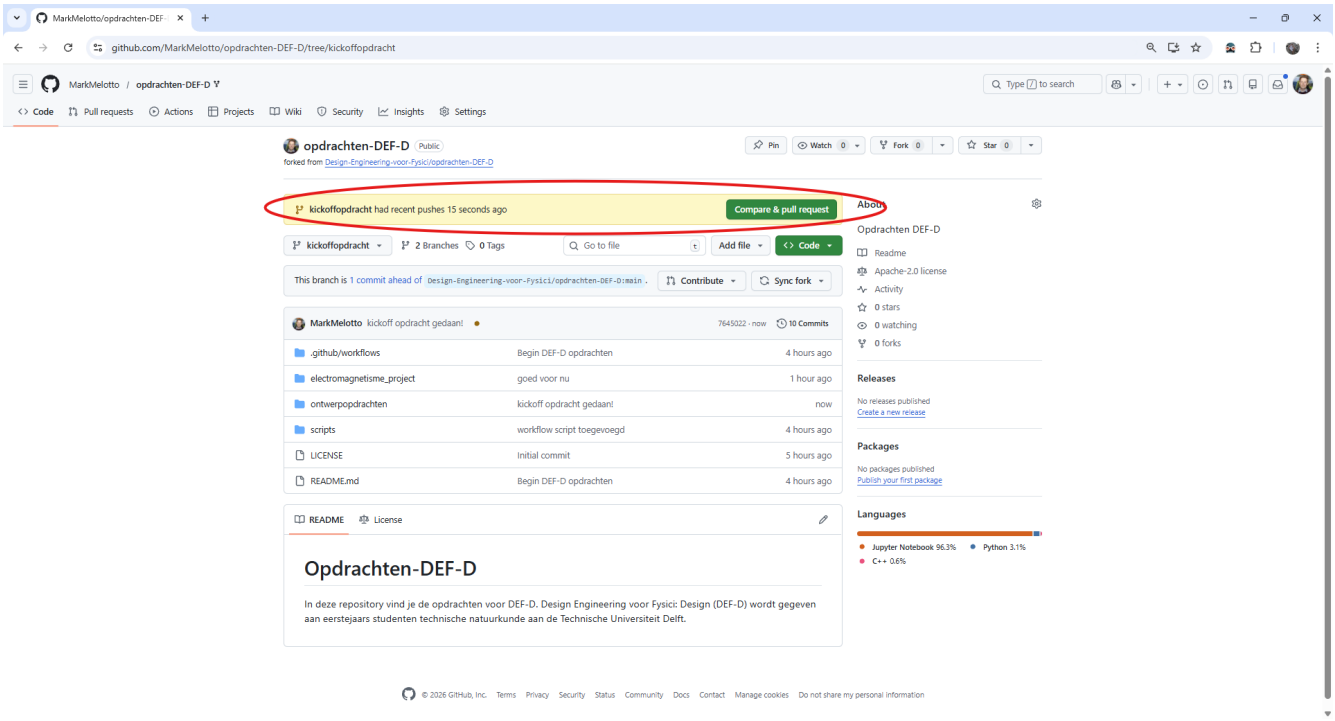


Figure 15: Pull request starten.

2. Nu zijn we in het *pull request* menu:

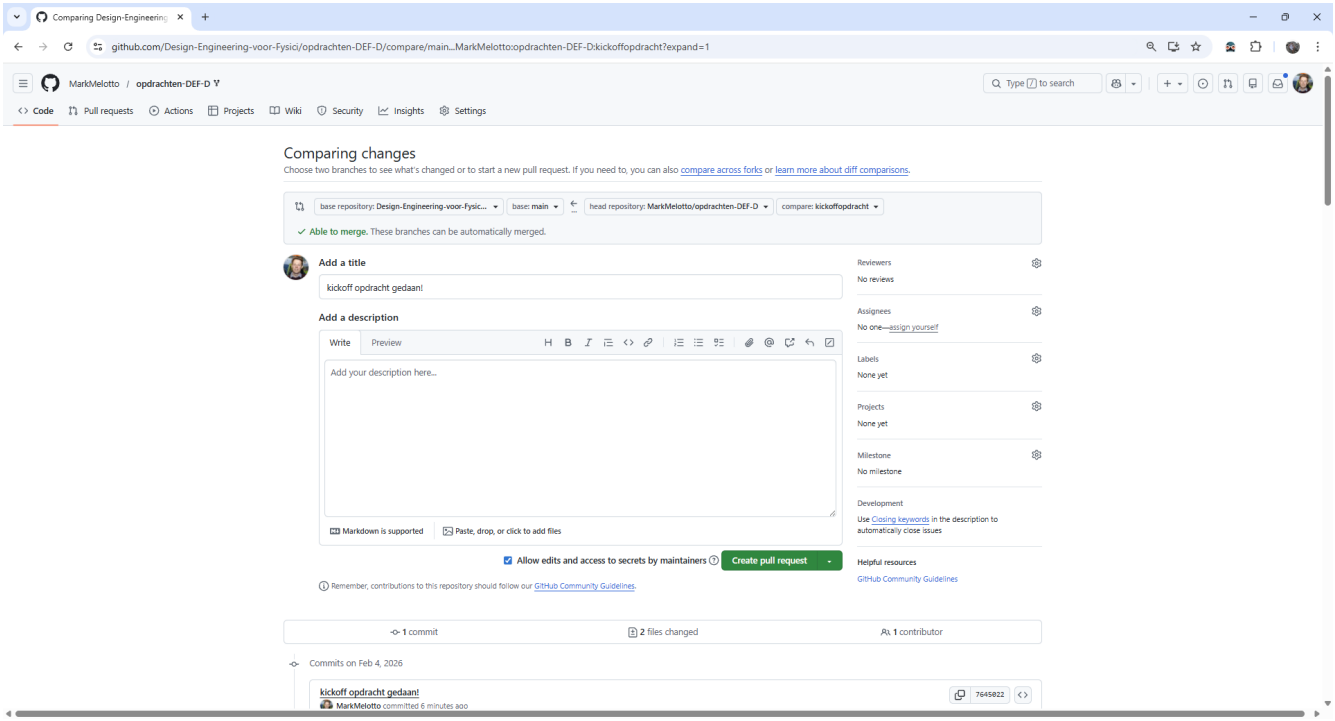


Figure 16: Pull request menu.

3. We moeten zorgen dat we niet een *PR* openen naar de DEF-D opdrachten *repo*, maar naar jouw eigen `main` *branch*, dit doe je hier:

