Versiebeheer Hand-out 2

In de vorige hand-out zijn een aantal concepten langsgekomen zoals een repository, GitHub, Pushes, visibility, gitignore, pushes en commits. Uit het resultaat van hand-out 1 kwam een repository waar je werk kon uploaden en bewaren. In hand-out 2 gaan we kijken naar het volgende:

- Cloning
- History
- Collaborators
- Danger zone
- Naming conventies

Deze hand-out bouwt voort op de vorige maar kan gebruikt worden in principe bij iedere Repository op GitHub.

Inhoudsopgave

Opdracht 1: Local en remote Repos (15 minuten)

Git is ooit begonnen als een versiebeheer systeem dat enkel lokaal werkte. Hiermee kon je op je computer een project opzetten met versiebeheer en dan via de command line nieuwe versie plaatsen en oude ophalen. Dit heet een local repository.

Op den duur werk het belangrijk om gebruikers te laten samenwerken en dan was het belangrijk dat iedereen bij deze local repository kon. Deze functionaliteit werd toegevoegd waardoor je via een lokaal netwerk de originele repo had staan bij de eigenaar en dat een clone gemaakt werd door de andere deelnemers die lokaal bij hen zou staan.

Het probleem hiermee is dat mensen buiten het netwerk moeilijk eraan konden werken en moesten wachten tot ze op hetzelfde netwerk zaten. Hier komt een digitale repository op een server bij kijken die vanuit het gehele internet bereikt kan worden indien het juiste account het probeert te bereiken. Dit is hoe de remote repositories zijn ontstaan. Doordat de originele repository op een server staat kan iedereen die gemachtigd is bij de repository en er aanpassingen op maken. Dit bevorderd samenwerking en zorgt ervoor dat iedereen in een project dezelfde code heeft.

Tegenwoordig maken we nogsteeds gebruik van een lokale repository maar dit is de clone waarin gewerkt wordt en vergeleken wordt met de remote repository om een push te doen.

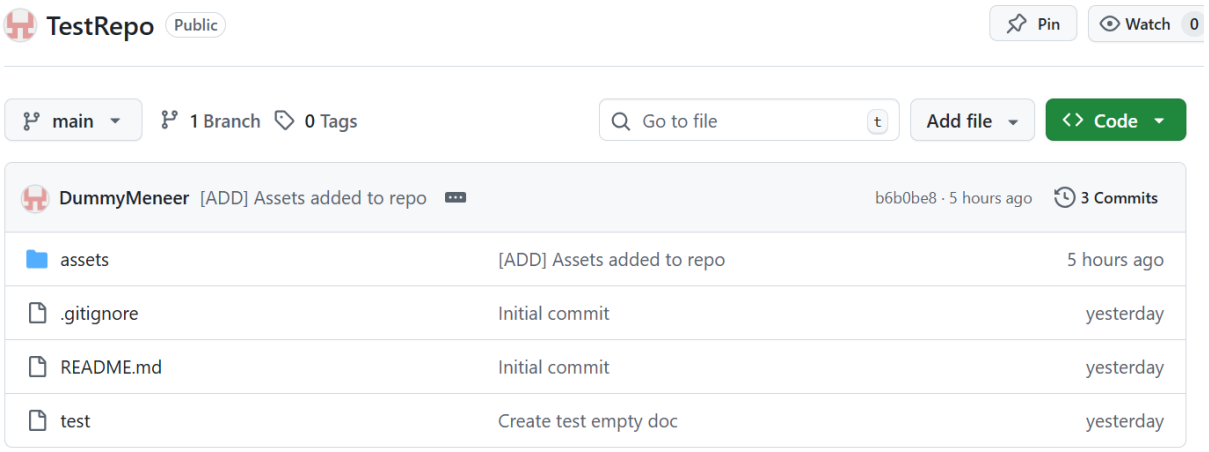
Opdracht 2: Clone je repo en zipfiles (5 – 10 minuten)

Het clonen van een Repo is nodig als je de code wil hebben op je device waar het nog niet op beschikbaar is. (Bijvoorbeeld als je je laptop opnieuw moet installeren) Het clonen van een repo is vergelijkbaar met het downloaden van een document van een cloud storage, alleen clonen heeft enkele verschillende routes:

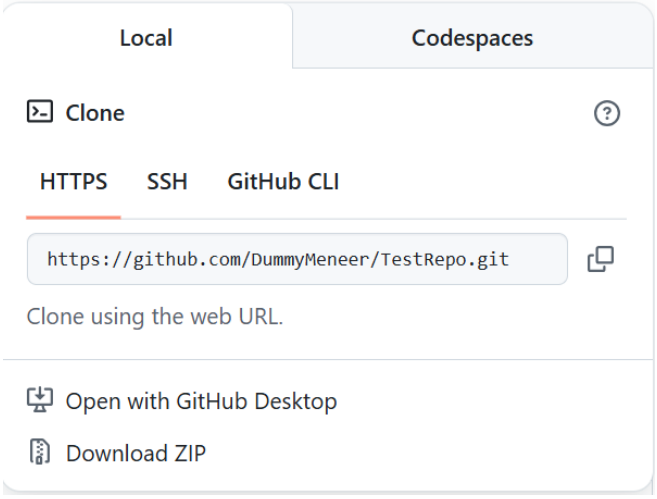
- Downloaden van een Zipfile
- Via GitHub Desktop
- Via een http link
- Via een SSH key
- Via GitHub command line (CLI)
- Via CodeSpaces (Wordt niet behandeld)

Downloaden van een zipfile:

Op de Hoofdpagina van je repo klikken we op de groene knop met "<>Code" dit opent een menu met daarin een aantal opties.



