



Git Standaards en Richtlijnen

Auteurs: Knights Harnasses team
Version: 0.14
Date: April 2021



Documentbeheer

Versiebeheer

Versie	Datum	Auteur	Wijzigingen
0.1	17-12-2019	Ganesh Nikumbh	Initiele versie
0.2	08-01-2020	Ganesh Nikumbh	Gebruik van Server Manager toegevoegd
0.6	06-02-2020	Marco Lemmers	Toevoeging van extra achtergrond informatie over GIT en document vertaald naar het Nederlands
0.7	07-02-2020	Jetty Schenk	Review en spellingcontrole
0.8	11-02-2020	Marco Lemmers	Verwerking review commentaar Dorothy Lekston
0.9	21-2-2020	Marco Lemmers	Toevoeging werkwijze met GIT GUI Layout aangepast
0.10	03-04-2020	Ganesh Nikumbh	Gewijzigd deel 7: Nieuwe aanpak voor het exporteren en importeren van Datastage Objects
0.11	27-05-2020	Ganesh Nikumbh	Gewijzigd deel 5: Aangepast aan nieuwe werkwijze rondom GIT Workflow en GIT richtlijnen
0.12	08-09-2020	Henri Mijnders	Gewijzigd deel 7: Aanpassing python dsx splitter uitleg (script aangepast met aanmaken Manifest bestand)
0.13	05-10-2020	Jetty Schenk	Aanpassingen naar aanleiding van overgang naar OTAP
0.14	29-04-2021	Tijmen Groustra	Toevoegen comment over apart aanbieden DROP statements voor database objecten

Distributielijst

Versie	Naam afdeling	Naam contactpersoon	Opmerkingen

Er is afgestemd (advies/overleg) geweest met de volgende partijen:

Naam afdeling	Naam contactpersoon	Opmerkingen

Inhoud

Documentbeheer	2
1 Algemene beschrijving	5
1.1 Introductie	5
1.2 Scope / afbakening	5
2 Algemene richtlijnen	5
3 GIT achtergrondinformatie	6
4 Mappenstructuur voor het DIM project in GIT	7
5 GIT Workflow en GIT richtlijnen	10
5.1 GIT Workflow	10
5.2 Gebruik van commit boodschappen en standaard boodschap formaat	11
5.3 Manifest file	12
6 Werken met GIT Gui en GIT Bash	14
6.1 ...clone een nieuwe werk directory?	14
6.2 ...wat is de status van mijn werk directory?	15
6.3 ...aanmaken van een nieuwe branch?	16
6.4 ...voeg bestanden die gecommited moeten worden toe aan de index?	18
6.5 ...verwijderen van bestanden uit de index/staging area?	19
6.6 ...commit van wijzigingen naar mijn lokale repository?	20
6.7 ...tonen van commits in grafische weergave	20
6.8 ...push mijn wijzigingen naar de centrale repository?	21
6.9 ...hoe te mergen?	23
6.10 ...verwijderen van een branch?	24
6.11 ...verwijderen van een remote branch?	25
6.12 ...wijzigen van commits die nog niet zijn gepushed?	26
6.13 ...terugkeren naar een vorige commit (ongedaan maken van wijzigingen)	27
6.14 ...ongedaan maken / rol back van wijzigingen die zijn gepushed?	28
6.15 ...ongedaan maken van een commit?	28
6.16 ...hoe merge conflicten op te lossen?	29
6.17 ...verwijderen van lokale branches die niet meer bestaan in de remote repository?	31
7 Het gebruik van GIT voor DataStage objecten	32
7.1 Exporteren van DataStage objecten (componenten) uit DataStage Designer	32
7.2 dsxsplitters.py hulpprogramma voor het splitsen van multi object dsx bestand en creëren van een manifest bestand	38
7.3 Importeren van DataStage objecten (componenten) met behulp van Datastage Designer ...	41



8	Scenario's.....	42
8.1	Zet de componenten voor een nieuwe user story in Git.....	42

1 Algemene beschrijving

1.1 Introductie

Dit document beschrijft de standaarden en richtlijnen rond het gebruik van de GIT versie beheer software binnen UWV.

1.2 Scope / afbakening

In dit document wordt de GIT versiebeheer software beschreven, de detail werking van een groot aantal GIT commando's, de integratie met DataStage en de daarbij te hanteren procedures binnen UWV.

