hogeschool vives

# Web Development 1 : GitHub tutorial

studiegebied **HWB**

bachelor in het **toegepaste Informatica**

campus **Kortrijk**

academiejaar **2024-2025**

# Inhoudsopgave

# 1 Inleiding

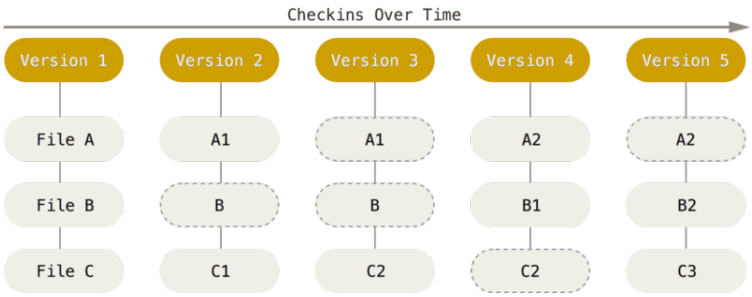
Git is een krachtige *versioning tool* die elke ontwikkelaar en IT specialist moet kennen. Je kan er lokaal, maar ook online, code mee opslaan. Elke versie die je *commit* gaat voor altijd bijgehouden worden.

Laten we eerst eens kijken naar git. Dit is een versiebeheersoftware voor ontwikkelaars:


Versiebeheer verwijst naar het proces van het opslaan van verschillende bestanden of “versies” in de verschillende stadia van een project. Dit stelt ontwikkelaars in staat om bij te houden wat er is gedaan en eventueel terug te keren naar een vorige fase wanneer ze een aantal van de aangebrachte wijzigingen ongedaan willen maken.

Dit kan om verschillende redenen van pas komen. Het maakt het bijvoorbeeld eenvoudiger om fouten op te lossen en fouten te corrigeren die tijdens de ontwikkeling zijn opgetreden. Je kunt ook wijzigingen in elke versie noteren, zodat teamleden op de hoogte blijven van wat er is voltooid en wat nog moet worden gedaan.

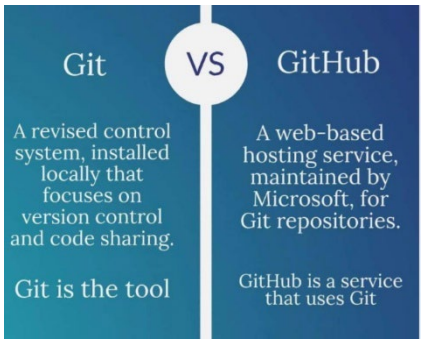
In tegenstelling tot de meeste andere versiebeheersystemen (ook wel Version Control Systems of VCS’s genoemd), slaat git elke opgeslagen versie op als een snapshot, een “momentopname” in plaats van een lijst met wijzigingen in elk bestand.




Git stelt je ook in staat om veranderingen van en naar installaties op andere computers te pushen en te pullen. Dit maakt het een zogenaamd “gedistribueerd versiebeheersysteem” en stelt meerdere ontwikkelaars in staat aan hetzelfde project te werken.

# 2 GitHub

## 2.1 Wat is GitHub?




Git VS GitHub

(bekijk zeker de video) GitHub maakt het makkelijker om met Git samen te werken. Het is een online platform dat gratis *Git repositories* aanbiedt zodat je ook online je code kwijt kan. Het is ook een platform dat opslagplaatsen van codes kan bevatten op meerdere locaties, zodat meerdere ontwikkelaars aan één project kunnen werken en elkaars bewerkingen in real-time kunnen bekijken.

Bovendien bevat het ook projectorganisatie- en beheerfuncties. Je kunt taken toewijzen aan individuen of groepen, toestemmingen en rollen instellen voor contributors en gebruik maken van opmerkingen om iedereen bij de les te houden.

Een extra functionaliteit van hen laat toe om 1 repository als een *web server* te doen fungeren, wat voor ons uiteraard zeer nuttig is om de labo's online te publiceren.

## 2.2 Verschil tussen Git en GitHub

1. Git is lokale VCS software waarmee ontwikkelaars snapshots van hun projecten in de loop van de tijd kunnen opslaan.
2. GitHub is een webgebaseerd platform met git's versiecontrolefuncties, zodat ze in samenwerkingsverband kunnen worden gebruikt. Het bevat ook functies voor project- en teambeheer, evenals mogelijkheden om te netwerken en sociale codering.

Dit document legt kort uit hoe je code op een Git repository kwijt kan en hoe je start met GitHub Pages. Voor gedetailleerde informatie over Git kan je terecht op de resources pagina van GitHub[1].