De eerste selectie is tussen local en codespaces. Codespaces is een cloud ontwikkelplatform waar je zonder een lokale opslag aan kan werken, deze wordt niet verder behandeld in dit vak aangezien de ervaring met GitHub centraal moet staan. Local bevat een aantal opties die toelaten om op je lokale apparaat te zetten.



Voor deze opdracht hoef je alleen te kijken naar de download zip knop. Dit start het proces om de repo in een zip bestand te plaatsen en dan te downloaden.

Dit zip bestand kan dan uitgepakt worden en mee gewerkt worden.

De andere opties:

Zoals genoemd zijn er vele manieren van een clone maken.

Sommige tools die met GitHub werken kunnen aan de hand van een https link een clone maken, andere gebruiken een SSH key, dit is een uniek lang wachtwoord wat gekoppeld is aan je repo. Deze applicaties kunnen voor je pushen, pullen en de repo clonen indien ze er toegang tot krijgen.

Een andere manier is de GitHub Command line of CLI. Dit werkt op een vergelijkbare wijze als de command line in CMD of in Visual Code echter is deze specifiek voor GitHub bestemd. Dit laat je toe om zonder “zware” visuele programma’s. Deze optie wordt vaak gebruikt voor hardware die niet veel ruimte heeft voor visuele programma’s, denk aan een kleine processor of een simpele board. CLI is handig om ervaring mee te hebben maar zal niet veel gebruikt worden tijdens het vak.

Als laatste is er de optie voor GitHub Desktop, dit is de officiële desktop applicatie van GitHub die je toelaat om GitHub te benutten zonder dat je met de browser hoeft te werken. De desktop applicatie houdt overzicht van je lokale repos en laat je toe om van deze de GitHub repo te updaten.

GitHub Desktop en andere visuele applicaties mogen gebruikt worden vanaf week 5 en worden nog even toegelicht in week 8

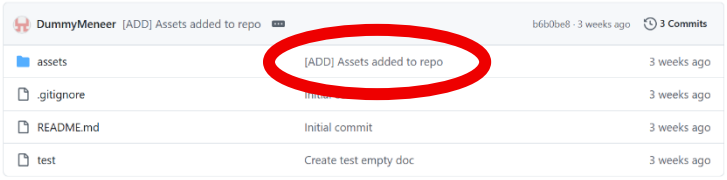
Opdracht 3: History bekijken van code (10 – 15 minuten)

Soms is het belangrijk om een oude versie van code te zien, bijvoorbeeld je hebt een poging gedaan om te optimaliseren maar dat mislukte en control Z gaat niet ver genoeg terug. Dan is het fijn als je terug kan kijken. Een andere situatie is als ander teamlid veel heeft gedaan op het project maar er wordt verwacht om te weten wat er gedaan is, losse commits doorlopen kan je veel vertellen het gemaakte werk.

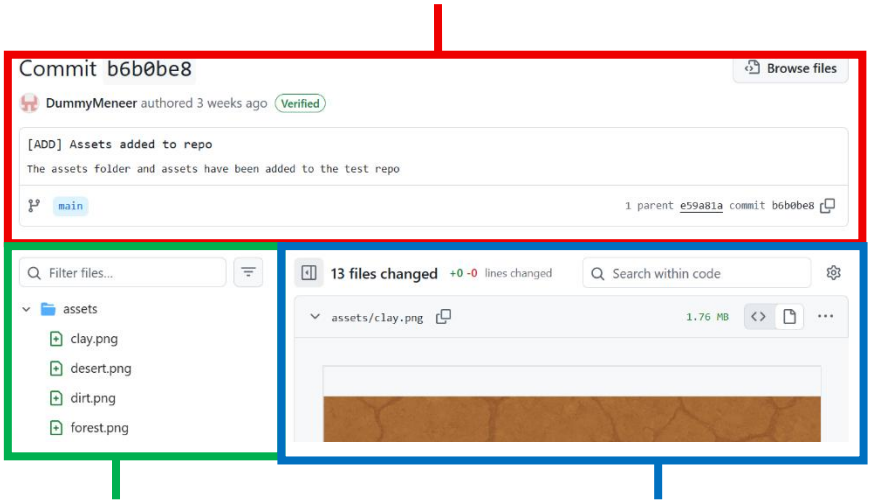
Er zijn 3 manieren om dit in te zien, geen van deze 3 is superior ten opzichte van de anderen. Probeer alle drie de manieren uit en kijk naar wat het pretigste werkt voor jou.

Manier 1: gebruik de commits in het code overzicht van de hoofdpagina van de repo

Klik op de middelste kolom met de commit naam die bekeken moet worden.

