Software development
Crebonr.
25604

GitHub

VERSIEBEHEER

j.geyteman@rocva.nl

ROC van Amsterdam

Waarom versiebeheer?

Kwalificatie dossier

Het vak versiebeheer is tot stand gekomen vanuit het Kwalificatie dossier. In het kwalificatie dossier, staat namelijk dat een beginnende beroepsbeoefenaar versiebeheer kan toepassen. Dit wordt specifiek genoemd bij het werkproces B1-K1-W1 “Plant werkzaamheden en bewaakt de voortgang”. Dit is natuurlijk de formaliteit, waarom jij het vak versiebeheer krijgt.

Versiebeheer

Naast dat het Kwalificatie dossier vindt, dat jij als student actief versiebeheer moet kunnen toepassen. Is dit in de programmeer wereld de normaalste zaak van de wereld. Je schrijft namelijk als een programmeur extreem veel code in je leven. Dit kan alleen, je werkt dan met niemand samen. De kans op fouten is dan ook niet supergroot, maar niet 0%. Het is daarom belangrijk dat je verschillende versies bij houdt van de applicatie waarin je werkt. Mocht je een fout maken en de versie werkt helemaal niet meer, moet je terug kunnen naar een vorige versie.

Wat leer ik bij dit vak?

Bij dit vak ga je verschillende technieken & methodes leren. Om dit wat specifieker te maken, je leert:

- Actief versie beheer toe te passen.
- Samen werken doormiddel van GitHub.
- Wat repositories zijn.
- Wat branches zijn & hoe je deze merged.
- De workflow binnen GitHub.
- De applicatie GitHub te gebruiken.
- Verschillende ontwikkeltools kennen & kunt ze uiteindelijk gebruiken.

Deze 7 onderwerpen worden door de lessen heen actief behandeld, mocht je veel van deze termen niet kennen. Geen enkel probleem, je zult langzaam merken dat deze termen makkelijker zullen worden.

Toetsing & afsluiting van het vak

Het vak Versiebeheer sluit je af met een toets. Voor deze toets krijg je een cijfer & deze komt uiteindelijk in eduarte. Om het vak met een voldoende af te sluiten heb je een 5,5 nodig. Het is natuurlijk altijd mogelijk, dat je ziek bent tijdens het toets moment. Je hebt voor dit vak 1 herkansing, het kan natuurlijk ook dat je een onvoldoende haalt. Dat is ook geen probleem, je hebt namelijk recht op 1 herkansing. Je mag ook herkansen als je een toets voldoende hebt afgesloten. In dit geval geldt de regel: “het hoogste cijfer telt”.

Kerntaken die gedeeltelijk, of in zijn geheel voorkomen in dit vak:

- B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang
- B1-K1-W2: Ontwerpt software
- B1-K1-W4: Test software
- B1-K2-W2: Presenteert het opgeleverde werk
- B1-K2-W3: Reflecteert op het werk

Er wordt tijdens dit vak NIET getoetst op de werkprocessen, delen hiervan zullen worden behandeld en helpen je tijdens je examen in leerjaar 3.

Inhoudsopgave

1. Wat is versiebeheer?

Versiebeheer is het bijhouden van verschillende fases van een document: van aanmaak en aanvulling tot de definitieve versie. Ook registreer je er de wijzigingen in een document mee.

Je kunt versiebeheer op elk soort document toepassen zoals een offerte, contract of een rapport. Je gebruikt versiebeheer bij het opstellen van documenten. Je zet versiebeheer in bij het up to date houden documenten.

1.1. Waarom is versiebeheer belangrijk?

Met goed versiebeheer zorg je ervoor dat voor iedereen duidelijk is welk document – en daarmee welke informatie – het meest recent en actueel is. Zodat je daar als organisatie de beslissingen op kunt baseren. Je beschikt over deze voordelen:

- Samen tegelijkertijd aan één document werken.
- Tijdwinst: de juiste informatie is sneller te vinden.
- Altijd de meest recente versie beschikbaar.
- Werken in oude of verkeerde versies is verleden tijd.
- Informatie is – ook achteraf – juist op te vatten.
- Zie je precies wie welke wijzigingen wanneer heeft gedaan.
- Is er minder opslagruimte nodig omdat een document maar op één plek staat.
- Is de juiste informatie gemakkelijker te delen.