[1] https://try.github.io/

# 3 Git

## 3.1 Benodigde tools

Git kan je via de commandline aanspreken of via een GUI (grafische interface) tool. Het werkt op Linux, Mac en Windows en het kiezen van de tool die je gebruikt is een kwestie van smaak. In deze introductie gebruiken we GitHub Desktop[2], maar er zijn nog een aantal andere tools die de moeite zijn:

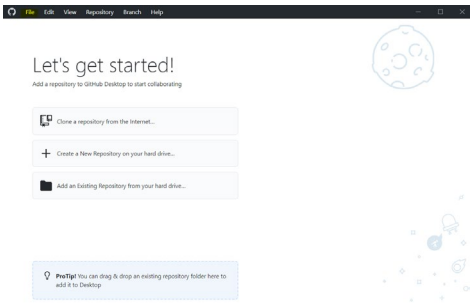
- GitKraken[3]
- TortoiseGit[4]
- Magit[5]

Het is een eindeloze lijst, dus probeer er een paar uit en kies degene die je het beste ligt. Deze introductie gebruikt GitHub Desktop zodat er gemakkelijker kan verwezen worden naar specifieke functionaliteit.

## 3.2 GitHub Desktop

Voer de volgende stappen uit:

1) Maak een GitHub account op https://github.com/.
2) Download en installeer de tool voor je systeem hier: https://desktop.github.com/.
3) Start GitHub Desktop



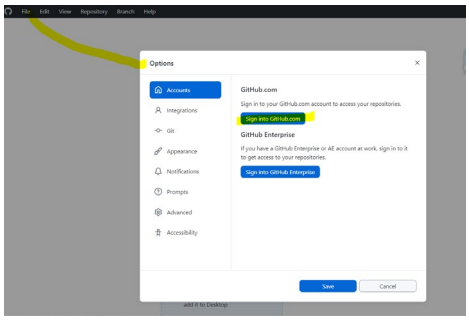

   a. GitHub Desktop integreert met GitHub, dus bij het opstarten van GitHub Desktop moet je inloggen met je GitHub account en Desktop *authorizen (File -> Options)*.

[2] https://desktop.github.com/

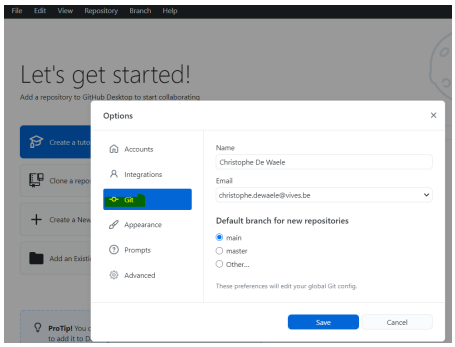
[3] https://www.gitkraken.com/

[4] https://tortoisegit.org/

[5] https://magit.vc/




b. Onder (File -> Options) *Configure Git* geef je een naam en email op. Deze gegevens zullen bij *elke nieuwe versie* van je codebase (dit noemen we een *commit*) gepubliceerd worden: zo weet je achteraf wie welke aanpassingen gemaakt heeft.




4) GitHub Desktop is nu klaar voor gebruik: je krijgt het scherm *'Let's get started'* te zien.

## 3.3 Git in a nutshell

Laten we de belangrijkste concepten van Git op een rijtje zetten door een test repository aan te maken en een aantal basis operaties te illustreren.

### 3.3.1 GitHub termen

Hieronder staan belangrijke GitHub termen:

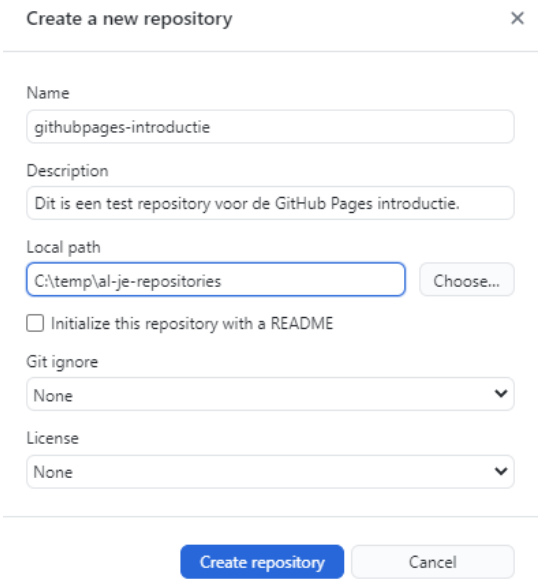
| GITHUB TERM | BETEKENIS |
|---|---|
| **REPOSITORY** | Hierin staan alle bestanden van je project en de historie van wijzigingen die je hebt gedaan. |
| **BRANCH** | Aparte plek binnen je repository, waar je bijvoorbeeld nieuwe code kan testen, zonder dat te hoeven doen op de "productversie". |
| **MAIN** | Dit is de hoofdbranch, oftewel de "productversie" van je project. Nieuwe code die is getest en goedgekeurd, wordt samengevoegd met de Main branch (vanaf versie 2.9 Main branch). |
| **FORK** | Een fork is een kopie van een repository. Hierdoor kan je werken aan een project van iemand anders, zonder het origineel aan te passen. |
| **COMMIT** | Git commando dat veranderingen toevoegt aan je lokale repository. |
| **PUSH** | Git commando om aanpassingen naar je (remote) repository te sturen, die staat op GitHub. |

| | |
|---|---|
| **PULL** | Git commando om aanpassingen van je (remote) repository naar je lokale bestanden te sturen. |
| **MERGE** | Git commando om aanpassingen van een branch samen te voegen met een andere branch. Bijvoorbeeld aanpassingen die getest en goedgekeurd zijn in de “Develop” branch samenvoegen met de “Master” branch. |
| **CHECKOUT** | Deze Git commando wordt vaak gebruikt om te switchen tussen branches. Je checkt als het ware uit bij een branch en gaat aan de slag in een andere branch. |

Er zijn natuurlijk nog meer GitHub termen.