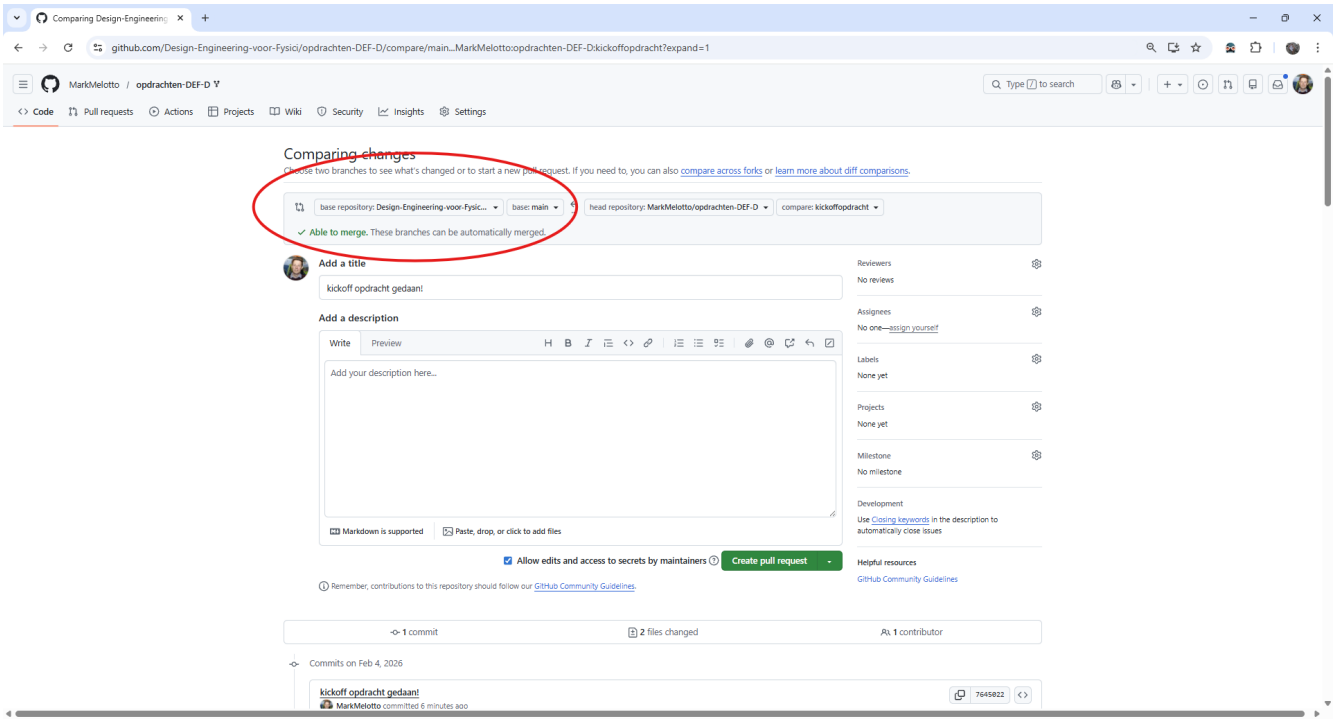


Figure 17: Base veranderen naar jouw eigen `main` branch.

4. Bij *reviewers* moet je jouw nakijker toevoegen, in mijn geval is dit hoofdTA: MartijnSonneveld.
    - Je hoort op de dag zelf wie jouw nakijker is.
5. Zet nu de link van jouw *PR* op de juiste plek in Brightspace.
    - Voor DEF-D moet je meestal zodra je je *pull request* hebt aangemaakt dit op Brightspace melden in een assignment. Dat is belangrijk om voor je DEF-D onderwijs je punten te krijgen. Dat doe je door de URL/link te kopieren. Dus de link van de pagina van je *PR* to kopiëren en die te plakken in de juiste assignment op Brightspace.
6. Martijn heeft nagekeken en was niet tevreden:

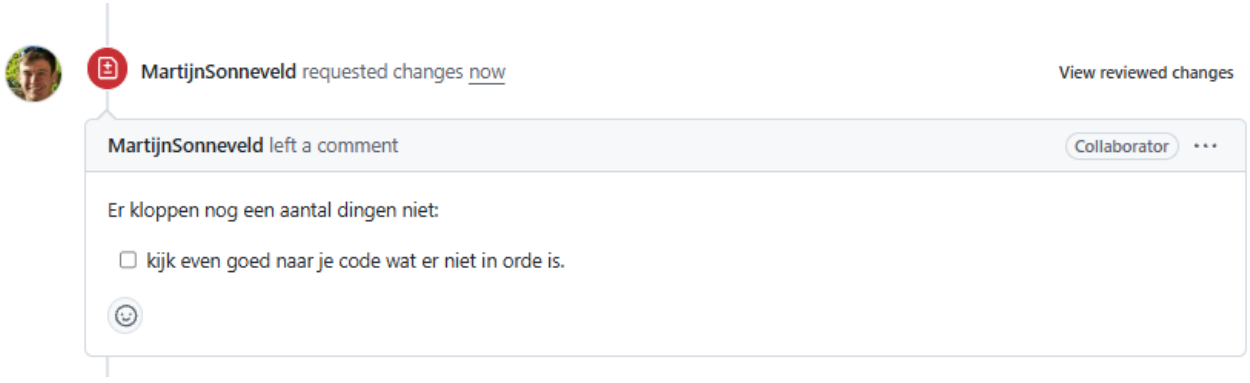


Figure 18: Feedback: niet goed genoeg.

7. Ik maak veranderingen en commit en push die, daarna kan ik nog een review aanvragen van Martijn:

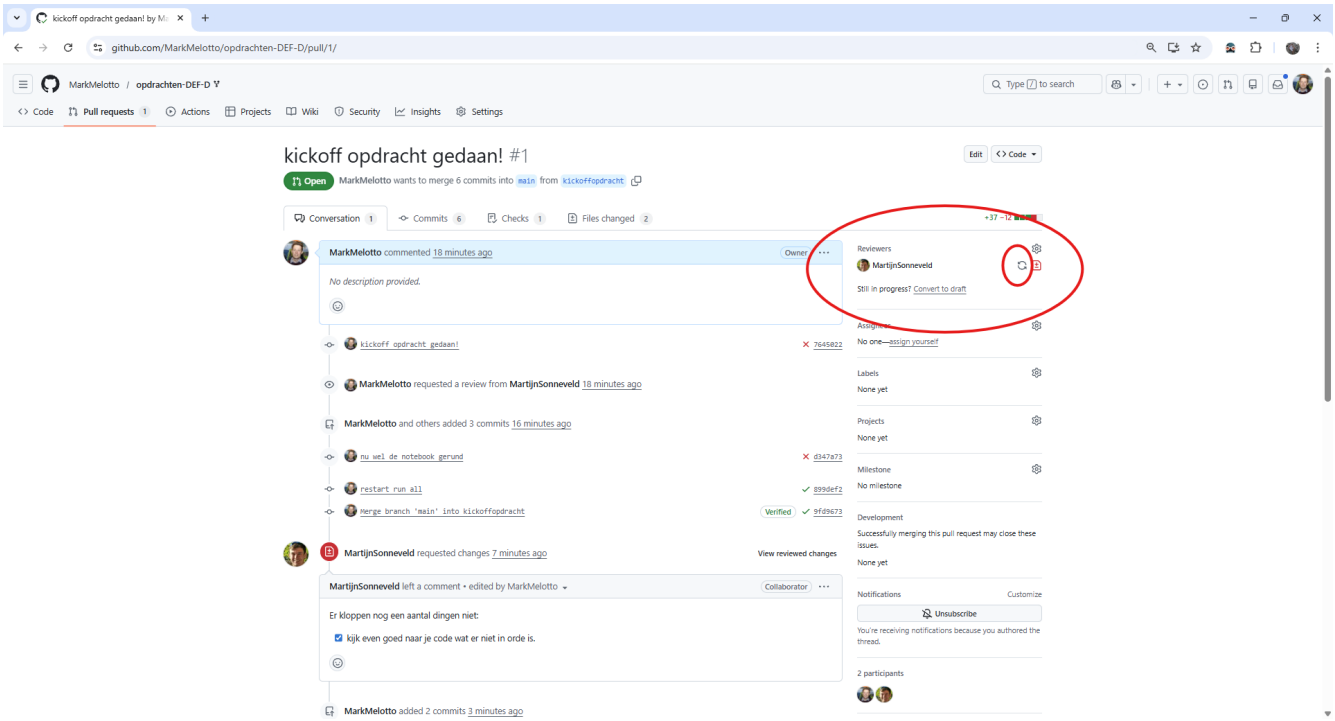


Figure 19: Nieuwe review aanmaken.