1.2. Onnodige risico's door chaotisch versiebeheer

Bedrijven waar versiebeheer niet goed is georganiseerd ervaren meer misverstanden en zijn gevoeliger voor fouten. Denk je bijvoorbeeld eens in wat er kan gebeuren als één van je medewerkers met de verkeerde informatie bij klanten of leveranciers komt? Of dat je je baseert op onvolledige feiten bij het maken van belangrijke beslissingen voor je organisatie? Om nog maar te zwijgen over het risico dat gevoelige documenten met de verkeerde mensen (binnen of buiten de organisatie) gedeeld worden. Naast deze risico's zijn medewerkers ook onnodig veel tijd en geld kwijt aan het zoeken naar de juiste informatie.

1.3. Hoe organiseer je versie- en documentbeheer in jouw organisatie?

De hoeveelheid informatie in en tussen bedrijven neemt alleen maar toe. Hierdoor groeit niet alleen de behoefte naar goed versiebeheer, maar ook de behoefte om documenten en informatie snel te zoeken en vinden. En goed te beveiligen, eenvoudig en veilig te delen en op tijd te archiveren. Versiebeheer kun je daarom zien als onderdeel van het document- en informatiebeheer in je organisatie. Maar hoe organiseer je dat? Realiseer je allereerst goed dat informatie- en documentbeheer een gezamenlijke verantwoordelijkheid is. Daarin wil je eenduidigheid en transparantie. Om je collega's hierbij te ondersteunen, kun je delen automatiseren. Voordat je serieus gaat oriënteren op een oplossing, is het belangrijk dat je weet wat je nodig hebt. Welke functionaliteit wens je? Stel jezelf (en je teamleden) daarom voor het onderdeel versiebeheer eerst eens de volgende vragen:

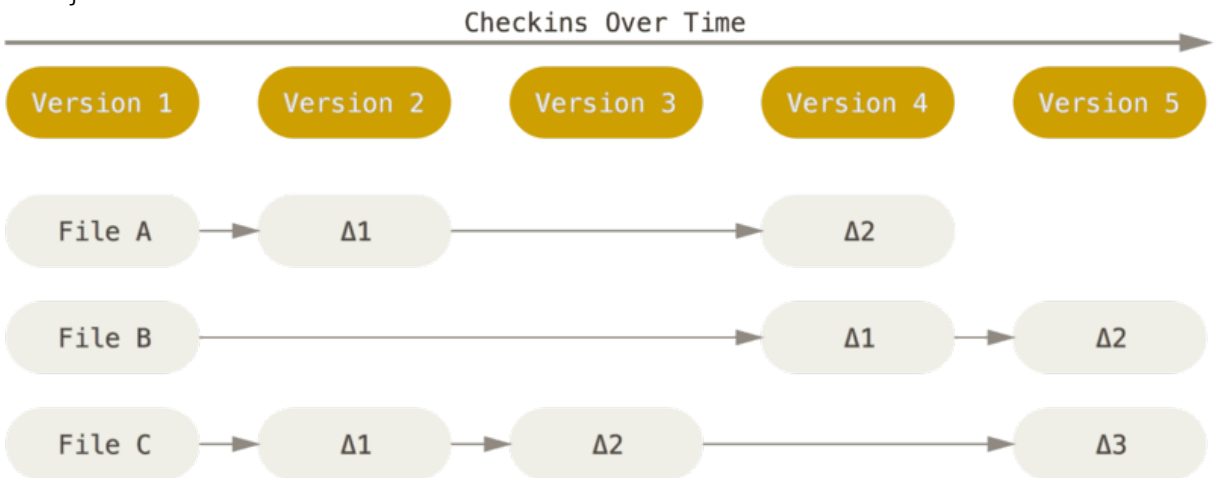
- Wil je op meerdere apparaten kunnen werken aan jouw documenten?
- Wil je met meerdere mensen tegelijkertijd aan één document werken?
- Wil je dat je documenten automatisch zijn uitgerust met je huisstijl?
- Moeten de verschillende versies centraal worden opgeslagen?
- Welke kenmerken moeten er worden geregistreerd?
- Op welke soorten documenten moet versiebeheer worden toegepast?