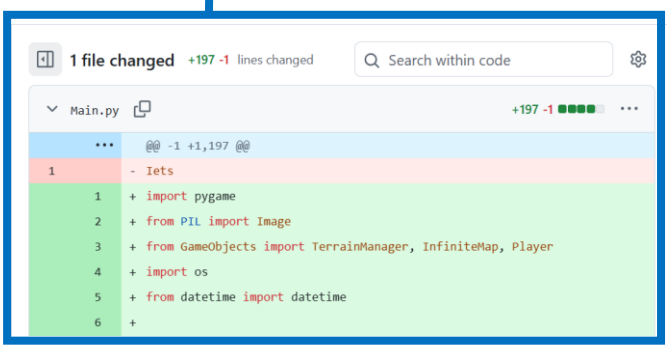


Dit opent het overzicht van de commit. Dit overzicht bestaat uit de Commit details. Dit bevat de naam, code, beschrijving, auteur en leeftijd/datum van de commit. De knop van “Browse files” laat je toe om de repo te zien in de staat van die commit, dit betekent dat code van een commit later niet zichtbaar is.



Dan is er het overzicht van de files die zijn aangepast aan de linker zijde. Een groene file betekend dat het is toegevoegd, een grijze dat er wijzigingen zijn en een rode dat deze verwijderd is.

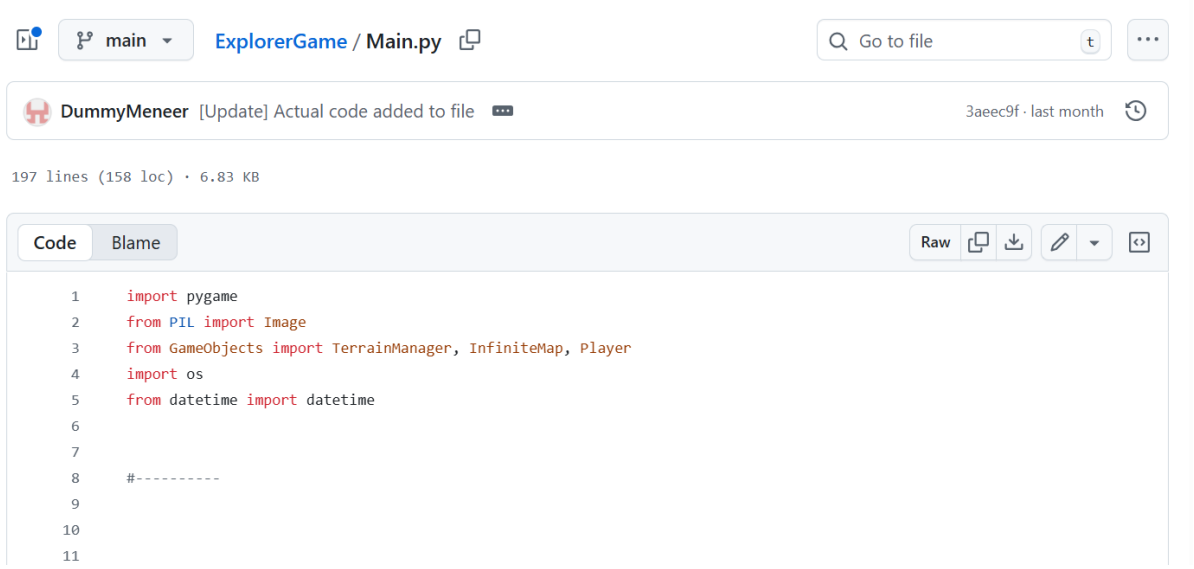
Aan de rechterkant staan de files die zijn aangepast met details erin van hoe het is aangepast. In het voorbeeld hierboven zijn er geen regels gewijzigd dus staat er +0 -0. In het voorbeeld hieronder staat een voorbeeld met regels die zijn gewijzigd. Hier kunnen we aan de rode lijnen zien wat er weg is gehaald en de groene lijnen wat er is toegevoegd.



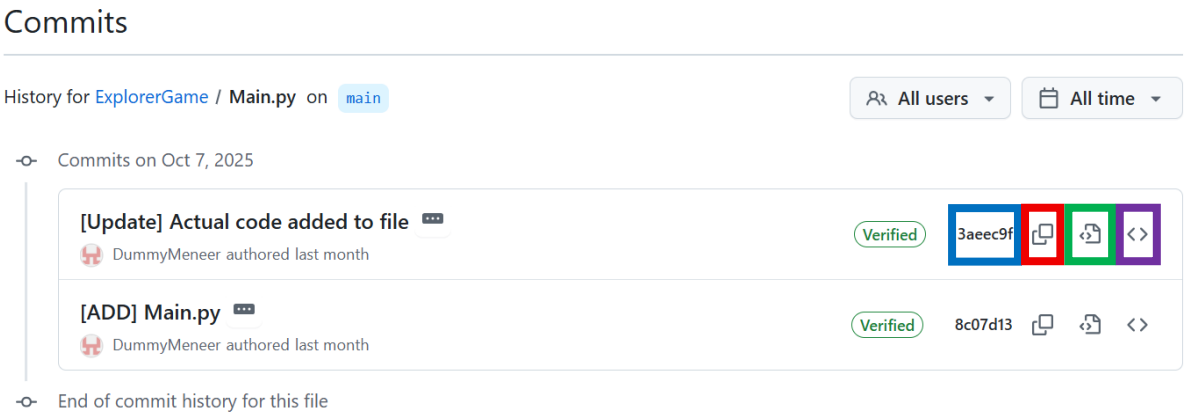
Het voordeel is dat je alle documenten die veranderd zijn in de Commit bij elkaar staan en een overzicht geven van wat er gedaan is.

