

Git tutorial

Door Steven Buitenkamp C4c

Git is een super handig programma voor versiebeheer en samen te werken op afstand.

Ik wil jullie graag uitleggen hoe de workflow van Git in zijn werking gaat. Van het aanmaken van een repo tot in het productieproces.

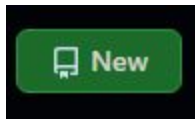
Hiervoor moet je wel eerst Git lokaal geïnstalleerd hebben.

[Download link voor Git](#)

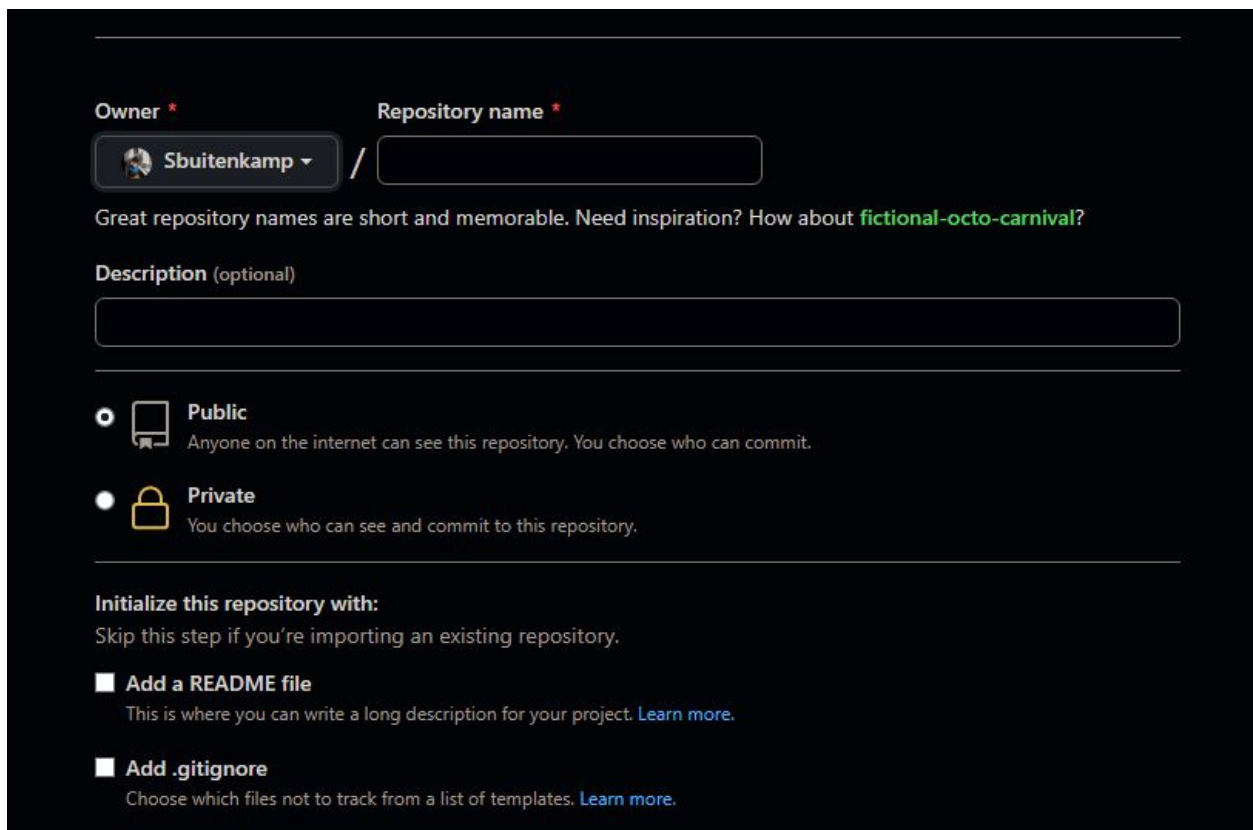
Opstarten

Als je voor het eerst begint met een project moet je natuurlijk een repository (of een repo in het kort) aanmaken op GitHub.com of een andere git site (GitLab of GitBucket zijn goede alternatieven als je iets tegen github hebt). Je gaat naar GitHub, logt in en naar je overzicht van repositories.

Dan druk je op deze knop om een nieuwe repository te maken.



Dan kom je vervolgens bij dit scherm:



Owner ^{*} Sbuitenkamp / Repository name ^{*}

Great repository names are short and memorable. Need inspiration? How about **fictional-octo-carnival**?