1.4. Versiebeheer in de ICT

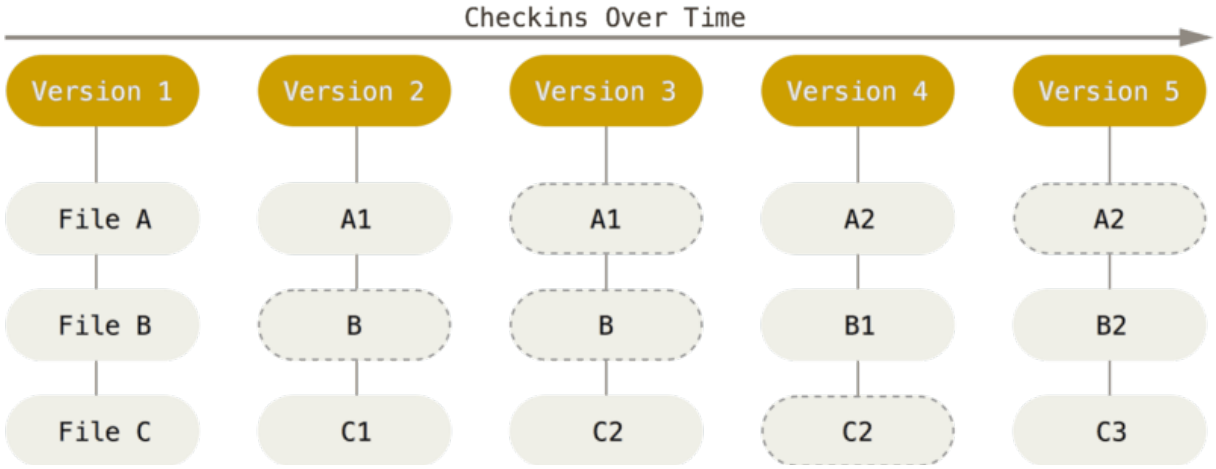
Als je een grafisch ontwerper bent of websites ontwerpt en elke versie van een afbeelding of opmaak wilt bewaren (wat je vrijwel zeker zult willen), is het zeer verstandig om een versiebeheersysteem (Version Control System in het Engels, afgekort tot VCS) te gebruiken. Het gebruik hiervan stelt je in staat eerdere versies van bestanden of het hele project terug te halen, wijzigingen tussen twee momenten in de tijd te bekijken, zien wie het laatst iets aangepast heeft wat een probleem zou kunnen veroorzaken, wie een probleem heeft veroorzaakt en wanneer. Een VCS gebruiken betekend meestal ook dat je de situatie gemakkelijk terug kan draaien als je een fout maakt, of bestanden kwijtraakt. Daarbij komt nog dat dit heel weinig extra werk met zich meebrengt.

1.5. Wat is Git?

Git is een opensource VCS-programma, dit programma is gratis te gebruiken en wordt wereldwijd door heel veel programmeurs gebruikt, maar er zijn natuurlijk veel meer open source VCS. Het grootste verschil tussen Git en elke andere VCS is de manier waarop Git denkt over haar gegevens. Conceptueel bewaren de meeste andere systemen informatie als een lijst van veranderingen per bestand. Deze systemen zien de informatie die ze bewaren als een groep bestanden en de veranderingen die aan de bestanden zijn aangebracht in de loop der tijd.



Het bewaren van data als veranderingen aan een basisversie van elk bestand. Git ziet en bewaart data heel anders. De kijk van Git op data kan worden uitgelegd als een reeks momentopnames (snapshots) van een miniatuur-bestandssysteem. Elke keer dat je **commit** (de status van je project in Git opslaat) wordt er als het ware foto van de toestand van al je bestanden op dat moment genomen en wordt er een verwijzing naar die foto opgeslagen. Uit oogpunt van efficiëntie slaat Git ongewijzigde bestanden niet elke keer opnieuw op, alleen een verwijzing naar het eerdere identieke bestand dat het eerder al opgeslagen had. Git beschouwt gegevens meer als een reeks van snapshots.



Dat is een belangrijk onderscheid tussen Git en bijna alle overige VCSen. Hierdoor moest Git bijna elk aspect van versiebeheer heroverwegen, terwijl de meeste andere systeem het hebben overgenomen van voorgaande generaties. Dit maakt Git meer een soort mini-bestandssysteem met een paar ongelofelijk krachtige gereedschappen, in plaats van niets meer een VCS.

1.6. Wat is GitHub?

GitHub bestaat uit twee woorden, namelijk Git + Hub. Laten we beginnen met Git. Git is een open source versiebeheersysteem. De website GitHub is gemaakt op basis van het versiebeheersysteem Git. Met GitHub heb je alle mogelijkheden van Git + Extra features.