Het enige nadeel van dit is dat de hoofdpagina alleen per bestand de laatste commit laat zien dus dit heeft als effect dat een echte history van een document niet makkelijk te lezen is.

Manier 2: gebruik het document geschiedenis



Klik op het document waarvan je de geschiedenis wilt bekijken, dit opent de huidige/meest recente versie. Om de geschiedenis te bekijken van dit document klikken we op het klokje rechtsboven. Dit opent de commit history/geschiedenis van het document.



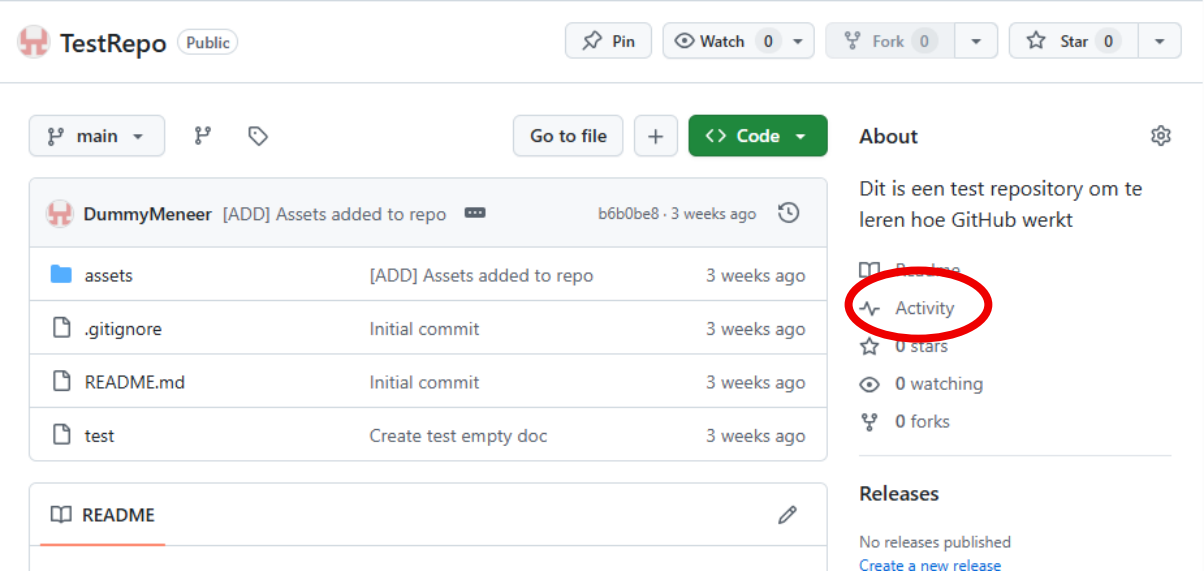
Bij iedere commit staan enkele symbolen aan de rechterzijde. De meest linkse (Blauw) is de commit code/ID, dit is een unieke code/ID die gekoppeld is aan de commit, als je hier op klikt kom je uit op het commit overzicht (Zie manier 1). De tweede (Rood) is de knop om de commit ID te kopiëren, Dit maakt het makkelijker om een specifieke commit te clonen. De derde (Groen) knop is om de code van dit document op dat moment te bekijken, dit laat je toe om eventuele oude code terug te halen mocht het verwijderd zijn tijdens het project. Als Laatste is de meest rechter van de knoppen (Paars), deze knop laat je toe om de repo te bekijken zoals het was op het moment van de git.

Het voordeel van deze manier is dat er heel veel handelingen bij elkaar staan en je toelaten om de op deze wijze de geschiedenis van de repo te doorzoeken.

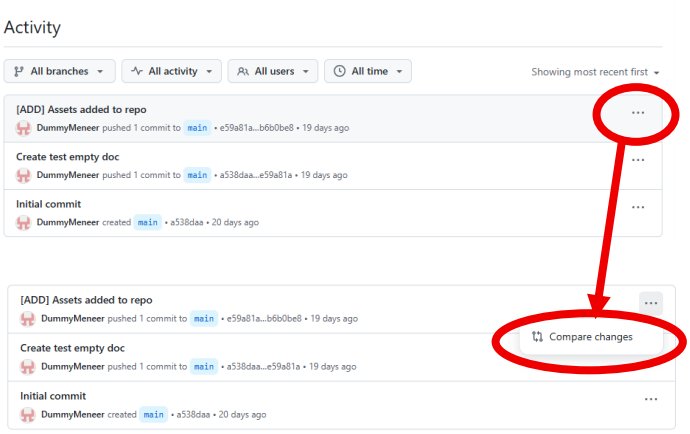
Het nadeel is dat het veel heen en weer geklik kan zijn om de juiste file versie te zoeken.