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☒ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

Geef een naam aan je repository en kies of deze publiek of privé is. Als hij publiekelijk beschikbaar is dan kan iedereen hem zien en aanpassingen aanbevelen maar jij moet deze alsnog goedkeuren. Ik raad af om een publieke repo te maken als jij met gevoelige data werkt.

Als je een privé repository hebt kun je maar met 4 personen tegelijk werken tenzij je een github premium abonnement aanschafft.

Ik raad aan dat je een README en een .gitignore laat genereren door GitHub. Bij de .gitignore kun je aangeven wat voor project je hebt en hierbij kun je dus een preset kiezen op basis van je project. Unity staat ook bij deze lijst.

Maar wat doet een .gitignore dan? In een .gitignore geef je aan welke bestanden niet meegenomen mogen worden als je eenmaal code aan het pushen bent. Dit is handig als je grote hopen aan files hebt die de ander niet nodig heeft of op een andere manier kan downloaden zonder dat je repo een gigantische grootte krijgt.

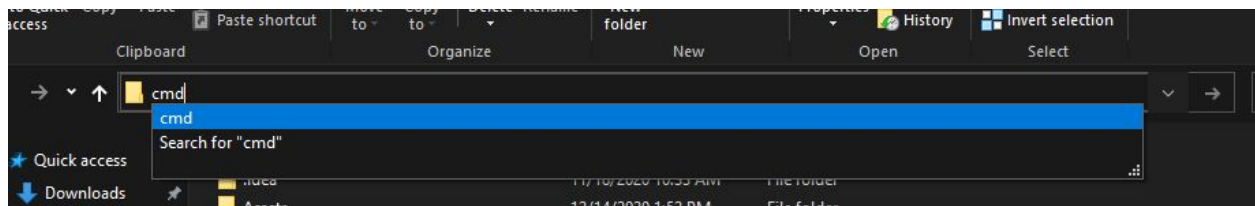
Kies de instellingen die je wilt en klik dan op “Create repository”.

Als je geen README of .gitignore hebt aan laten maken dan staan er een paar aanbevelingen van GitHub over de volgende stappen die je kan doen, deze kun je voor nu wel negeren.

We hebben nu een repo op GitHub staan maar je kan er nog geen code naar toe pushen.

Daarom moeten we nu een van 2 dingen doen:

Als je wel bestanden aan hebt laten maken dan maak je een folder aan waar je het project wilt hebben op je computer. Dan open je een command prompt door in de navigatiebalk “cmd” te typen zoals dit:



Druk op enter en er wordt een command prompt gestart die automatisch genavigeerd is naar de folder waar je je op dat moment in begeeft.

Op mac doe je Finder>Services>Open new terminal in folder.

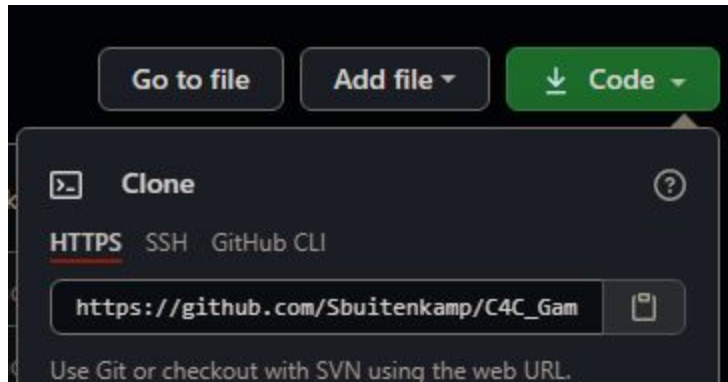
Als je in de verkeerde folder zit kun je deze navigeren met het command “cd path/to/directory”.

Als je een folder terug wilt doe je “cd ..” in de command prompt.

Als je een folder naar voren wil dan doe je “cd ./foldernaam”.

Als je in de juiste folder zit met je command prompt dan gebruik je het commando “git clone url” en url vervang je met de url van je repo.

De clone url kun je vinden door deze url te kopiëren vanuit jouw repo pagina:



Dan maakt git een folder aan met de naam van je repo en zet alle bestanden erin (voor nu dus een .gitignore en een README). Dan voer je het command “cd projectnaam” in om naar die folder te gaan en vanuit daar kun je met git gaan werken in die folder.

Als je geen files hebt laten generen is het nog niet mogelijk om te clonen. Wat je dan gaat doen is een nieuwe folder aanmaken, een command prompt openen in die folder net zoals bij de vorige stappen en dan voer je het command “git init” uit. Wat je dan ook nog moet doen is het command “git remote add origin url” en url vervang je ook weer met die url die je van github haalt (vergeet de .git op het einde niet!).