Volgens veel programmeurs is Git momenteel het beste versiebeheersysteem, daarom is GitHub ook zo populair. In een versiebeersysteem kunnen programmeurs projecten beheren met code. Het is mogelijk om verschillende versies te beheren en eventueel terug te vallen op een oude versie, mocht er iets misgaan.

Maar waar staat “Hub” voor in GitHub? Met “Hub” wordt eigenlijk het centrale deel bedoeld, waar alle projecten samenkomen die beheerd worden met Git. De website GitHub is de centrale plek waar programmeurs hun projecten beheren met Git.

Video Git explained in 100 seconds

1.7. Welke andere mogelijkheden heeft GitHub?

Je weet nu dat het mogelijk is om in een team projecten (Waarin code staat) te beheren op GitHub. Het is natuurlijk ook mogelijk om je eigen projecten op GitHub te beheren. Zo heb je een goed overzicht van de aanpassingen en verschillende versies.

Daarnaast staat je project online. Als je een belangrijk project bijvoorbeeld alleen lokaal op je eigen computer opslaat en je computer houdt er mee op, dan ben je zwaar de pineut. Op GitHub kan je er gewoon altijd bij.

Maar welke mogelijkheden heeft GitHub nog meer? Hier staan een aantal belangrijke mogelijkheden op een rijtje:

- Een project maken gebaseerd op een project dat al bestaat
- Discussie starten over een project
- Code reviewen
- Aparte branches maken, waarin je bijvoorbeeld code aanpassingen doet om te testen, die niet gelijk in de “live versie” komen
- Code van branch samenvoegen met andere branch
- Tags meegeven aan verschillende versies, zoals V1.0 en V2.0
- Kwetsbaarheden in de code makkelijker ontdekken, GitHub stuurt ook een mail als het kwetsbaarheden ontdekt
- Een website hosten

1.8. Waarom is GitHub handig?

GitHub zorgt ervoor dat een individueel of een project in teamverband erg overzichtelijk is. Je hebt een goed overzicht van de laatste aanpassingen en welke persoon dat heeft gedaan.

Je kan eenvoudig meerdere versies maken en eenvoudig terugvallen op een vorige versie, mocht het misgaan. Door de verschillende branches loop je geen enkel risico bij het aanpassen en testen van nieuwe code.

Voor programmeurs maakt GitHub het werken een stuk makkelijker en leuker. Stel dat je als programmeur niet aan versiebeheer zou doen of dat zou doen met een slecht versiebeheer systeem, dan ga je vroeg of laat in de problemen komen.

GitHub is een centrale plek waar heel veel programmeurs en teams hun projecten beheren. Veel projecten zijn open source en kunnen door iedereen worden ingezien en aangepast.

Het is erg leerzaam om naar projecten van anderen te kijken. Als je wat meer ervaring hebt, kan je misschien wel verbeteringen voorstellen en meedoen aan gave projecten!

Met GitHub gaat er als programmeur een wereld voor je open. Het is goed om projecten te beheren en je kan er ook heel veel leren.

1.9. Betekenissen GitHub termen

Op GitHub zijn er een aantal termen die handig zijn om alvast te weten. Als je meer gebruik gaat maken van GitHub, dan leer je de betekenis van deze termen vanzelf kennen.

Hieronder staan belangrijke GitHub termen + de betekenis ervan:

GitHub term	Betekenis
Repository	Een repository kan je zien als de hoofdmap van je project. 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 codes kan testen, zonder dat te hoeven doen op de “live versie”
Master	Dit is de hoofdbranch, oftewel de “live versie” van je project. Nieuwe code die is getest en goedgekeurd, wordt samengevoegd met de Master branch.
Fork	Een fork is een kopie van een repository. Hieronder 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.