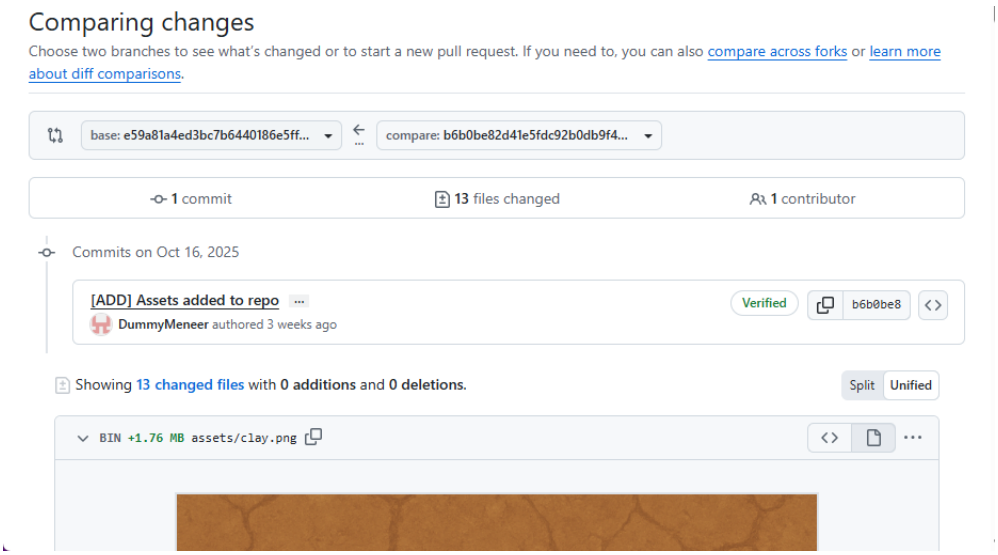
Manier 3: Gebruik activity van de about sectie.

Vanuit de hoofdpagina van de repo ga je naar het about deel aan de rechterkant van het scherm. Daar klik je op het activity kopje.



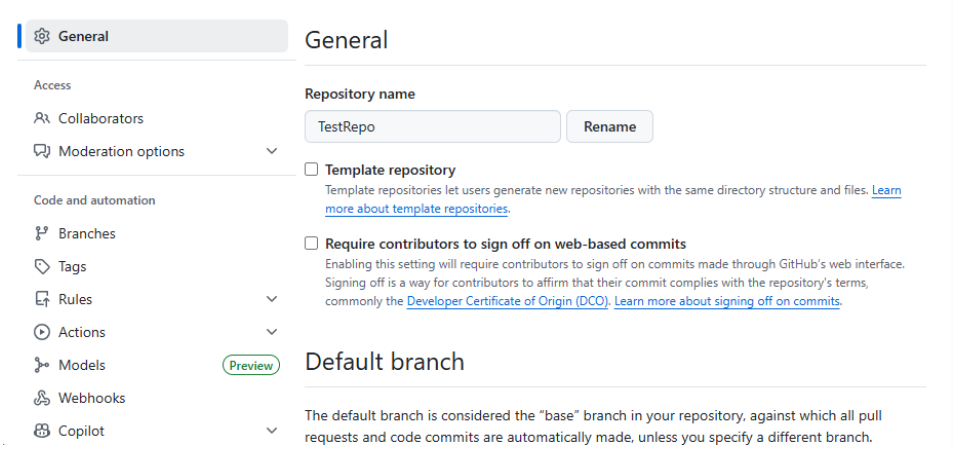
4Dit opent het Activity overzicht. In dit overzicht staan de verschillende commits die zijn gemaakt binnen de repo en de verschillende activiteiten die plaats hebben gevonden. Vanuit dit scherm kan je op de ... knop drukken en krijg je de “compare changes” knop te zien. Klik hierop om een overzicht te bekijken die de verschillen laat zien tussen de gecommitte versie en de versie ervoor.





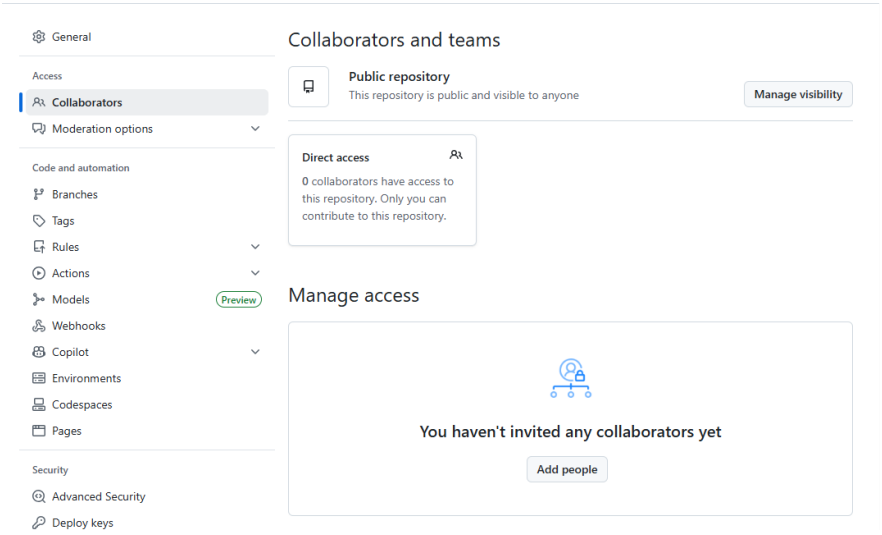
Opdracht 4: Collaborators toevoegen (10 minuten)

GitHub is zeer handig voor groepsprojecten omdat het mensen toelaat om een source van code te hebben. Dit voorkomt dat er zipfiles heen en weer gestuurd hoeven te worden en dat er verwarring kan ontstaan van wat de laatste versie is. Standaard kan enkel de eigenaar iets toevoegen aan een GitHub repo maar bij het toevoegen van een collaborator kan iemand anders ook code toevoegen. Om dit te doen ga je van je hoofdpagina van de repo naar de settings.