8. Deze keer vond Martijn het goed!

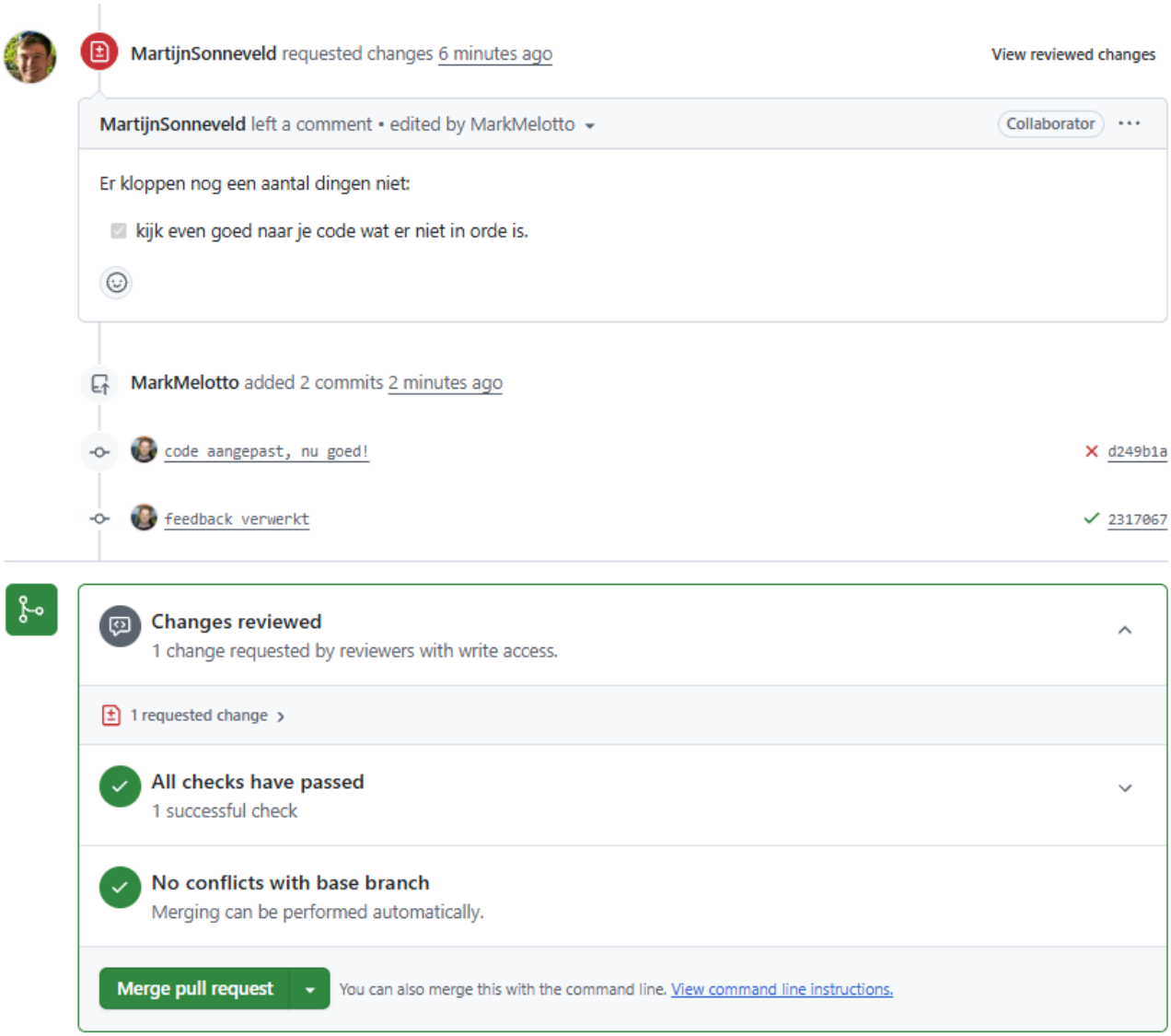


Figure 20: Feedback: goed gekeurd.

Nu willen we mergen met de `main` *branch*, *squash & merge* om precies te zijn!

Nu het is goedgekeurd door de TA en zal hij zorgen dat alles op jullie `main` *branch* komt! Dit doet hij door deze stappen te volgen:

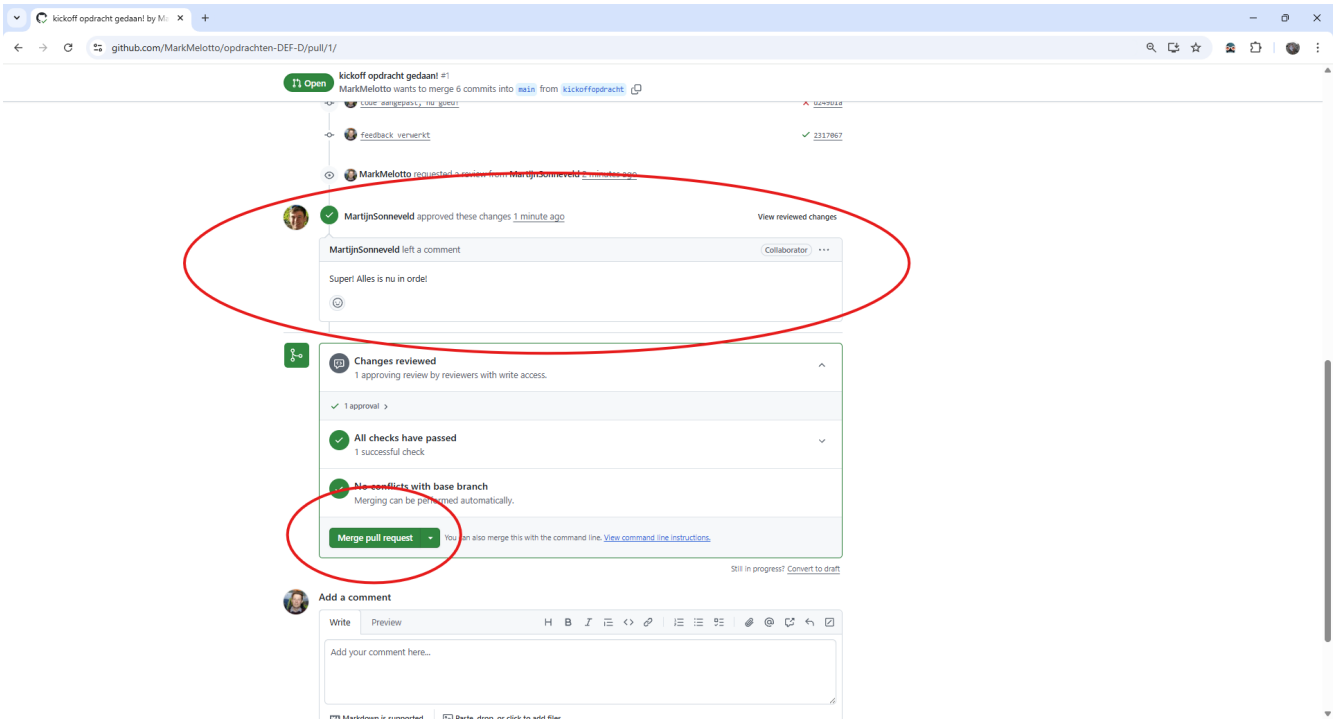


Figure 21: Branches mergen.