Tip: je kan dit doen in de folder van je Unity project zodat je straks niet alles hoeft te verslepen.

Nu ben je ook klaar om te werken met git op deze repo.

Files naar git sturen

Nu we alles gereed hebben kunnen we beginnen met files op te sturen. Sleep nu je Unity project files in dezelfde folder of als je slim bent heb je je git init gedaan in je Unity project folder.

Als je alleen werkt of als je alle bestanden voor het eerst op GitHub zet kun je de eerste stap over branches overslaan maar als je samenwerkt moet je branches gebruiken anders krijg je problemen met je groepsgenoten.

(Maar 1 iemand hoeft een unity project aan te maken en deze te pushen naar GitHub, de rest kan de vorige stappen gebruiken om het project te clonen.)

Om een nieuwe branch te maken en er meteen naartoe te gaan gebruiken we het commando `git checkout -b branchNaam`

`git checkout` is het commando om van branches te switchen en de prefix `-b` maakt een nieuwe branch aan (`-b` werkt alleen als de branch nog niet bestaat).

Als je samenwerkt vind ik het persoonlijk altijd handig om mijn initialen voor de branch te zetten zodat mijn teamgenoten weten aan welke branches ik gewerkt heb. Dus dan doe ik het commando als volgt:

```
git checkout -b sb/exampleBranch
```

Nu kun je al je aanpassingen doen en deze staan dan altijd op deze branch.

Als je dan weer terug gaat naar de branch main (vroeger heette dit master maar dat is sinds kort veranderd om een of andere reden) door `git checkout main` te doen dan zul je zien dat je veranderingen niet meekomen. Dat is het punt van branches, je kan nu alle verschillende werkzaamheden opsplitsen in dezelfde folder op je pc en in de cloud.

Als je klaar bent om je bestanden naar git te sturen dan doe dat volgens deze workflow:

- `git add --all`
- `git commit -m "commit message"`
- `git push -u origin branchName`

Eerst voeg je de bestanden toe met `git add`. Je gebruikt de flag `--all` om aan te geven dat alle bestanden in het hele project (naast de folders en bestanden die in je `.gitignore` staan) meegenomen moeten worden.

Dan commit je de bestanden met `git commit`. Je gebruikt de `-m ""` prefix om snel een message te kunnen typen ipv met VIM een heel verhaal te moeten typen. VIM is niet erg gemakkelijk om te begrijpen voor beginnende programmeurs. Deze commit message komt ook bij de pull

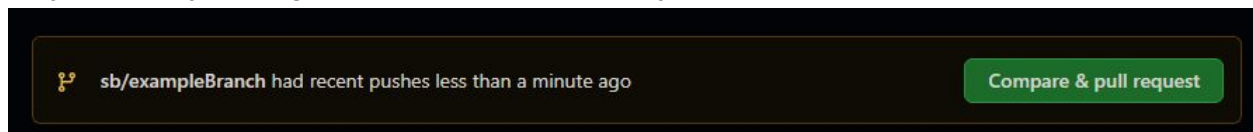
request te staan. Geef deze dus een goede beschrijving zoals “Fixed bug in walking script” of “Jumping script added”.

Dan als laatste push je de bestanden die je gecommit hebt naar de github pagina. Geef de naam van de branch waar je op aan het werken bent mee aan het command dus in mijn vorige voorbeeld zou dat zijn: `git push -u origin sb/exampleBranch`