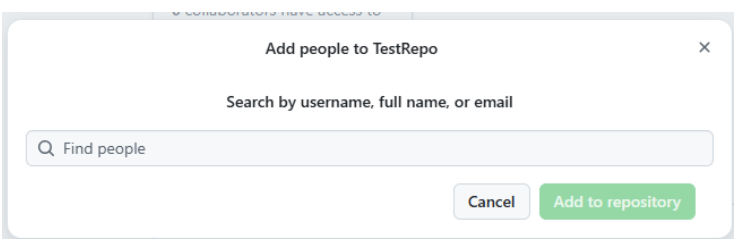


Hier zien we de general settings en een lijst met tabjes met specifieke settings. Voor deze opdracht kijken we naar de collaborators tab, dit is de tweede van boven.

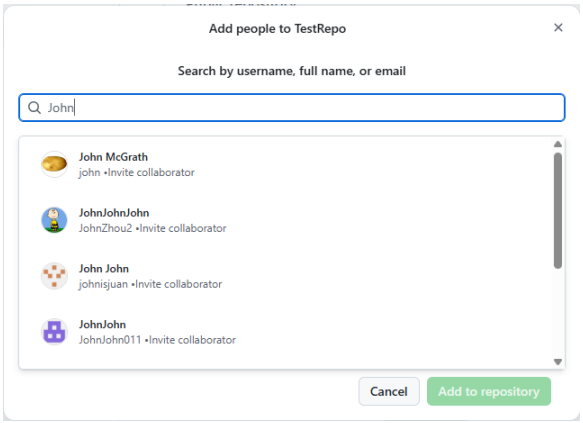
Dit opent de pagina met daarin 2 onderdelen: de visibility/zichtbaarheid van de repo en een access venster.



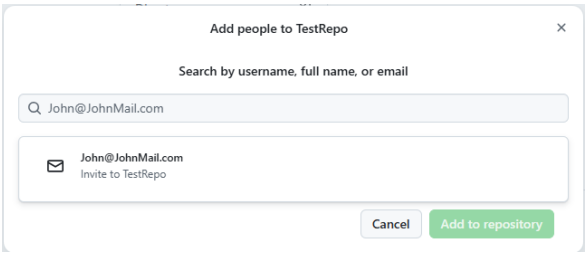
De access is het belangrijkste voor deze opdracht. Als je hier klikt op "Add people" krijg je een boxje te zien waar je de GitHub naam van of email kan invullen van de persoon die je wilt toevoegen.

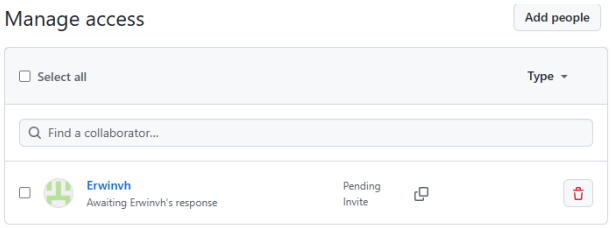


Gebruik je de GitHub naam van iemand dan zie je een lijstje met namen die erop lijken. Zorg ervoor dat je de juiste persoon toevoegt aangezien sommige mensen ander misbruik maken van de toegang die je verleend hebt.



Ja kan ook een uitnodiging sturen naar iemand email adres, dit zorgt ervoor dat de persoon in kwestie zelf kan bepalen met welk account die mee wil werken aan de repo. Dit kan zijn omdat je groepsgenoot een school en Privé GitHub account heeft of omdat je de persoons GitHub naam niet weet.

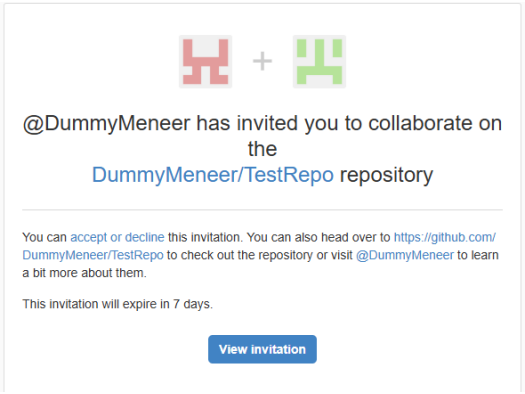


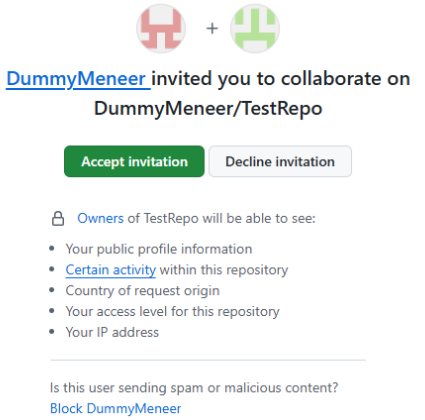


Nadat je de uitnodiging hebt verstuurd via de “add to repository” knop. Komt de persoon in het access vakje terecht. De status van de persoon zal staan op pending invite totdat de persoon de invite geaccepteerd heeft.