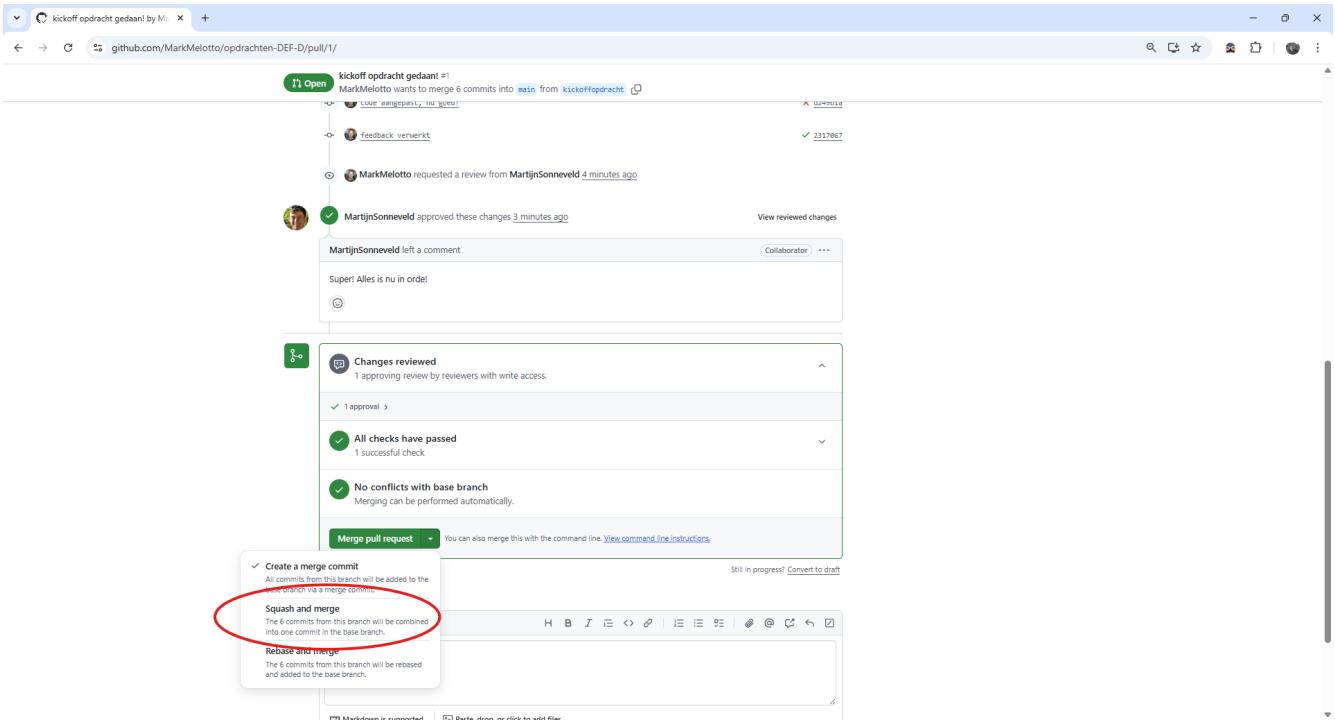


Figure 22: Squash mergen selecteren.

- Dan deleten we de *branch*:

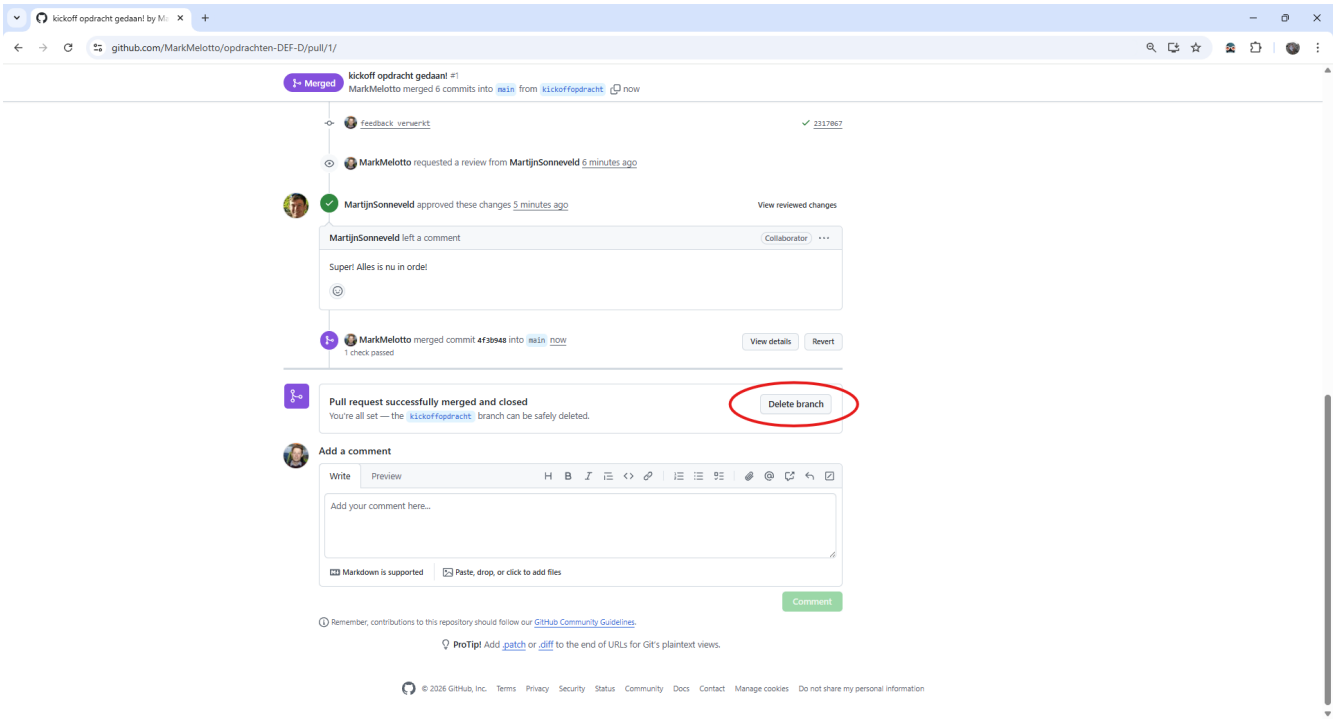


Figure 23: De branch verwijderen.

Automatische Checks

Wij hebben een paar automatische checks toegevoegd, de TA's kijken pas na als al die checks zijn goed gegaan. Vraag dus ook pas voor een review als de checks goed zijn gekeurd. Als de checks niet goed zijn, lees goed wat er mis is, pas dat aan in je opdracht, *commit & push* opnieuw, dan gaan de checks opnieuw kijken of het nu wel in orde is.