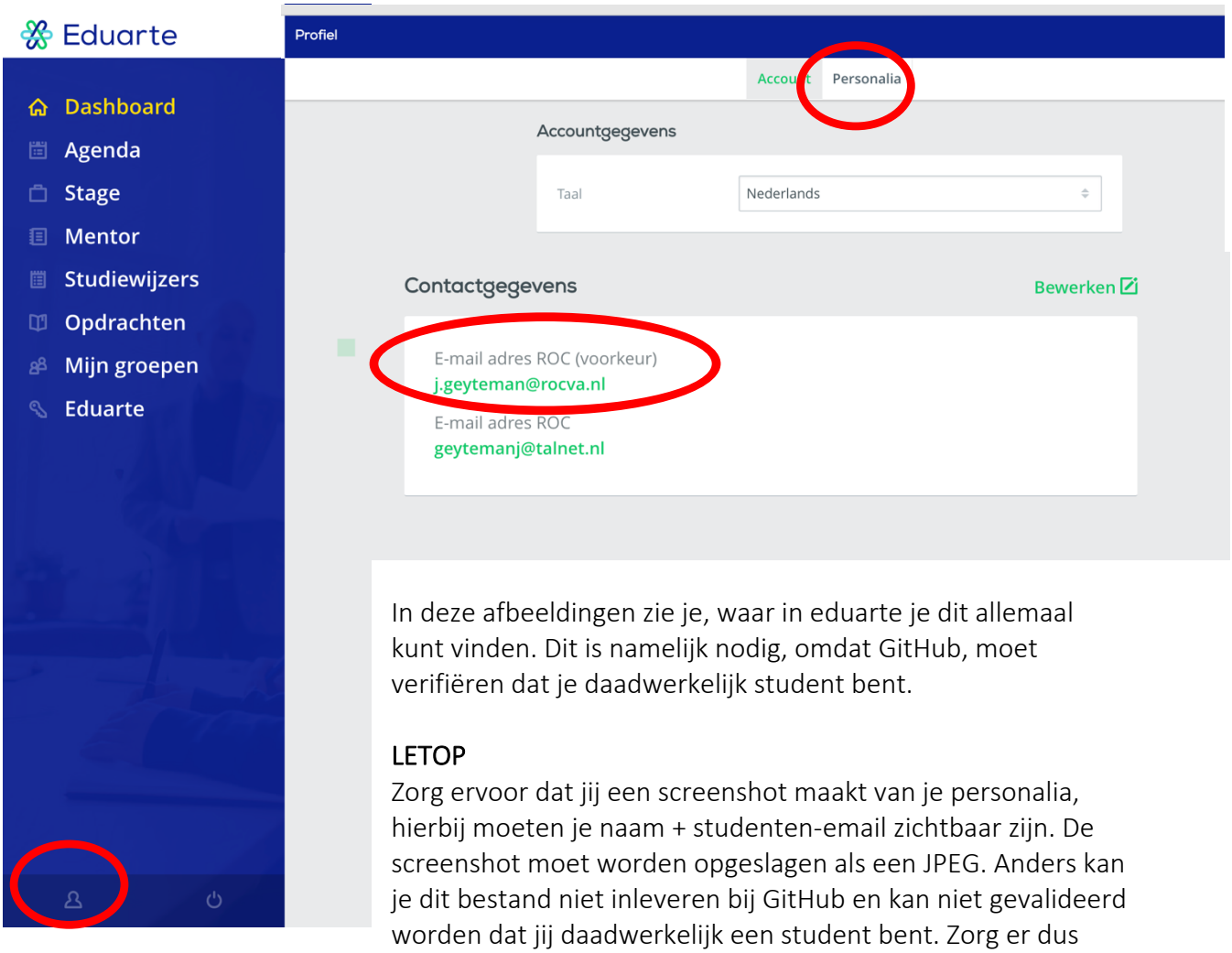
Er zijn natuurlijk nog meer GitHub termen, maar voor beginners zijn dit de belangrijkste om te weten.

1.10. Zelf aan de slag met GitHub

Account

Om zelf aan de slag te gaan met GitHub moet je meerdere dingen doen. Als eerste moet je een account aan maken op GitHub. Maak hier een account, nadat je een account gemaakt hebt, vraag hier een studenten licentie voor het developer pack aan.

Je maakt het account aan, met je @student.rocva.nl mailadres. **GEBRUIK NIET JE @talnet.nl EMAIL.** Deze email is te vinden in het personalia in eduarte.



In deze afbeeldingen zie je, waar in eduarte je dit allemaal kunt vinden. Dit is namelijk nodig, omdat GitHub, moet verifiëren dat je daadwerkelijk student bent.

LETOP

Zorg ervoor dat jij een screenshot maakt van je personalia, hierbij moeten je naam + studenten-email zichtbaar zijn. De screenshot moet worden opgeslagen als een JPEG. Anders kan je dit bestand niet inleveren bij GitHub en kan niet gevalideerd worden dat jij daadwerkelijk een student bent. Zorg er dus voor dat je dit wel doet.

Git

Om GitHub op jouw machine te kunnen gebruiken, moet je natuurlijk het programma Git hebben. Dit kun je via deze link downloaden en activeren op jouw machine.