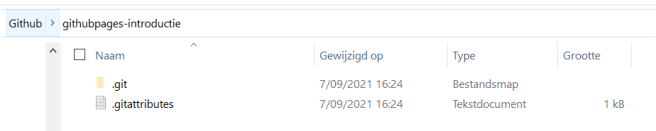
### 3.3.2 Repository aanmaken

Open GitHub Desktop en maak een nieuwe repository aan door *File > New Repository* te kiezen in de menu (zie Figuur 1). Je repository krijgt een naam, een beschrijving en je moet het *Local path* opgeven waarin je repository (dit wordt een aparte folder binnen dat *Local* path) geplaatst wordt. Klik ‘Create repository’ en check dat je repository effectief gemaakt werd.

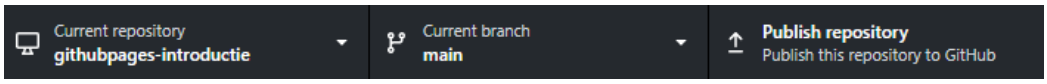



Figuur 1 Screenshot 'Create a new repository'.

Je repository is nu een lege folder met daarin 2 *hidden files*: een `.Git` folder en een `.gitattributes` file. Deze files heeft Git nodig om correct te werken, dus pas er voorlopig niets in aan of verwijder ze niet.



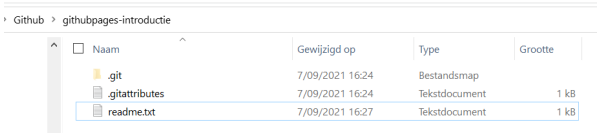
In GitHub Desktop zie je nu het volgende:




De repository *githubpages-introductie* is actief, op branch *main*. Deze repository staat *lokaal op je computer*, en heeft nog niks te maken met GitHub. Andere Git tools kunnen dus ook perfect met deze repository aan de slag.

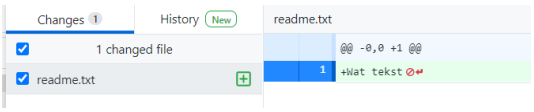
### 3.3.3 Bestanden toevoegen

De repository is momenteel leeg. Voeg een bestandje toe in je folder. Noem het '`README.txt`' en zet er wat tekst in. Je komt nu in de staat van Figuur 2 terecht.




Figuur 2 Eerste bestand toegevoegd.

Deze aanpassing zal zichtbaar worden in GitHub Desktop. Je ziet 1 *pending change* staan (zie Figuur 3). Er werd gedetecteerd dat `README.txt` werd toegevoegd en je kan aan de rechterkant ook zien welke tekstlijnen er werden aangepast (in ons geval dus enkel toevoegingen: de tekstlijnen van je bestand).



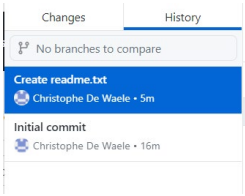

Figuur 3 Repository status na toevoeging file.

Om deze aanpassingen *vast te leggen* in je repository moeten de volgende stappen gemaakt worden:

1) *Add* de bestanden. Dit doe je door het bestand *aan te vinken* in GitHub Desktop, wat je bestanden toevoegt aan je *stage*. De *stage* is de verzameling van aanpassingen die je wil vastleggen.

2) *Commit* je stage. Dit doe je in GitHub Desktop door op *Commit to main* te klikken. Een *commit* in Git geef je typisch een *commit message* mee. Dit bevat een 1) Summary 2) Description. Het is voor *echte* repositories belangrijk om dit steeds te doen. Zo krijg je een mooie tijdlijn van aanpassingen en kan je gemakkelijk zien wie er wat op welk moment heeft aangepast.

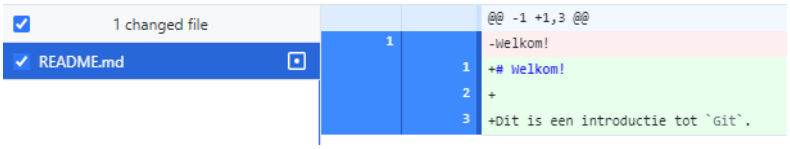
Je nieuwe file staat nu vastgelegd in je repository. Klik op *History* in GitHub Desktop om je commits te bekijken (zie Figuur 4). Hier kan je elke commit zien die er gebeurd is binnen je repository, wie de commit gemaakt heeft en wat de aanpassingen waren.