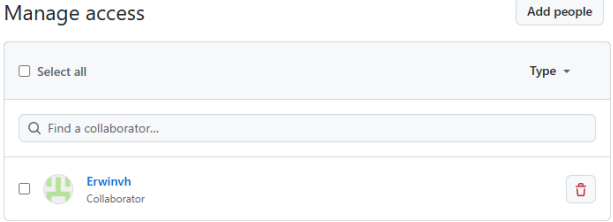
Deze invite zit in de mail van de persoon die je probeert toe te voegen en is 7 dagen geldig. Hierna komt de invite te vervallen en moet het opnieuw aangemaakt worden.

Door de invitation te accepteren kan de andere persoon bij de repo en kan deze meewerken aan het project.





In het overzicht zal de status van de persoon ook verwijderd worden zoals in de afbeelding hieronder. Mocht het toch zo zijn dat de verkeerde persoon is toegevoegd dan kan je als eigenaar van de Repo de persoon verwijderen door op het vuilnisbakje te klikken.



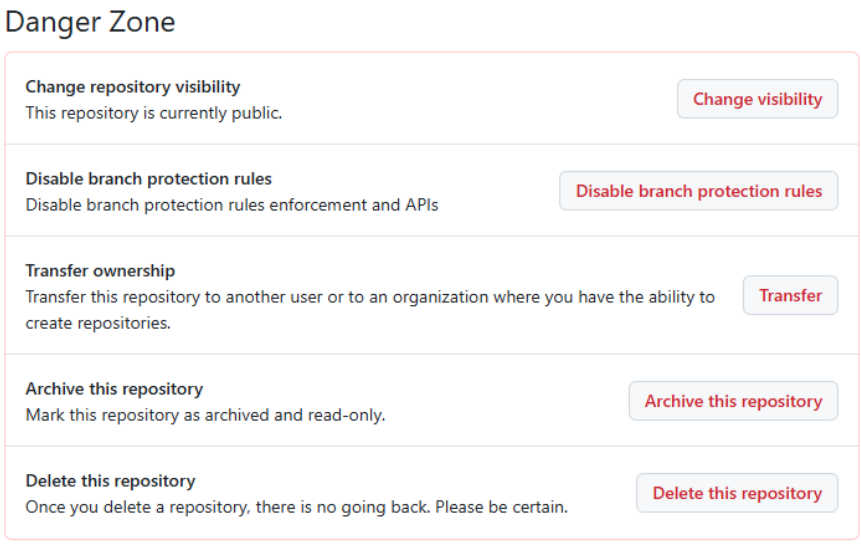
Voor deze **opdracht** maak je 1 repo waarin je een aantal klasgenoten zet, Dit hoeft niet je werkgroepje te zijn voor de Pygame. Maak deze repo aan en zorg ervoor dat iedereen de minstens 1 push heeft gedaan op deze repo. Deze repo gaan jullie weer gebruiken in hand-out 3 en 4.

Opdracht 5: Repo Danger zone (5 minuten)

Binnen de settings bij general is er een deel (Onderaan) dat heet de Danger Zone. Dit is omdat de acties die hier uitgevoerd kunnen worden grote consequenties kunnen hebben op je repo en project. Bij deze opdracht wordt er alleen kennis gemaakt met de functionaliteiten, er wordt niet gevraagd om deze te gebruiken.

De eerste acties is de zichtbaarheid aanpassen. Van privé naar publiek opent de repo op tot commentaar en zichtbaarheid van gegevens. Als je een file in de repo hebt staan met wachtwoorden of tokens kan dit geen veilige actie zijn. Van publiek naar private kan handig zijn maar sommige functionaliteiten zijn niet beschikbaar voor private repos voor de gratis accounts.

De tweede is het verwijderen van branch beschermingsregels. Er is over het algemeen vrij weinig reden om de basis branch beveiliging uit te schakelen dus deze wordt niet aangeraden om te wijzigen.



De derde is de transfer ownership. Dit laat je toe om de repo aan iemand anders te geven. Deze persoon wordt dan eigenaar. Het is verstandig om dit enkel te doen als het expliciet nodig, het overdragen van eigenaarschap is zeer gevoelig voor fouten en als het bij een onbekende komt kan deze alles verwijderen.

De vierde optie is om de repo te archiveren, dit betekent dat de Repo zal blijven bestaan maar dat wijzigingen eraan toepassen niet meer mogelijk is. Dit is vaak handig als een project afgerond is en niet meer aan gewerkt wordt. Deze actie is omkeerbaar.

De laatste is het meest gevaarlijk, dit is de delete repository optie. Zoals de naam aangeeft kan je hiermee de Repository verwijderen. Soms is dit handig, zeker als je een slechte of ongebruikte repo hebt die je niet meer op in je GitHub wil hebben om het opgeruimd te houden. Maar wees er 100% zeker van aangezien deze actie niet omkeerbaar is.