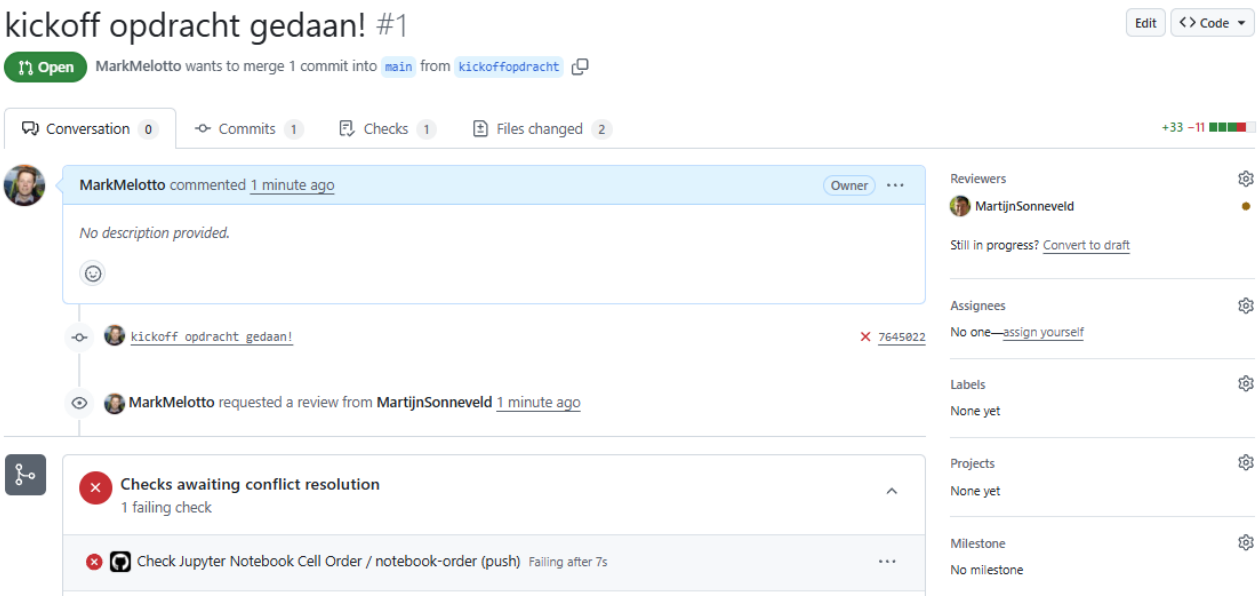


Figure 24: Checks zijn niet goed.

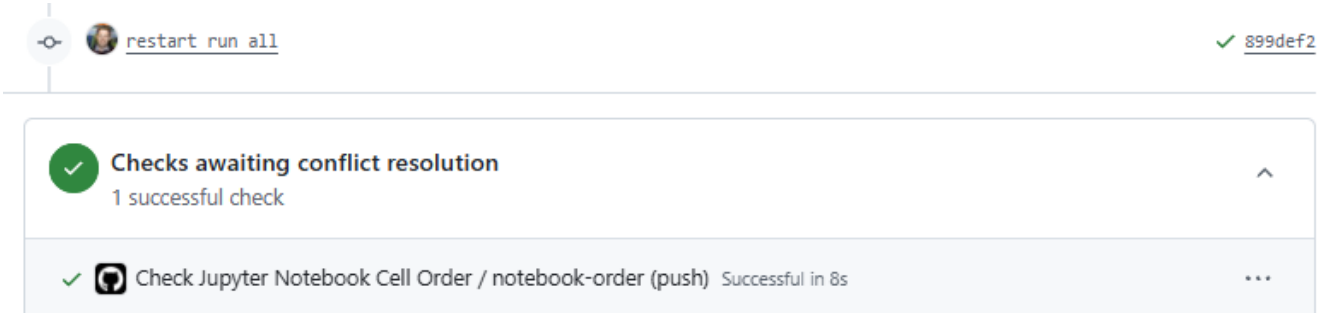


Figure 25: De checks zijn goed.

Deel je werk met je groepsgenoot

Als je je werk wilt delen met je groepsgenoten, zodat hun *repo* ook geüpdate is met jullie opdracht, kan dit ook via een *PR*. Maak dus nog een *PR* aan vanuit jouw *branch* waar je in hebt gewerkt, naar de `main` branch van de forks van je groepsgenoten. 1. Maak dus nog een *PR* aan en kies hier je groepsgenoten:

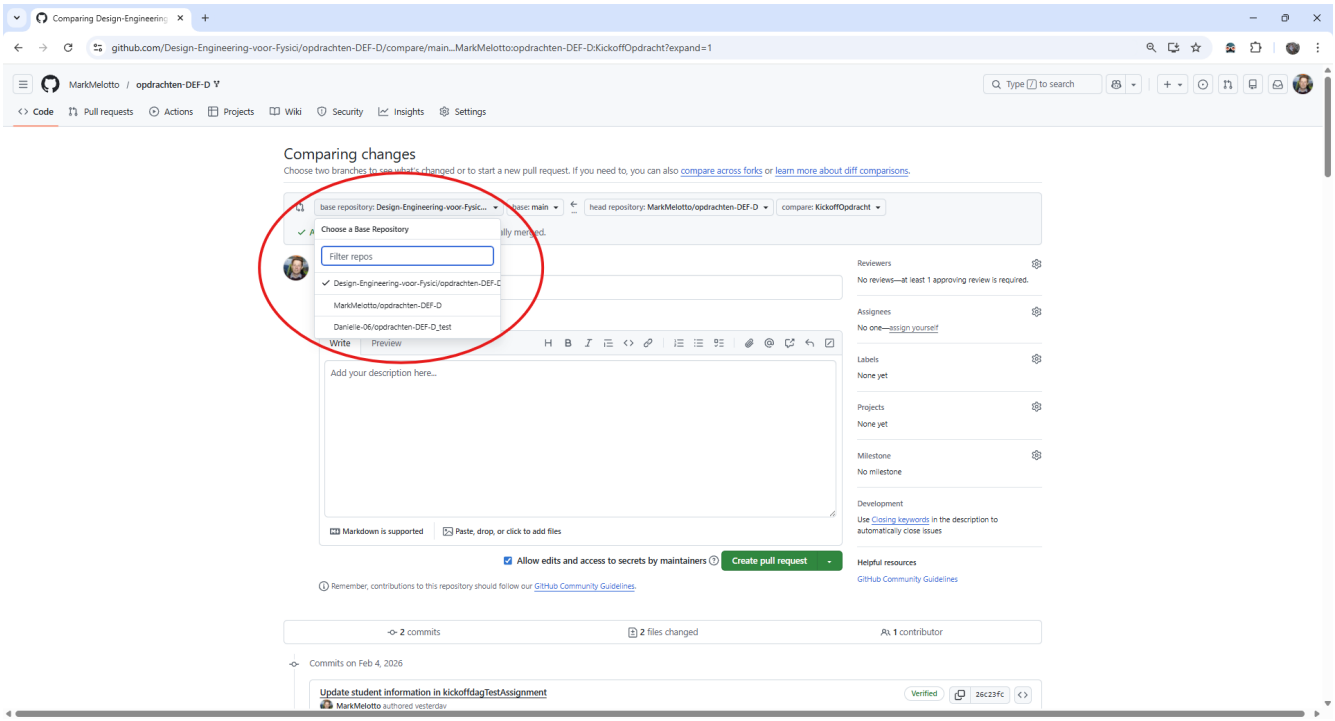


Figure 26: Werk delen met je groepsgenoot.

2. Mijn groepsgenoot is Danielle, ik push mijn `opdracht` *branch* naar haar `main` *branch*:

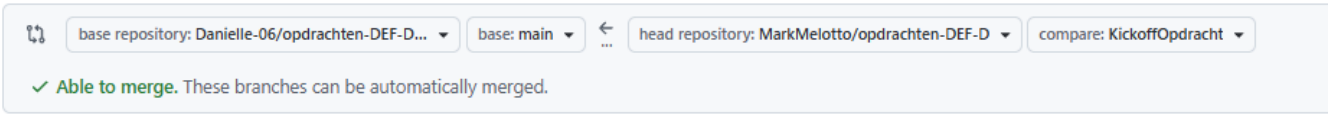


Figure 27: Pull request naar je medestudent.

3. Zij moet dan ook de reviewer zijn, samen met de hoofdTA, en de *PR* pas accepteren als de TA de normale *PR* heeft goedgekeurd.

Git Conflicts

Dit is iets wat jullie waarschijnlijk gaan meemaken, hier leggen we uit hoe je er mee om moet gaan. *Conflicts* kunnen best tricky zijn, al kom je er niet uit, vraag het aan je TA!

Een *conflict* gebeurd als 2 *pushes* dezelfde regels hebben aangepast. Het voorbeeld dat volgt, is alleen voor als je *branch* en je *target branch* aan dezelfde file hebben gewerkt. Git vraagt dan om een keuze te maken tussen de versies.