Figuur 4 Status na commit van eerste file.

### 3.3.4 Bestanden aanpassen

Het aanpassen van bestanden gebeurt analoog. Voeg een lijn toe aan je `README.md` file en **save** je file. In GitHub Desktop zie je de aanpassing ook verschijnen (zie Figuur 5). Deze aanpassing moet je weer *Adden* (door het aangevinkt te laten) en *Commiten* (door op *Commit to main*) te klikken. Geef er een goede *Summary* tekst (een one-liner die beschrijft wat de aanpassing is en *Description* (alles wat je niet in je summary kwijt kon, optioneel) voor op.

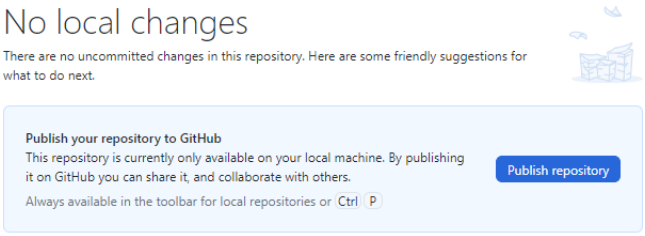



Figuur 5 Repository na tweede aanpassing.

Check in je history of je commit er is bijgekomen!

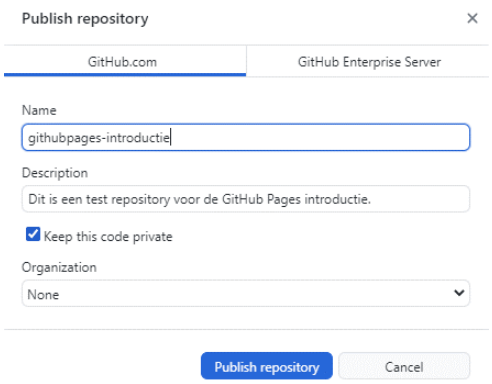
### 3.3.5 Van lokaal naar online

We hebben op GitHub een account, wat ons toelaat om op hun systeem *repositories* aan te maken en te hosten. Je kan dit gemakkelijk doen via GitHub Desktop door in je hoofdvenster (zie Figuur 6) te klikken op *Publish repository.* Dit zal een repository aanmaken op GitHub en je lokale repository eraan koppelen.




Figuur 6 Hoofdvenster, no local changes.

Na het klikken op *Publish repository* krijg je een venster met wat opties (zie Figuur 7). Deze opties zijn gelijkaardig aan die wanneer je je lokale repository aanmaakte. Je kiest een naam en een beschrijving. In dit geval (en meestal) mogen die perfect hetzelfde zijn als je lokale repository, om alles egaal te houden.



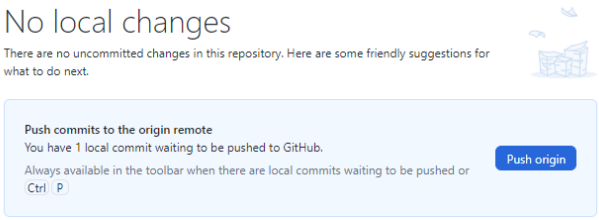

Figuur 7 Publish repository options.

Klik op *Publish repository* en wacht tot de operatie gelukt is. Eens gelukt kan je op https://github.com kijken of je repository correct werd gepubliceerd. Log in, en kijk onder je repositories of de repository *githubpages-introductie* (als je dezelfde naam gebruikt hebt) zichtbaar is, inclusief commits en files.

### 3.3.6 Publiceren van aanpassingen

Nu je repository ook online te vinden is kunnen we het laatste *update* mechanisme uitleggen. Pas je lokale `README.txt` nog een keer aan door er een lijn aan toe te voegen (of verwijder / update er één). *Commit* deze change via GitHub Desktop. Deze aanpassing is nu enkel vastgelegd in je lokale repository (verifieer dit door in GitHub Desktop in je history te kijken, en op github.com te kijken naar je repository).

Om de lokale aanpassingen ook door te voeren in je *remote* repository (in dit geval je GitHub.com repository) moet je de commits *pushen*. Dit doe je in GitHub Desktop gemakkelijk door op *Push origin* te klikken in je hoofdvenster (zie Figuur 8).



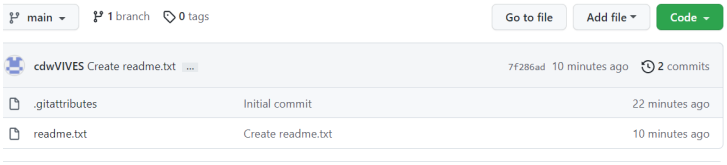

Figuur 8 GitHub Desktop, pending commit.

Ter info: *origin* betekent hier je *lokale repository*. Een *remote* repository is een niet-lokale repository. Nadat je je commit hebt gepushed kan je dit verifieren door op GitHub.com te checken of die zichtbaar wordt.