De hieraan gerelateerde deployment van software componenten in de OTAP omgeving zit niet in de scope van dit document. Meer informatie hierover kunt u vinden in het Deployment_document_DataFabriek_v1.2.

2 Algemene richtlijnen

GIT Bash versus GIT GUI

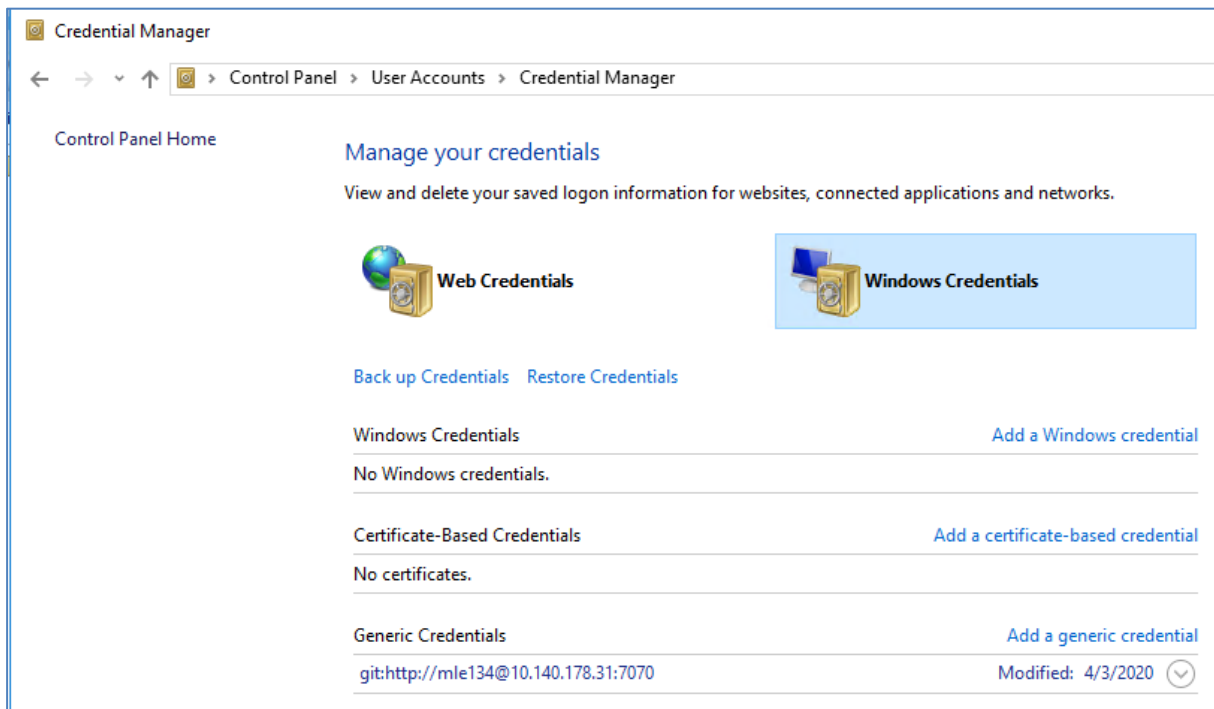
Binnen UWV maken we gebruik van de GIT client software voor Windows. Deze software is beschikbaar op de OTAP omgeving. Bij het gebruik van GIT heb je de keuze tussen de GIT Bash client (een command line interface) en de GIT GUI. GIT Bash biedt de meeste GIT functionaliteit, en wordt daarom in de praktijk veel gebruikt. De GIT GUI bevat echter de basis functionaliteit, en zal in de praktijk vaak afdoende zijn. De keuze tussen gebruik van GIT Bash en GIT GUI voor dagelijks gebruik ligt daarmee bij de individuele ontwikkelaar.

Branching strategie en deployment

In dit document wordt een zeer voorlopige werkwijze beschreven voor branching. Deze werkwijze dient verder uitgewerkt te worden als deel van de deployment standaarden.

Git wachtwoord management

Voor het beheer van het GIT wachtwoord kan de 'Windows Credential Manager' manager worden gebruikt.



De Git password manager is niet altijd per default geconfigureerd. Check hier voor de Git configuratie:

```
git config --list
```

De output dient de volgende regel bevatten:

```
credential.helper=manager
```

Als dat niet zo is, activeer de Git password manager dan eenmalig als volgt:

```
git config --global credential.helper manager
```

Bij een Pull of Push actie wordt je nog eenmalig om userid/password gevraagd, waarna deze automatisch middels Windows credential manager worden opgeslagen.