Belangrijk al kom je er niet uit, vraag hulp aan je TA.

Voorbeeld

GitHub laat je altijd weten als er een *conflict* is:

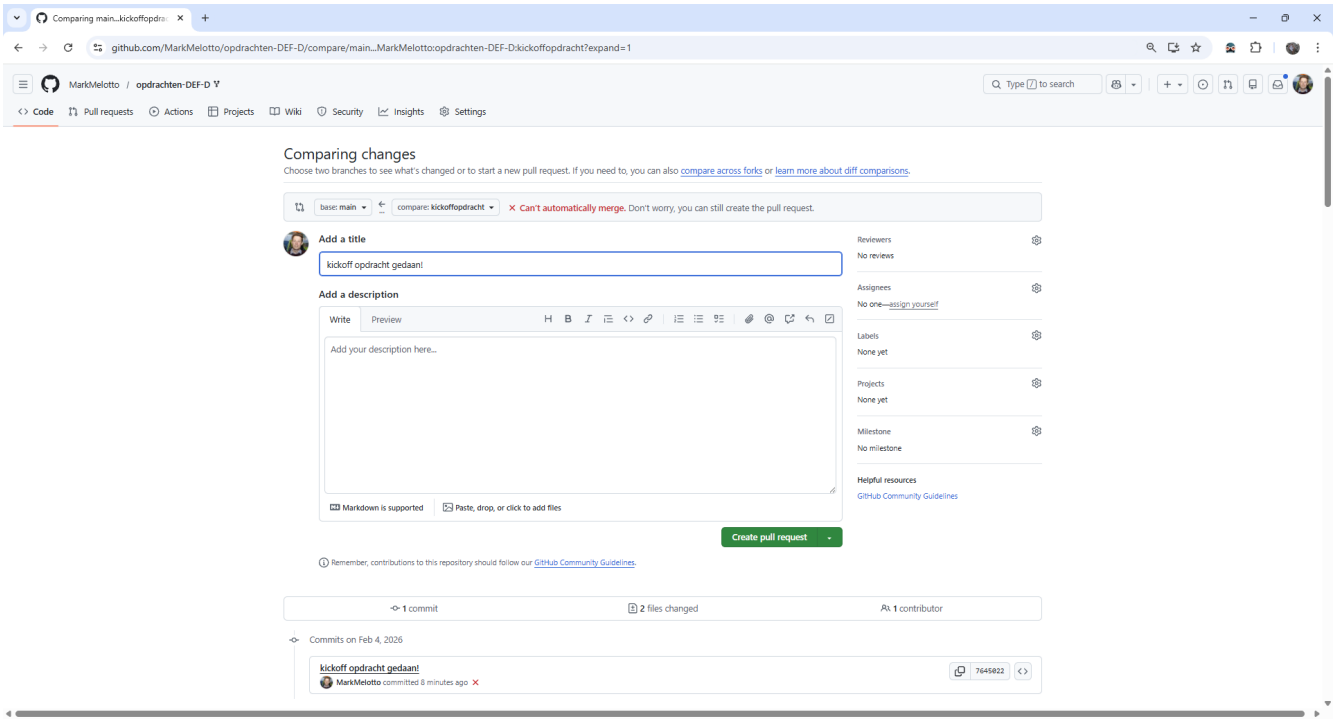


Figure 28: Er gaat een conflict komen.

Onderaan de PR zie je de knop: `Resolve conflicts`, druk hierop!

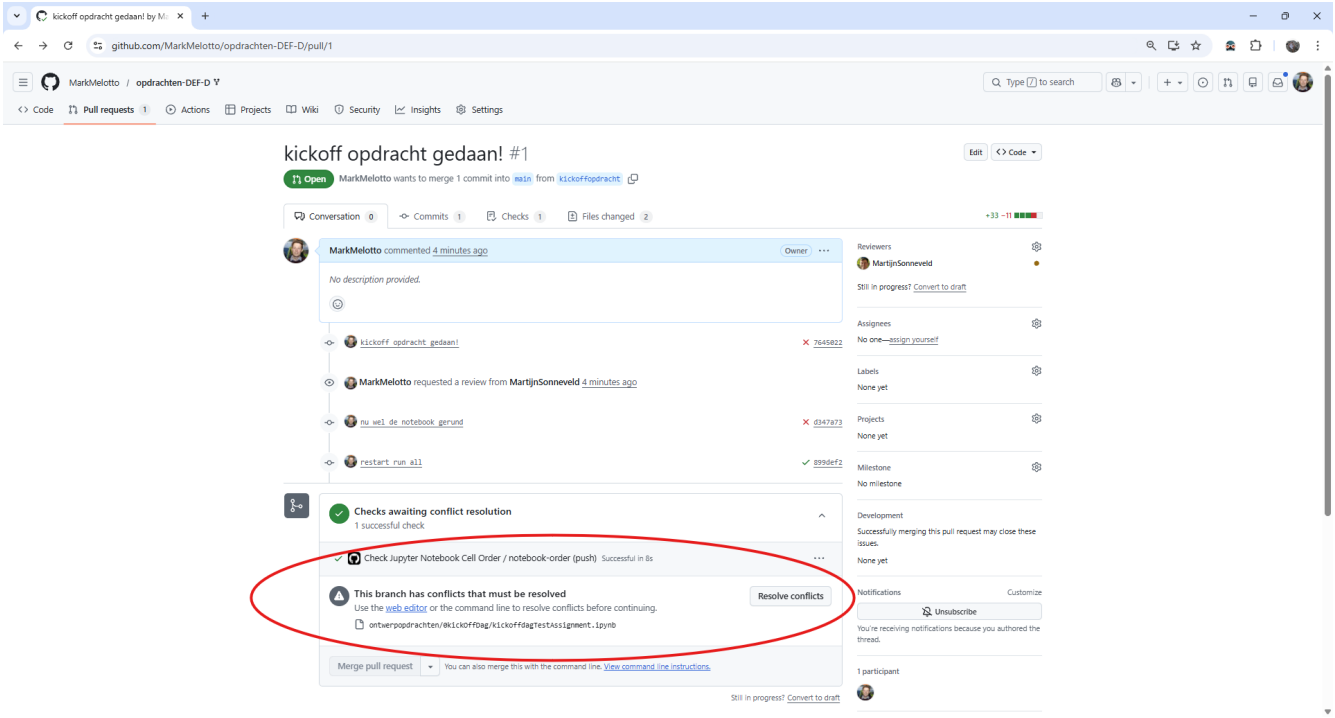


Figure 29: Resolve conflict knop.

Dan kom je uit bij de file die *conflicten* geeft, eerst het plaatje dan de uitleg. Een notebook up

GitHub ziet eruit als een tekst file, dus wat je hieronder ziet is een notebook cell.