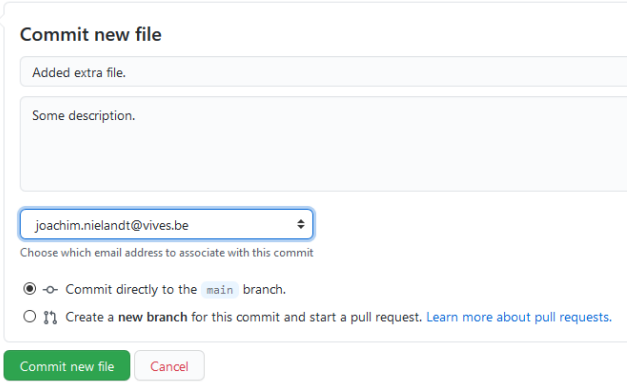
### 3.3.7 Updaten van lokale repository

Ten slotte kan het zijn dat iemand anders je *remote repository* heeft geüpdatet met een *commit*. Dit betekent dus dat je *remote* iets verder staat in de versie geschiedenis dan je *lokale* repository. Op dit moment kan en mag je geen commits pushen. Als je dit probeert zal Git klagen en een foutmelding geven, met de boodschap dat je eerst een *pull* moet doen.

Dit kunnen we snel testen door op github.com via de web interface een bestand toe te voegen. Klik op de knop *Add file > Create new file* (zie Figuur 9) onder je repository op GitHub.com. Geef de file een naam (`extra.txt`) en typ wat inhoud. Onderaan de pagina kan je dan je nieuwe file ook *committen*. Geef een *summary* en optioneel een *description* op (zie Figuur 10) en klik vervolgens op *Commit new file*. Check in je online repository of je commit effectief werd doorgevoerd: `extra.md` moet zichtbaar worden in het file overzicht.

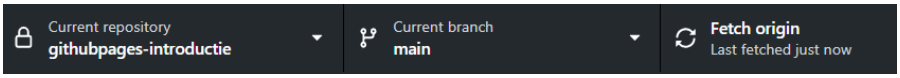



Figuur 9 Remote repository status.

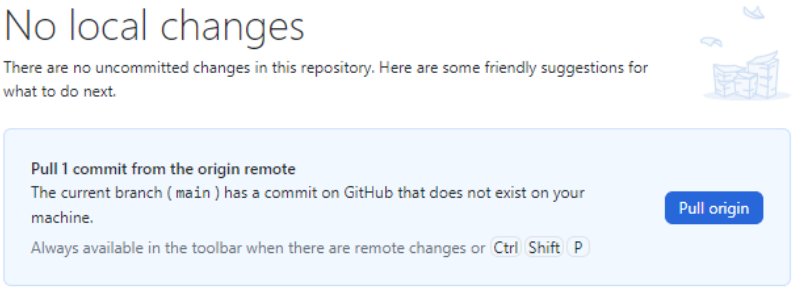



Figuur 10 Online commit procedure.

Het nieuwe bestand ga je nog niet kunnen zien in je *local* repository. Om deze aanpassing (en dus de laatste commit) ook lokaal te krijgen moet je een *Git pull* doen. In GitHub Desktop kan je dat doen door in het menu *Repository > Pull* te klikken. Je kan ook op het derde knopje bovenaan klikken:




Dit *fetcht* de laatste updates van je remote repositories. Daarna kan je in het hoofdvenster (zie Figuur 11) op de knop *Pull origin* klikken.




Figuur 11 Hoofdvenster voor pull.

Nadat je de laatste commit van je *remote repository* hebt *gepulled* zie je deze commit ook verschijnen in je *local repository*. Verifieer dit in je lokale folder: de file die je hebt toegevoegd op GitHub.com moet je ook terugvinden.

## 3.4 Zelfstudie

Dit document dient voornamelijk om je zo snel mogelijk in gang te zetten zodat je er gebruik van kan maken in Web Development. Git kan echter meer voor je betekenen dan wat er hier uitgelegd staat, eens je het onder de knie hebt is het een onmisbare tool.

GitHub heeft zelf een tutorial website[6] waarop Cheat Sheets (handig overzicht van commando's), tutorials en interactieve voorbeelden staan. We raden sterk aan om deze eens zelf door te nemen.

De belangrijkste concepten die je nog moet bekijken:

- *Git status* kan je oproepen om de huidige staat van je repository te bekijken. Hier zie je, e.g., wat er op je *stage* staat, welke files er *untracked* zijn en welke aanpassingen er gebeurd zijn.
- *Mergen* moet je doen wanneer er *upstream* een aanpassing is gebeurd die je *lokaal* nog niet hebt. In ons geval niet belangrijk, maar eens je op verschillende computers ontwikkelt is het cruciaal dit te begrijpen.
- *Branches* gebruik je om een zijspoor in te slaan in je code of om je development door meerdere ontwikkelaars beter te managen.
- *Logs* kan je bekijken om te zien wanneer en in welke commit er een aanpassing gebeurd is. Git laat toe om dan selectief dingen terug te brengen naar het *heden*.
- *Pull requests* kan je aanmaken om aanpassingen te suggereren in andere mensen hun repository. Zij kunnen dan besluiten je aanpassing te mergen in hun code. Het is de plek om een discussie te starten over, e.g., een nieuwe feature.
- Een *fork* maak je wanneer je je code wil baseren op een bestaande repository. Deze *fork* je dan zodat je hun geschiedenis overneemt, maar je bewandelt op dat moment je eigen weg.
- *Stash* kan je gebruiken om een aantal aanpassingen even aan de kant te zetten wanneer je een commit doet. Deze aanpassingen zullen dan niet opgenomen worden in de commit, en kan je weer tevoorschijn toveren nadat je commit is gemaakt.