PHPStorm van Jetbrains

Om het jezelf makkelijker te maken, raad ik aan om PHPstorm van Jetbrains te gebruiken als editor. Je kunt hier namelijk jouw GitHub account aan linken & dit maakt het werken met Git & GitHub een heel stuk makkelijker. PHPstorm is gratis voor studenten, je hebt alleen een studenten licentie nodig. De studenten licentie kun je hier aanvragen via deze link.

2. Jouw eigen repository.

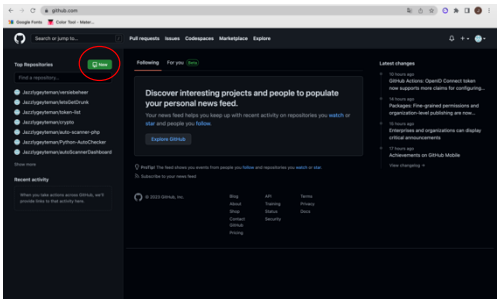
In GitHub heet een project een repository. Dit wordt ook wel afgekort met “repo”, zodra er gesproken wordt over een repo, wordt er een repository bedoeld. In een repo staat al jouw code van het hele project waar je aan werkt. In de repo, kunnen meerdere versies van jouw code staan. Dit is omdat je ook met andere mensen samen kan werken aan het project. Deze verschillende versies worden branches genoemd. De hoofd branch, is de live versie van je website. De code die hierin staat is de versie die altijd werkt. In het geval dat je bij een bedrijf werkt is dit de daadwerkelijke live versie. De hoofd branch heet binnen GitHub, de “master Branch”, of “origin”.

2.1. Branches “mergen”

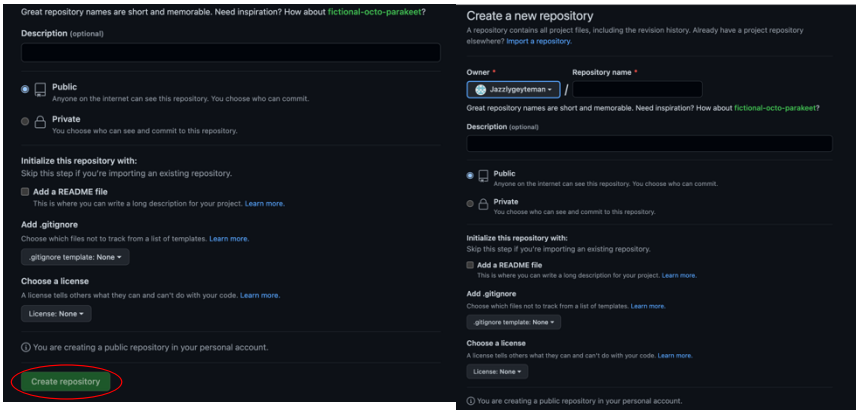
Door in branches te werken kunnen verschillende developers makkelijker simultaan aan verschillende features van de website werken, zonder dat ze elkaars code overschrijven. Maar, op een bepaald moment is een feature klaar of is het template af en dan moet alle code natuurlijk bij elkaar komen. Dat is het moment waarop je verschillende branches bij elkaar brengt waardoor een nieuwe versie van de codebase ontstaat. Het samenbrengen van branches heet ‘mergen’.

Je merged verschillende feature branches bijvoorbeeld eerst allemaal in de development branch. De development branch deploy je naar een development omgeving zodat alles als coherent geheel goed getest kan worden. Als je alle nieuwe features hebt getest en alles werkt naar behoren, dan kun je naar de staging omgeving deployen. Als daar ook alles goed werkt, dan kan alles naar de live-omgeving. Op de live omgeving draait uitsluitend de master branch. Dus als alles goed werkt op staging, merge je de development branch in de master branch en de master branch deploy je naar de live omgeving. Mocht er onverhoopt toch op live iets niet goed gaan, dan kun je met Git dus heel makkelijk een versie teruggaan en die deployen.