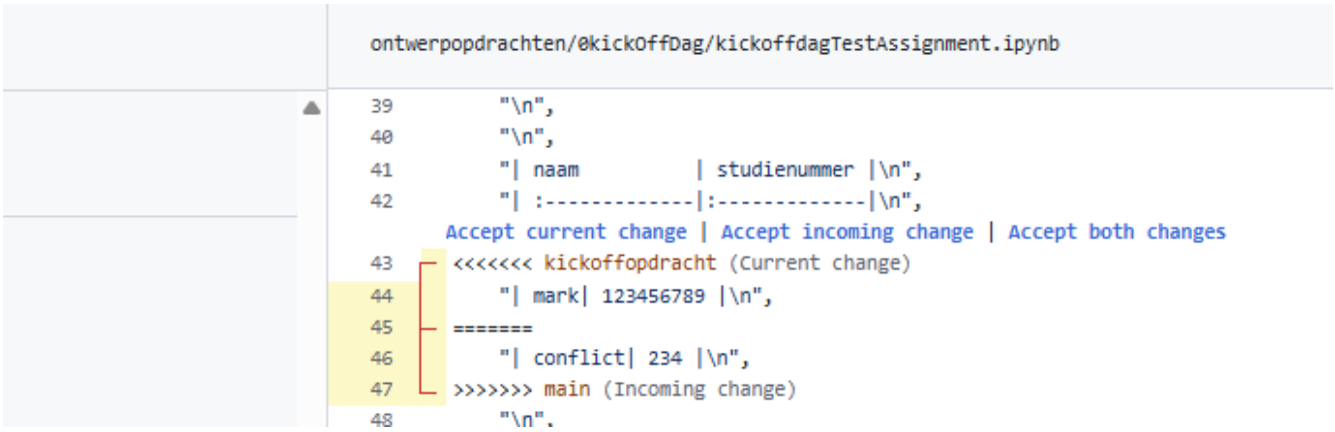


Figure 30: Een conflict ziet er zo uit.

De grote rode lijn aan de linkerkant en rechts van de regel-getallen laat zien om welke regels het gaat. <<<<<<< tot en met ======= gaat om jouw deel en alles daarna van de plek waar naartoe: *merged* of *pusht*! (in dit geval mergen) Na ======= is wat er al stond. Tegenwoordig heeft GitHub de opties om versies te accepteren: 'Accept current change | Accept incoming change | Accept both changes' Die spreken voor zich, maar pas op met accept both changes, soms is het makkelijker handmatig even alles goed te zetten.

Als alles is 'geresolved' ziet het er zo uit en moet je dit nog **committen**!

Figure 31: Het conflict is gefixt.

Fork Syncen

We houden de *fork* met de originele *repo* intact zodat wij nog eventuele verbeteringen kunnen doorzetten. Dit scheelt iedereen moeite en geeft ons een snelle oplossing om eventuele foutjes snel te verbeteren.

Hoe dit werkt laten we hier zien:

1. Omdat wij dus nu zelf aan de opdrachten al hebben gewerkt, werkt de knop `Sync Fork` niet… We drukken dus op de '*commits* behind' knop in onze *fork*:

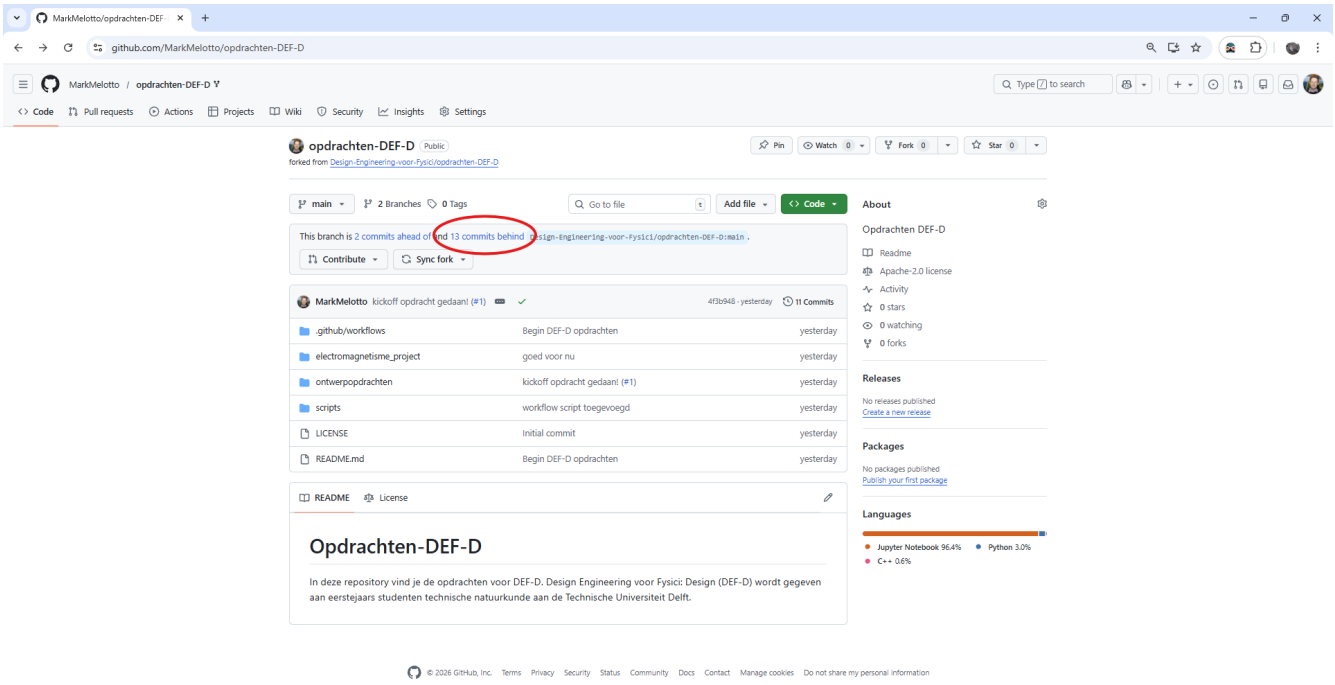


Figure 32: Je *fork* updaten. Klik op de '*commits* behind'.

2. Je komt dan bij het volgende scherm, check dat je van head: DEF naar base repo: jouw eigen repo gaat en druk op `create pull request`:

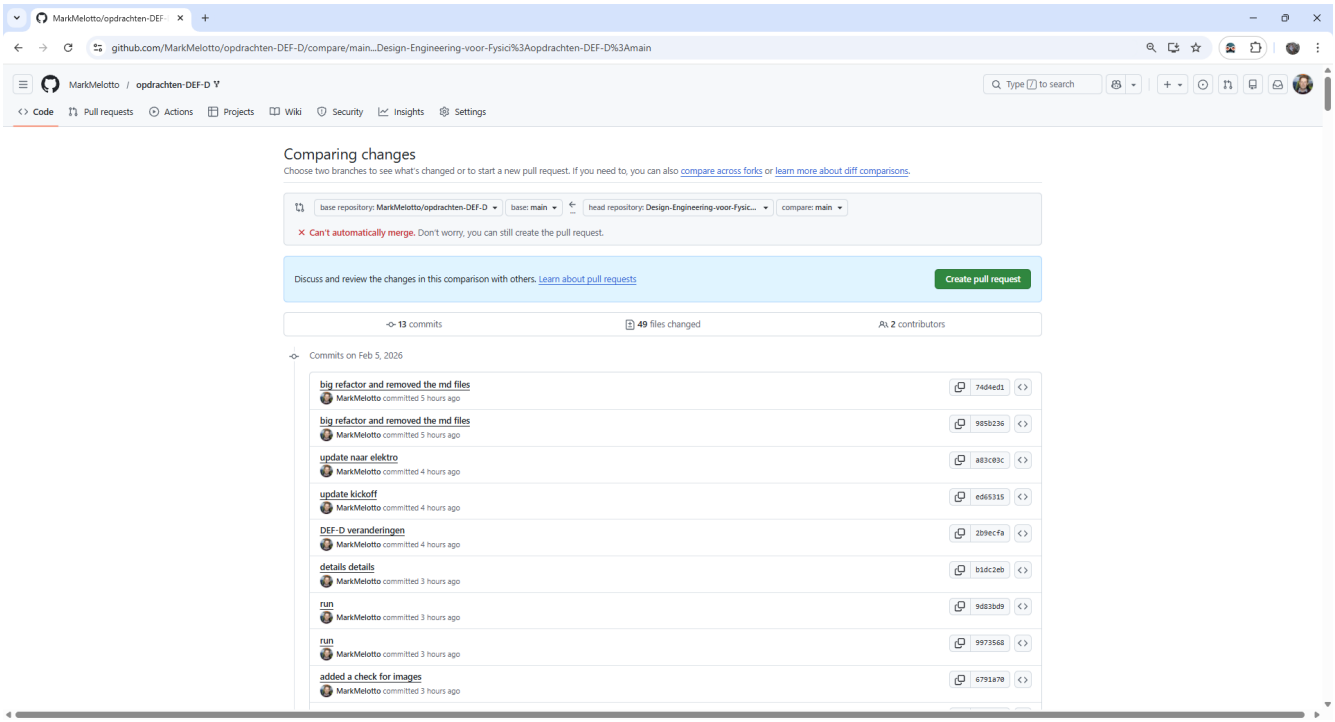


Figure 33: Check de *base repo*!