[6] https://try.github.io/

# 4 GitHub Pages

GitHub laat toe om 1 specifiek genoemde repository te gebruiken als een web server[7]. GitHub gaat zelf zorgen voor het hosten van de files onder die repository, wat voor ons een aangename manier van werken is. Deze sectie dient om je snel op gang te zetten voor de labos van Web Development.

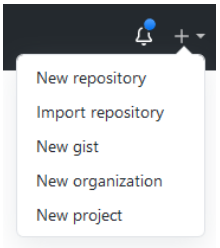
Om alles te volgen heb je je GitHub **username** nodig. Dit kan je zien door rechtsboven te klikken op je user icoontje, waarna je "Signed in as **username**" te zien krijgt. Of, in de url van je test repository in Sectie 3:

`https://github.com/username/githubpages-introductie`

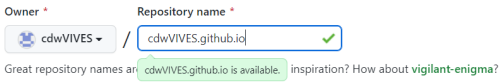
## 4.1 Aanmaken repository

Volg de volgende stappen exact:

1) Maak een nieuwe repository aan op GitHub.com (rechtsboven de pagina, zie Figuur 12).
2) Noem deze ***exact*** op de volgende manier: **username.github.io** (zie Figuur 13).
3) Geef het een optionele beschrijving en **maak de repository *public*** (belangrijk, anders werkt GitHub Pages niet).
4) Klik *Create repository*.




Figuur 12 Create repository op GitHub.com.




Figuur 13 Naamgeving GitHub Pages repository.

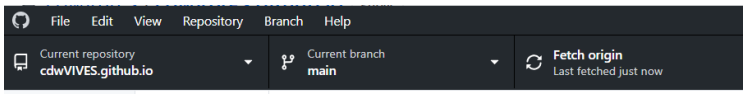
## 4.2 Repository clonen

Nu hebben we een situatie die we nog niet zijn tegengekomen: er is een *remote repository* (op GitHub.com) die we *lokaal* nog niet hebben staan (op onze computer). Hier dient de operatie *Git clone* voor. Open GitHub Desktop en kies de optie *File > Clone Repository* in het menu. GitHub Desktop maakt het hier gemakkelijk door al je GitHub.com repositories reeds op te lijsten. Kies je

[7] https://pages.github.com/

`username/username.github.io` repository en klik *Clone* nadat je in *Local path* je folder hebt gekozen waarin je al je Git repositories bijhoudt.

Check dat je de folder lokaal effectief ziet verschijnen op je computer. In GitHub Desktop kan je nu ook switchen tussen de verschillende lokale repositories. Klik hiervoor op de eerste van de drie bovenste knoppen '*Current repository*', zie volgende figuur:




## 4.3 Pages testen

We maken nu twee dingen aan: een `index.html` file een een *fake labo* onder de folder `fake.labo`, met daarin een `.html` file labo.html. Zorg dat je de volgende situatie nabootst:

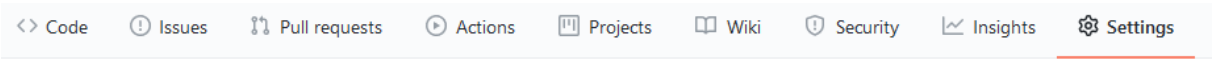
`C:\temp\GitHubRepositories\username.github.io\index.html (C:\temp\GitHubRepositories zal al een ander pad zijn bij jullie)`
`C:\temp\GitHubRepositories\username.github.io\fake.labo\labo.html`

Voeg wat tekst of HTML-elementen toe aan deze `.html` files. *Commit* vervolgens deze lokale aanpassingen (het aanmaken van de bestanden en toevoegen van inhoud) via GitHub Desktop. Nadat je de *lokale commit* gedaan hebt, *push* je ook deze commit naar de GitHub.com repository (je *remote repository*). Verifieer nu online op GitHub.com dat je nieuwe bestanden zijn gepushed.