3 GIT achtergrondinformatie

GIT is een veel gebruikt versiebeheer systeem dat ontwikkeld is voor gedistribueerd versiebeheer van software componenten (ter info: GIT is een betekenisloze naam; het is geen afkorting of acroniem). Er is een centrale GIT repository (de 'remote', ook wel 'origin' genoemd) van waaruit elke ontwikkelaar een complete versie van de software componenten kopieert naar zijn/haar eigen lokale omgeving. Dit kan een initiële kopie zijn (een 'clone') of een verversing wanneer al eerder een clone gemaakt is (een 'checkout' of 'pull'). In deze lokale omgeving worden wijzigingen uitgevoerd en wordt wijzigings-historie opgebouwd middels 'commits'. Wanneer de

softwarecomponenten gereed zijn, worden deze naar de centrale repository gezonden middels een 'push', waarna deze de nieuwe basis vormen voor verdere ontwikkeling.

De verschillende versies van een applicatie worden beheerd middels 'branches'. Er zijn meerdere werkwijzen voor het gebruik van branches. Branches kunnen gedefinieerd worden per user-story, per release, per OTAP omgeving, of combinaties daarvan.

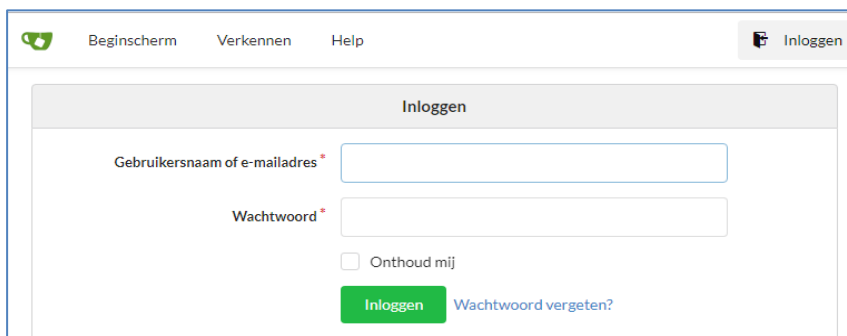
Een speciale branch is de 'master' branch. Deze representeert de software die gereed is voor productie. Wijzingen worden nooit direct op de master branch uitgevoerd!

In volgende hoofdstukken worden de details rond het gebruik van GIT beschreven.

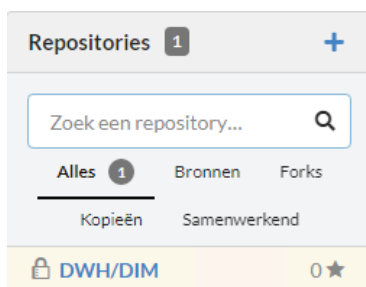
4 Mappenstructuur voor het DIM project in GIT

In de centrale GIT repository is project DIM gedefinieerd binnen het domein DWH.

Gebruik URL <http://10.140.178.31:7070> en je user-id en OTOD wachtwoord om op deze repository aan te melden.