3. Dubbel check de *PR*, geef een goede titel en vraag voor reviewers de hoofdTA's aan (voor

dit voorbeeld alleen Martijn):

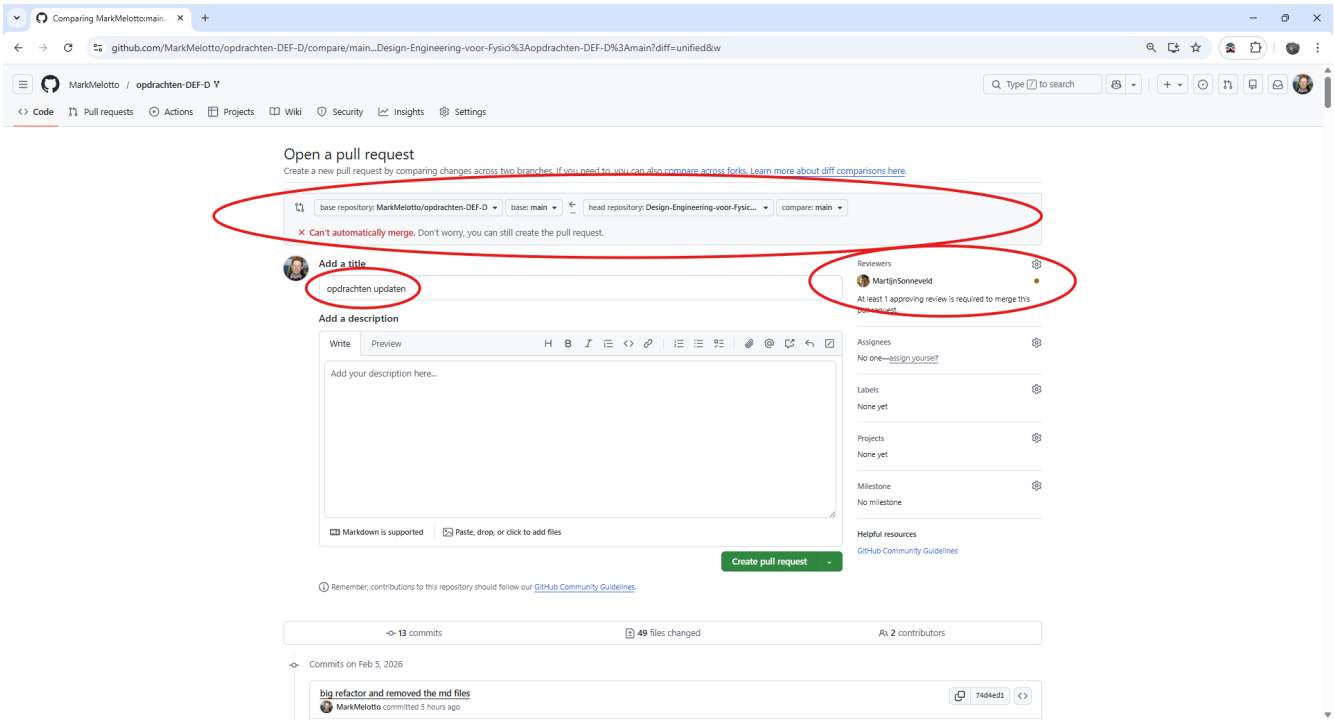


Figure 34: De 'fork' *pull request* aanvragen.

4. Dit ziet er dan zo uit:

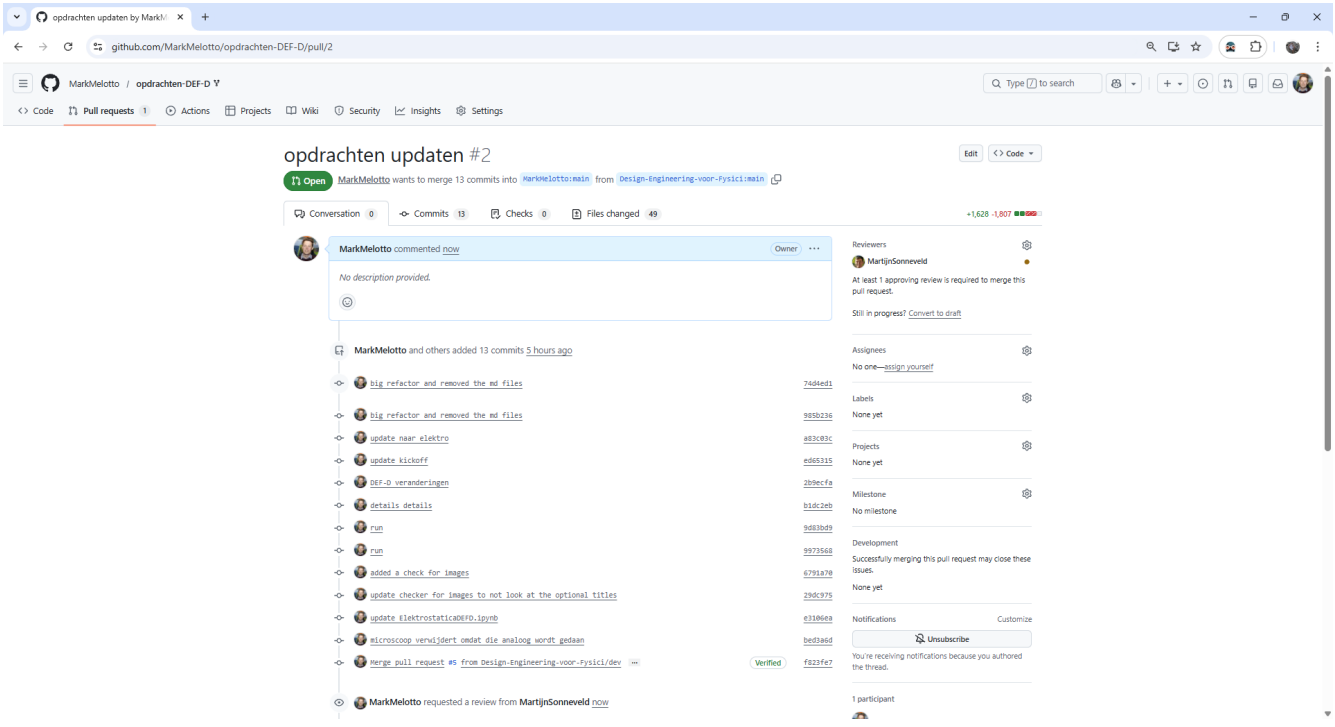


Figure 35: Zo ziet de *PR* er dan uit. De TA gaat het vanaf hier overnemen.

5. Vanaf hier regelt de TA dat alles goed komt te staan!