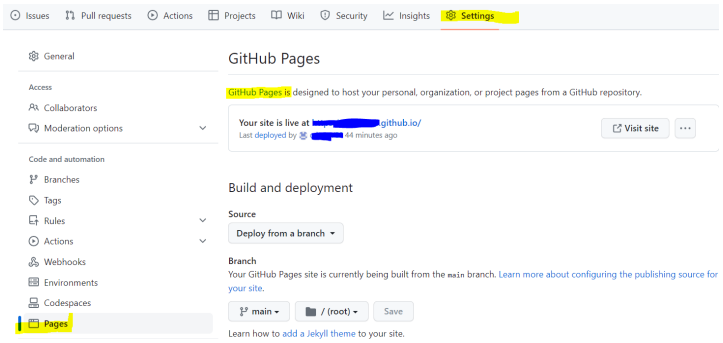
### 4.3.1 Pages status check

Je kan op verschillende plaatsen checken of GitHub je repository correct behandelt als een *GitHub Page*.

Op je repository main page (https://github.com/username/username.github.io) klik je op Settings




Klik vervolgens op Pages (sidebar) en verifieer dat je de GitHub Pages ziet (zie Figuur 14). GitHub vertrouwt hier op een correcte en exacte naamgeving van je repository. Er kan maar 1 repository per account een Pages behandeling krijgen, dus zorg ervoor dat je dit *exact* volgt.




Figuur 14 GitHub Pages settings.

Het duurt soms even vooraleer je repository correct wordt *gedeployed* op de GitHub web server.

### 4.3.2 GitHub Pages webserver

Als alles goed ging, heeft GitHub nu je repository gepubliceerd als een webpagina, op hun webserver. Dit kan je testen door te surfen naar https://**username**.github.io/.

Hier wordt *by default* de index.html file geopend. De browser zou dus correct de inhoud ervan moeten tonen. Bij wijze van test kunnen we ook ons *fake labo* testen. Surf hiervoor naar https://**username**.github.io/fake.labo/labo.html en verifieer dat je daar ook de inhoud van te zien krijgt.

## 4.4 Labo's publiceren

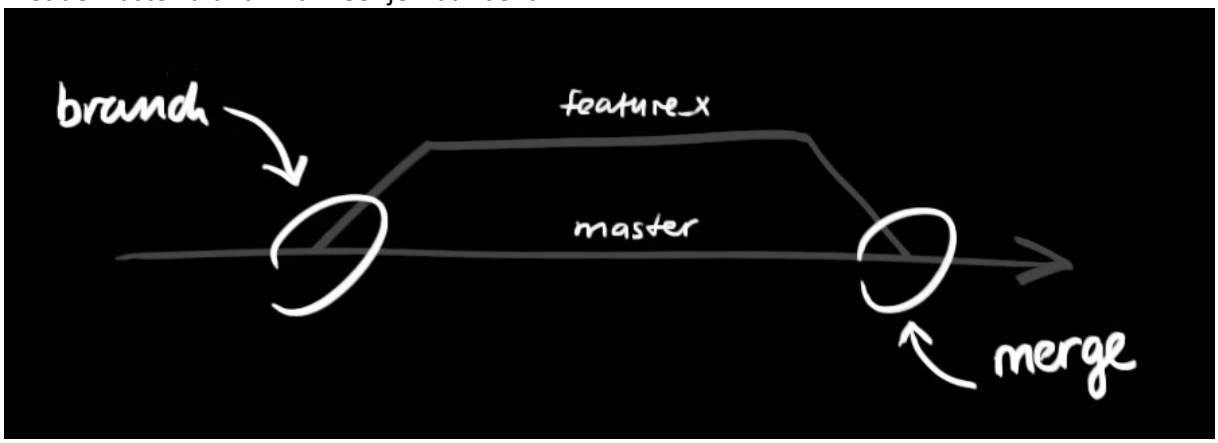
In plaats van *fake.labo* kan je natuurlijk nu je *echte* labo oefeningen hier publiceren. Plaats de inhoud van de folder met je bestanden van *Labo HTML deel 1* bijvoorbeeld in een subfolder `labo1` onder je repository `username.github.io` en *commit* en *push* de aanpassing naar GitHub.com. Je kan nu *online* je labo bekijken via https://**username**.github.io/labo1/X.html. Pas X uiteraard aan naar de juiste naamgeving. Of laat `X.html` weg als je een `index.html` gebruikt bij wijze van shortcut.

**Opgepast:** let op met spaties en andere vreemde tekens in je folder- en bestandsnamen. Beperk je idealiter tot kleine letters en punten. URL's worden op een speciale manier behandeld door browsers, en door deze beperking zorg je ervoor dat je niet op onverwachte problemen stoot.

# 5 Branching

Bijna elk VCS ondersteunt een bepaalde vorm van branchen. Branchen komt erop neer dat je een tak afsplitst van de hoofd-ontwikkellijn en daar verder mee gaat werken zonder aan de hoofdlijn te komen. Branches worden gebruikt om verschillende features te ontwikkelen in geïsoleerde omgevingen, los van elkaar.

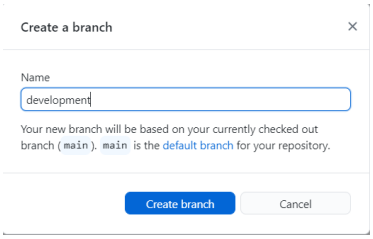
Maak nieuwe branches aan wanneer je nieuwe toevoegingen ontwikkelt en voeg ze samen (*merge*) met de *master* branch wanneer je klaar bent.