2.2. Hoe maak ik een Repository?

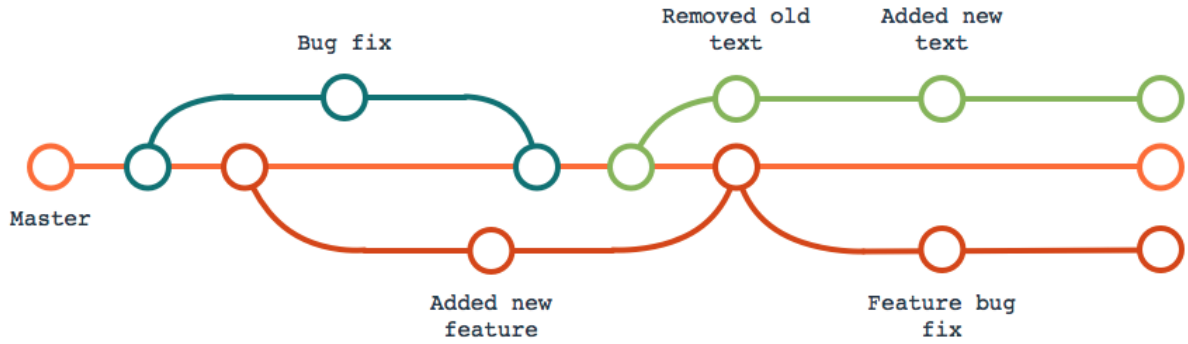


1. Je gaat naar github.com en je logt in.
2. Je klikt op de groene “new” knop.
3. Je geeft je repository een naam.
4. Zet je repository op public of private
5. Klik op de groene “create repository” knop.
6. Veel plezier met je nieuwe repository.



2.3. Branches en workflow.

Zoals je weet kun je branches mergen in GitHub. Je kunt ook nieuwe branches toevoegen, om bijvoorbeeld code in te testen. Om dit allemaal te visualiseren zijn er in GitHub branch tree's te zien.



Hierboven zie je een branch tree. Dit maakt het heel makkelijk om de verschillende versies te zien van jouw project. Hierbij is elk bolletje een **commit**. Doordat dit zo visueel is, kan je zien wat er gebeurt in elke versie.

2.4. .Gitignore

Een .gitignore-bestand is een tekstbestand dat in je git-repository is geplaatst en dat git vertelt bepaalde bestanden en mappen die je niet wilt uploaden naar je masterrepository niet bij te houden. Het heeft veel toepassingen en je zult het bijna altijd moeten configureren als je een nieuwe repository opzet.

Wat is het nut van .gitignore?

In de meeste projecten is er een verschil tussen code / config en gegenereerde bestanden van deze code of configuratie. Dit laatste is meestal niet nuttig en kan meestal worden genegeerd.

Meestal wil je alleen dat de code en configuratie worden gevolgd via git. Dit komt omdat de gegenereerde bestanden kortstondig zijn en als u ze verwijdert, kunt u ze eenvoudig opnieuw genereren. Het is vaak niet nodig om ze te downloaden, omdat ze de zaken alleen maar ingewikkelder maken en onnodige samenvoegconflicten veroorzaken.

Andere items kunnen ook worden toegevoegd aan .gitignore voor het gemak. MacOS genereert systeembestanden genaamd .DS_store, die u altijd kunt negeren. Misschien sla je API-sleutels op in .env/ dat u niet wilt worden gevolgd in bronbeheer, kunt u ze ook toevoegen. Caches, logboeken en andere uitvoer kunnen meestal worden genegeerd.

3. Versiebeheer & GitHub in a nutshell

Versiebeheer is een belangrijk aspect van het programmeren omdat het ontwikkelaars helpt bij het bijhouden van wijzigingen in de code en het gemakkelijk maken van samenwerking. Een populaire methode voor versiebeheer is het gebruik van een versiebeheersysteem zoals Git. GitHub is een cloud-based platform dat Git gebruikt voor versiebeheer. Het biedt ontwikkelaars de mogelijkheid om hun code op te slaan en samen te werken met andere ontwikkelaars.

Met GitHub, kan de ontwikkelaar een repository aanmaken, de code uploaden en deze beheren. Hierdoor kan men bijvoorbeeld een versie van de code bekijken, wijzigen en een nieuwe versie maken. Hierdoor kan men ook oude versies terugzetten als er iets misgaat. GitHub maakt het ook gemakkelijk om samen te werken met andere ontwikkelaars door hen toegang te geven tot de repository en hen in staat te stellen om wijzigingen aan te brengen.

Met GitHub kan men ook werken met branches, een nieuwe afzonderlijke versie van de code waar de ontwikkelaar aan werkt zonder de huidige versie van de code te beïnvloeden. Hierdoor kan men bijvoorbeeld experimenteren met nieuwe functionaliteiten zonder dat het de huidige werking van de code beïnvloedt. Ook kan men met GitHub issues aanmaken, bugs of verzoeken van gebruikers bijhouden en deze oplossen.