Opdracht 6: Naming conventies (15 minuten)

In week 1 is er al genoemd dat commit namen het beste consistent gehouden kunnen worden. Dit wordt hier uitgebreid zodat het er goed is ingesleten tegen het einde van het vak. Bij grote projecten komen veel commits bij kijken. Als er dan door de commits gekeken moet worden waar een fout of een stukje verwijderde code zit dan is het handig om een manier te hebben waaraan je kan zien wat er per commit gedaan is. Hier komen naming conventies bij kijken.

In deze opdracht gaan we kijken naar 3 naming conventies, echter zijn er veel meer dan dat online te vinden. Het belangrijkste bij naming conventies is consistentie aangezien meerdere door elkaar gebruiken ervoor kan zorgen dat de positieve aspecten ervan wegvallen.

3-staps benoeming

Iedere commit titel bestaat uit 3 stukken: Globale, Onderdeel en specifiek. Dit zorgt ervoor dat de persoon die er doorheen gaat precies weet wat er is gebeurd. De 3-staps benoeming is over het algemeen van globaal naar specifiek in volgorde. Bijvoorbeeld:

> Add, Server, EmailService
>
> [FIX] UI confirmbutton color
>
> Delete Main duplicate code

De schrijfwijze moet ook consistent zijn maar voorbeelden laten zien hoe de 3-staps benoeming op verschillende manieren toegepast kan worden.

De details van de commit worden dan vervolgens uitgeschreven in de beschrijving.

Zinsvorm

Zinsvorm is de manier waarbij er een zin gemaakt wordt binnen de titel die een korte beschrijving geeft van de commit. Als de commit te veel bevat om een duidelijke, korte en overzichtelijke zin te maken dan heb je te veel in je commit staan. De zin is 50 karakters max. een set voorbeelden zijn:

> Een set achtergrond afbeeldingen toegevoegd.
>
> Ik heb de convert functie geoptimaliseerd.
>
> De testcase is verwijderd omdat er een bug in zat.

De details worden dan beschreven in de omschrijving van de commit. Welke tastcase en welke bug, welke achtergrond afbeeldingen zijn toegevoegd en hoe is de functie geoptimaliseerd, al dit komt te staan in de omschrijving.

Tag benoeming

Tag benoeming werkt op een vergelijkbare manier als de tags van GitHub maar is bedoeld om mensen toe te laten om de Search functie van een browser te benutten. De titel van een commit is dan ook opgebouwd uit steekwoorden zoals:

> FIX SERVER BUG
>
> ADD UI PAGINA KLANTEN
>
> [DEL] [LINK] [TEST] [DB] [DATA]

Ook hier worden de details uitgewerkt in de beschrijving.

Voor de opdracht bekijk deze benoemingen en overleg rustig met anderen welke conventie jij zou gebruiken of dat je een eigen zou kiezen en waarom. In het geval van je eigen conventie, wat zijn de regels en waarom werkt dat beter? Schrijf je uiteindelijke gekozen conventie op, deze lever je met de rest van week 2 in.

Opdracht 7: Repo eindopdracht maken met groepje

Als deel van je oplevering van dit vak moeten jullie een repository aanmaken en bijhouden van jullie PyGame groepsopdracht. In deze opdracht gaan jullie in het groepje aan de slag om de basis neer te zetten. De volgende dingen moeten gedaan worden in de GitHub:

Maak een Repository aan voor het project. (Maar 1 persoon hoeft dit te doen!)

Maak een duidelijke en herkenbare naam en een duidelijke omschrijving van wat de Repo in gaat houden.

Voeg je Groepsgenoten toe aan de Repo.

Indien je de repo op private zet, voeg dan mij toe (DummyMeneer). Dit laat me toe om de repo te bekijken en beoordelen richting het einde. Voor een publieke repo is dat niet nodig, enkel een linkje of naam is genoeg.

Maak een documentje aan waarin staat wat jullie afspraken zijn rondom hoe jullie commits gaan noemen.

Week 2 inleveren

In week 2 lever je een zipfile in met het resultaat van opdracht 2. Verder wordt er gekeken naar of je de andere opdrachten (4 en 5) hebt gedaan door middel van je GitHub te bekijken. Als je wilt dat er naar een specifieke repo gekeken wordt, zet een tekstbestand in de zipfile met daarin de naam/namen van de repo(s) die bekeken moeten worden.

Maak ook een tekstbestand aan met daarin de naming conventie die je wilt gaan gebruiken voor je eigen projecten. Dit mag een eigen naming conventie zijn of een die al bestaat maar geef aan welke je wilt gaan gebruiken.

Voor degene die al veel met GitHub hebben gewerkt lever alsnog een zipfile in met een clone naar keuze en geef aan wat voor soort naming conventie je gebruikt of wilt aanhouden.

Als laatste voeg een tekstbestand toe met de namen van je groepsleden, de naam van de repo en welke naming conventie jullie als groep gaan gebruiken.

Volgende week

Volgende week worden de volgende onderwerpen behandeld:

- GitHub projecten
- Insights
- Issues