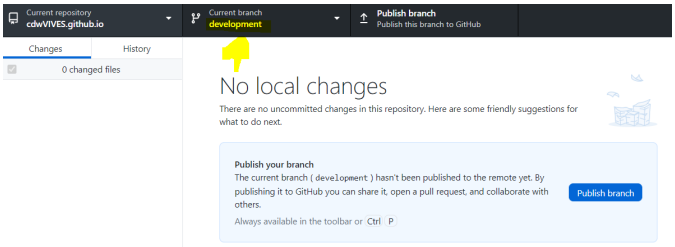

## 5.1 Aanmaken van een nieuw branch

Laten we nu een branch aanmaken, enkele wijzigingen aanbrengen en kijken hoe de wijzigingen specifiek zijn voor deze branch.

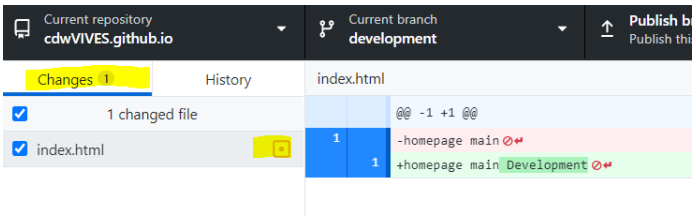
1. Ga in de GitHub Desktop-client naar Branch > New Branch en maak een nieuwe branch. Noem het een "development" -branch en klik op Create Branch.



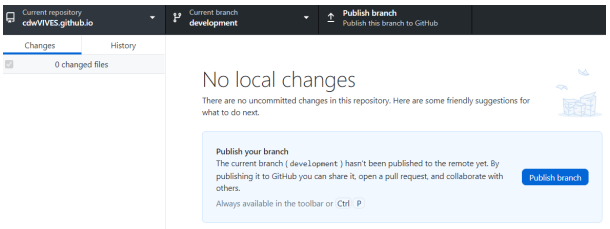

Wanneer de branch werd aangemaakt, zie je dat het drop-down menu aangeeft dat je in die branch werkt. Het maken van een branch kopieert de bestaande inhoud (van de branch waarin je je momenteel bevindt, master) naar de nieuwe branch (development).



2. Voeg aan het bestand index.html een lijn toe (via kladblok en save)
3. Keer terug naar GitHub Desktop en merk op dat je op het tabblad Changes nieuwe gewijzigde bestanden hebt.




4. Commit de wijzigingen, klik op Commit to development.
5. Klik op Publish branch om de local branch ook beschikbaar te maken via GitHub (Vergeet niet dat er meestal twee versies van een branch zijn — de local version en de remote version.)

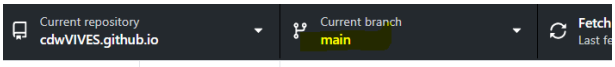



6. Schakel terug naar je master branch – main branch (m.b.v. de Branch drop-down). Kijk dan naar je bestand (index.html). Merk op dat de bestandswijzigingen niet werden doorgevoerd in de master branch.

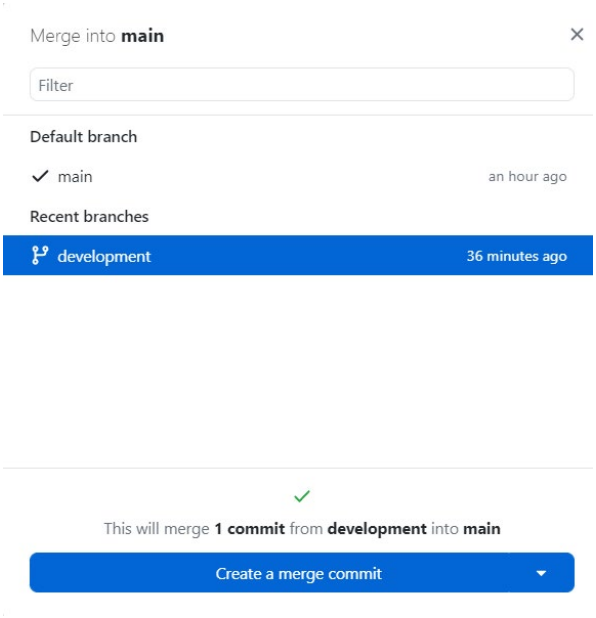
## 5.2 Branches samenvoegen

Laten we nu de development-branch samenvoegen met de master-branch.

1. Schakel in de GitHub Desktop over naar de branch waarin je de development-branch wenst toe te voegen. Selecteer in de branch selector main-branch




2. Ga naar Branch > Merge into Current Branch
3. Selecteer in het Merge Branch window de development-branch en klik vervolgens op “Create a merge commit”




4. Als het bestand index.html opent (kladblok), zou je de wijzigingen moeten zien.
5. Klik vervolgens op “Push orgin” om de wijzigingen naar de GitHub te pushen.