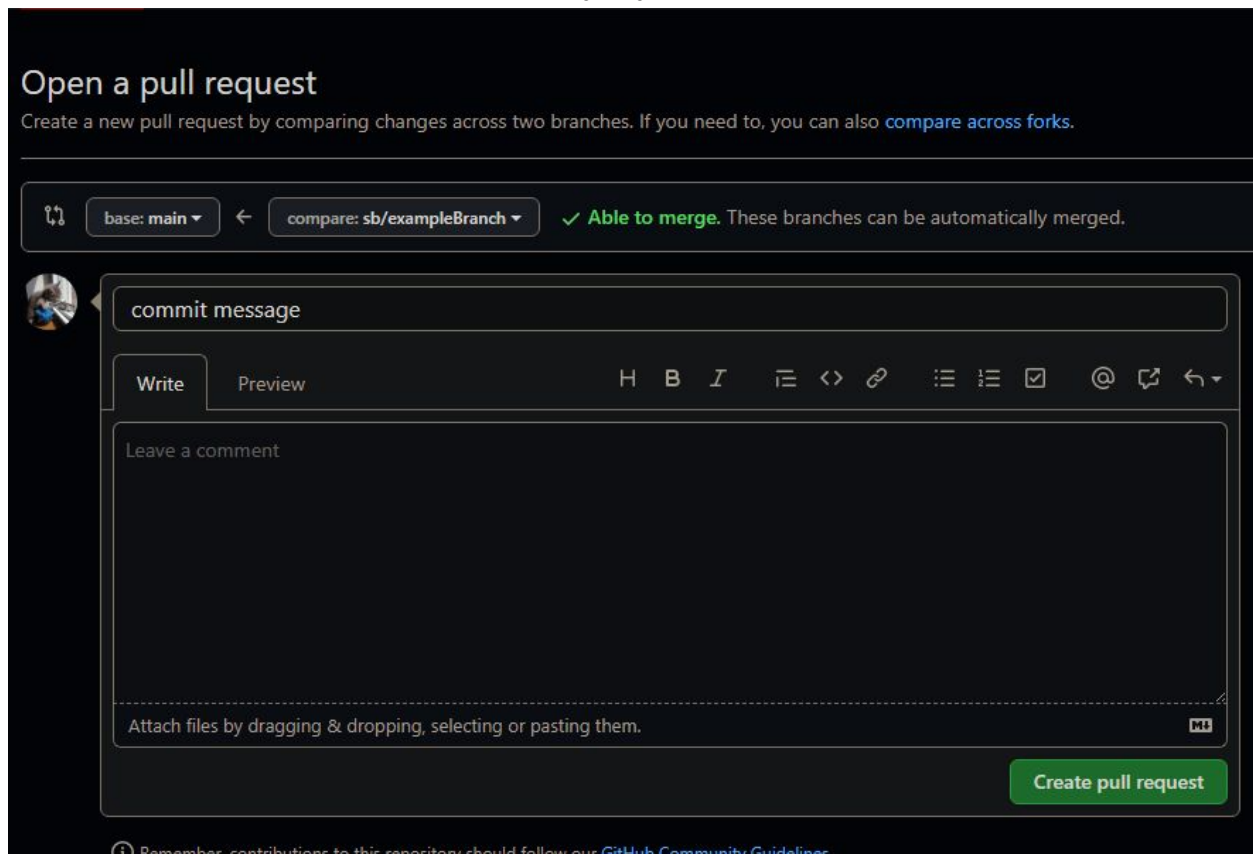
Als je in je eentje werkt kun je gewoon `git push -u origin main` gebruiken. Als je naar main pushed hoeft je ook niet te mergen op de site.

De veranderingen mergen naar je main branch

Als je nu naar je repo gaat op GitHub.com dan zul je dit zien:

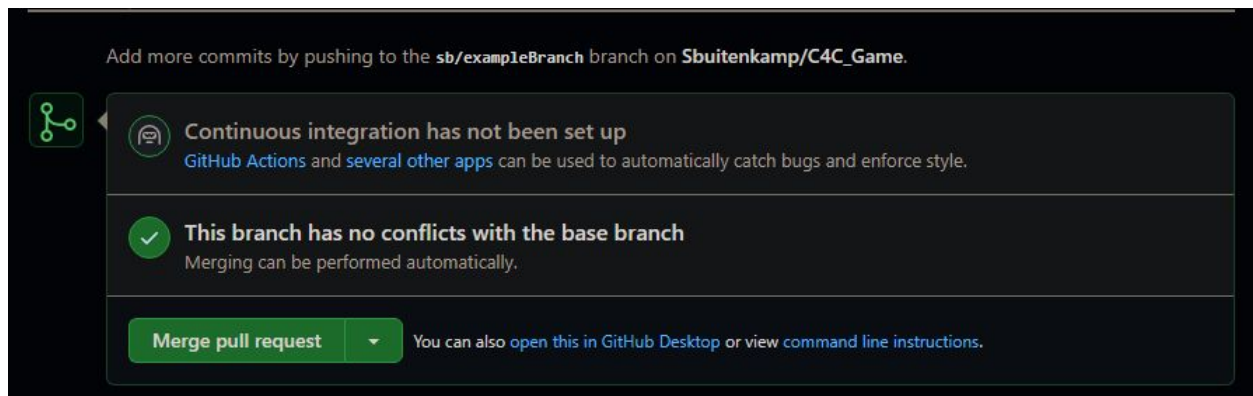


Druk op compare & pull request en dan kom je bij dit scherm:



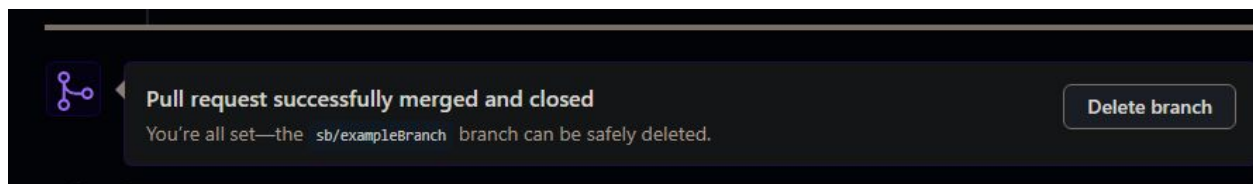
Hier kun je de target branch aanpassen (hou dit maar gewoon op main tenzij je weet wat je doet maar dan zou je dit toch niet meer lezen). Je kan ook je commit message aanpassen en er extra commentaar aan toevoegen als je dat wilt.

Dan druk je op create pull request en zie je dit laatste scherm:



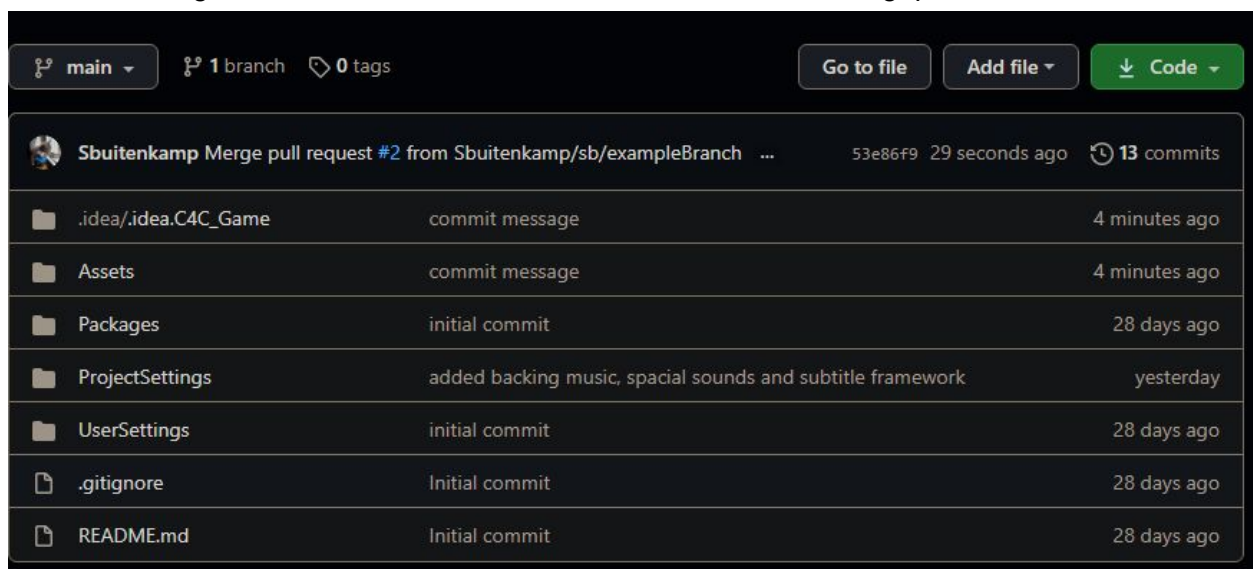
Als je in hetzelfde bestand als iemand anders aan het werken bent geweest kan het nu zijn dat je eerst de merge conflicts moet resolvable, dit is heel gemakkelijk en github heeft hier een editor die je daar super goed bij helpt.

Als alle conflicts resolved zijn druk je op merge pull request.



Delete nu de branch om je repo schoon te houden.

Als je nu terug gaat naar het hoofdscherm van je repo zie je allemaal bestanden staan met de commit message van de laatste commit die de bestanden heeft aangepast:



De veranderingen weer lokaal krijgen

Nu heeft iedereen toegang tot de gemergde versie van je project maar jij moet hem zelf ook nog ophalen, want misschien heeft een groepsgeenoot ook nog wel veranderingen gemaakt. Dit doen we als volgt:

In onze commandprompt doen we eerst “git checkout main”. Dan doen we “git pull” om alles naar onze main te halen (want de veranderingen stonden alleen nog op de branch). Als alles naar wens is kun je als laatste nog de lokale branch verwijderen door “git branch -D branchName” te doen. In mijn geval is dat dus git branch -D sb/exampleBranch.

Je kan nu weer vanaf de stap met de branches beginnen en dit blijven doen totdat je klaar bent met je project.

Het beste is om per feature een pull request te maken.

(Ga niet om elke lijn code een commit maken tenzij het een bugfix is.)

Als er nog vragen zijn dan hoor ik het graag en anders veel success met programmeren!