In samenvatting, GitHub is een populair platform voor versiebeheer en samenwerking in softwareontwikkeling. Het biedt ontwikkelaars de mogelijkheid om hun code te beheren, samen te werken met andere ontwikkelaars, en ook om bugs en verzoeken bij te houden. Hierdoor kan men efficiënter samenwerken en een betere code leveren.

4. Opdrachten versiebeheer les 1

4.1. Opdracht 1: Een GitHub-account maken.

Opdracht

Maak een GitHub-account aan en zorg ervoor dat jij een studenten account aanmaakt. Hierdoor krijg jij namelijk toegang tot de pro functies van GitHub.

Beoordeling

Je levert in teams een screenshot in van jouw GitHub-account. Laat zien dat je een account hebt aangemaakt & ik wil zien dat het een studenten account is doormiddel van de (pro badge). Je hebt je GitHub-account namelijk nodig voor volgende opdrachten.

4.2. Een repository aanmaken

Opdracht

Maak een repository in GitHub aan. Zorg ervoor dat deze repository op public staat. Als je deze repository hebt aangemaakt, wil ik minimaal 1 file zien waar code in staat.

Beoordeling

Je maakt een screenshot van je repository en levert deze in op teams. Moet je een link bijvoegen van de repository, hierdoor kan ik namelijk zien hoe je de code hebt toegevoegd

4.3. Een programma uploaden in GitHub.

Opdracht

Je maakt een simpele html pagina. Deze zet je in een nieuwe GitHub repository.

Beoordeling

Je deelt een link van je repository. Zorg er dus voor dat deze op public staat. Anders kan ik niet zien wat je hebt gedaan in je repository.

Eindbeoordeling.

Voor deze 3 opdrachten krijg je 3 losse cijfers. Hiervan neem ik het gemiddelde. Dit is dan je eindbeoordeling van de opdrachten van les 1.

5. Opdrachten versiebeheer les 2

5.1. Branches aanmaken.

Opdracht

Maak een branch aan in een nieuwe repository. Geef deze branch de naam “development”. Vul deze branch met een kleine applicatie. Hoe je dit vormgeeft mag je zelf weten, als er maar een repository is, met een branch “development”.

Beoordeling

Je maakt een screenshot van je repository en branches. Deze lever je in op teams. Ik wil bij deze screenshot ook weer een link van je repository zodat ik dit kan zien & na checken.

5.2. Een pull request aanmaken.

Opdracht

Je code staat soms op een repository, maar niet op je device. Hiervoor moet je een pull request aanmaken. Ik wil dat er vanuit een bestaande repository een pull request gemaakt wordt. Zorg ervoor dat je kan uitleggen wat en vooral waarom je deze pull request hebt aangemaakt.

Beoordeling

Je levert een screenshot van je pull request in. Hierbij wil ik toelichting, waarom je het pull request maakte en hoe.

5.3. Een groepsproject.

Opdracht

Maak een nieuwe repository aan. Zorg ervoor dat jij je samen met 2 andere studenten gekoppeld wordt aan een repository, maak gebruik van verschillende braches en zorg ervoor dat je code uploadt in je repository.

Beoordeling

Lever een link van de gezamelijke repository in op teams.

Eindbeoordeling.

Voor deze 3 opdrachten krijg je 3 losse cijfers. Hiervan neem ik het gemiddelde. Dit is dan je eindbeoordeling van de opdrachten van les 2.

6. Opdrachten versiebeheer les 3

6.1. Branches mergen.

Opdracht

Als je samenwerkt zal je erachter komen dat je soms branches moet mergen. Als opdracht wil ik dat jullie in een groepje van 3 personen. Uiteindelijk je branches mergen in je repository.

Beoordeling

Ik wil dat jullie een screenshot sturen van de branch tree/workflow. Deze leveren jullie individueel in op teams.

6.2. Merge conflicts oplossen.

Opdracht

Soms kom je erachter dat als je branches merged in jouw repository dat je een merge conflict hebt. Ik wil dat jullie deze bewust creëren en oplossen. Dit mag in dezelfde repository als in opdracht 6.1

Beoordeling

Lever een screenshot hiervan in op teams. Ieder levert individueel een screenshot in.

Eindbeoordeling.

Voor deze 2 opdrachten krijg je 2losse cijfers. Hiervan neem ik het gemiddelde. Dit is dan je eindbeoordeling van de opdrachten van les 3.