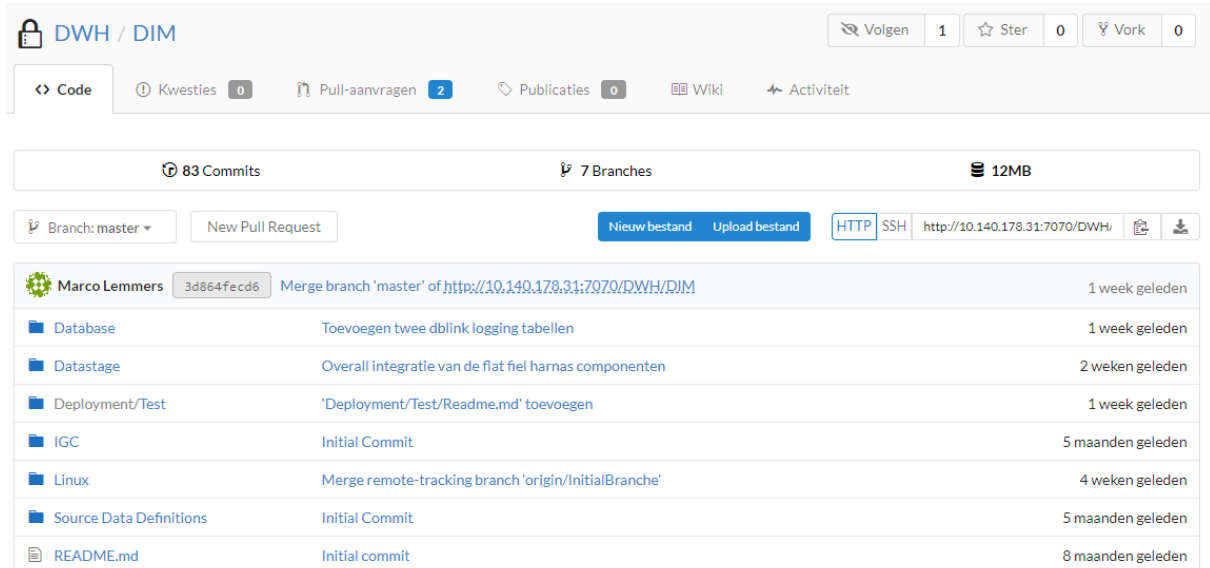


Selecteer repository DWH/DIM:



In het DIM project is de globale standaard mappenstructuur te zien die gebruikt wordt voor het organiseren van de te ontwikkelen softwarecomponenten.

Alle DataStage componenten worden opgeslagen in de **DataStage** map, alle database scripts in de **Database** map etc.



The screenshot shows a Git repository interface for 'DWH / DIM'. At the top, there are buttons for 'Volgen' (1), 'Ster' (0), and 'Vork' (0). Below this is a navigation bar with 'Code', 'Kwesties' (0), 'Pull-aanvragen' (2), 'Publicaties' (0), 'Wiki', and 'Activiteit'. The main section displays repository statistics: 83 Commits, 7 Branches, and 12MB. Below this are buttons for 'Branch: master', 'New Pull Request', 'Nieuw bestand', 'Upload bestand', and a link to the repository via HTTP or SSH. A list of commits is shown, including a merge by Marco Lemmers and several folder-specific commits for Database, Datastage, Deployment/Test, IGC, Linux, Source Data Definitions, and README.md.

Commit	Message	Time
3d864fec6	Merge branch 'master' of http://10.140.178.31:7070/DWH/DIM	1 week geleden
	Database: Toevoegen twee dblink logging tabellen	1 week geleden
	Datastage: Overall integratie van de flat fiel harnas componenten	2 weken geleden
	Deployment/Test: 'Deployment/Test/Readme.md' toevoegen	1 week geleden
	IGC: Initial Commit	5 maanden geleden
	Linux: Merge remote-tracking branch 'origin/InitialBranche'	4 weken geleden
	Source Data Definitions: Initial Commit	5 maanden geleden
	README.md: Initial commit	8 maanden geleden

De **Datastage** map is verder onderverdeeld naar DataStage project en de mappen binnen het DataStage project. In onderstaande schermprintjes zie je de map voor het project DIM_ontwikkel, en de sub mappen die daarbinnen worden gebuikt.

De **Linux** mappenstructuur is gelijk aan de structuur op de Linux server. Hier worden shell scripts, parameterfiles en schemafiles opgeslagen.

De **Deployment** mappenstructuur heeft submappen voor elke omgeving (Test, Acceptatie en Productie). In deze submappen komen de manifest files die nodig zijn voor het deployen van code. In de **Database** mappen structuur bestaat per database schema een subfolder waarin alle SQL scripts staan opgenomen die met die specifieke database user moeten worden uitgevoerd.

In een SQL script kunnen dus nooit database objecten voor *verschillende* database schema's worden gedefinieerd.

Let er bij het maken van SQL script (mn DDL) dat 'schone' SQL wordt gebruikt. Definieer alleen die non-default properties die ook werkelijk nodig zijn, gebruik geen quoted tabel of kolomnamen en kopieer nooit in SQL Developer gegenereerde code 1-op-1 in een SQL script.



DWH / DIM

Volgen 1 Ster 0 Vork 0

Code Kwesties 0 Pull-aanvragen 0 Publicaties 0 Wiki Activiteit

26 Commits 1 Branch

Branch: master DIM / Datastage Nieuw bestand Upload bestand Geschiedenis

gni020 f725528335 Initial Commit	20 uur geleden
..	
DIM_ontwikkel Initial Commit	20 uur geleden
README.md Initial Commit	20 uur geleden

README.md

DIM

DWH / DIM

Volgen 1 Ster 0 Vork 0

Code Kwesties 0 Pull-aanvragen 0 Publicaties 0 Wiki Activiteit

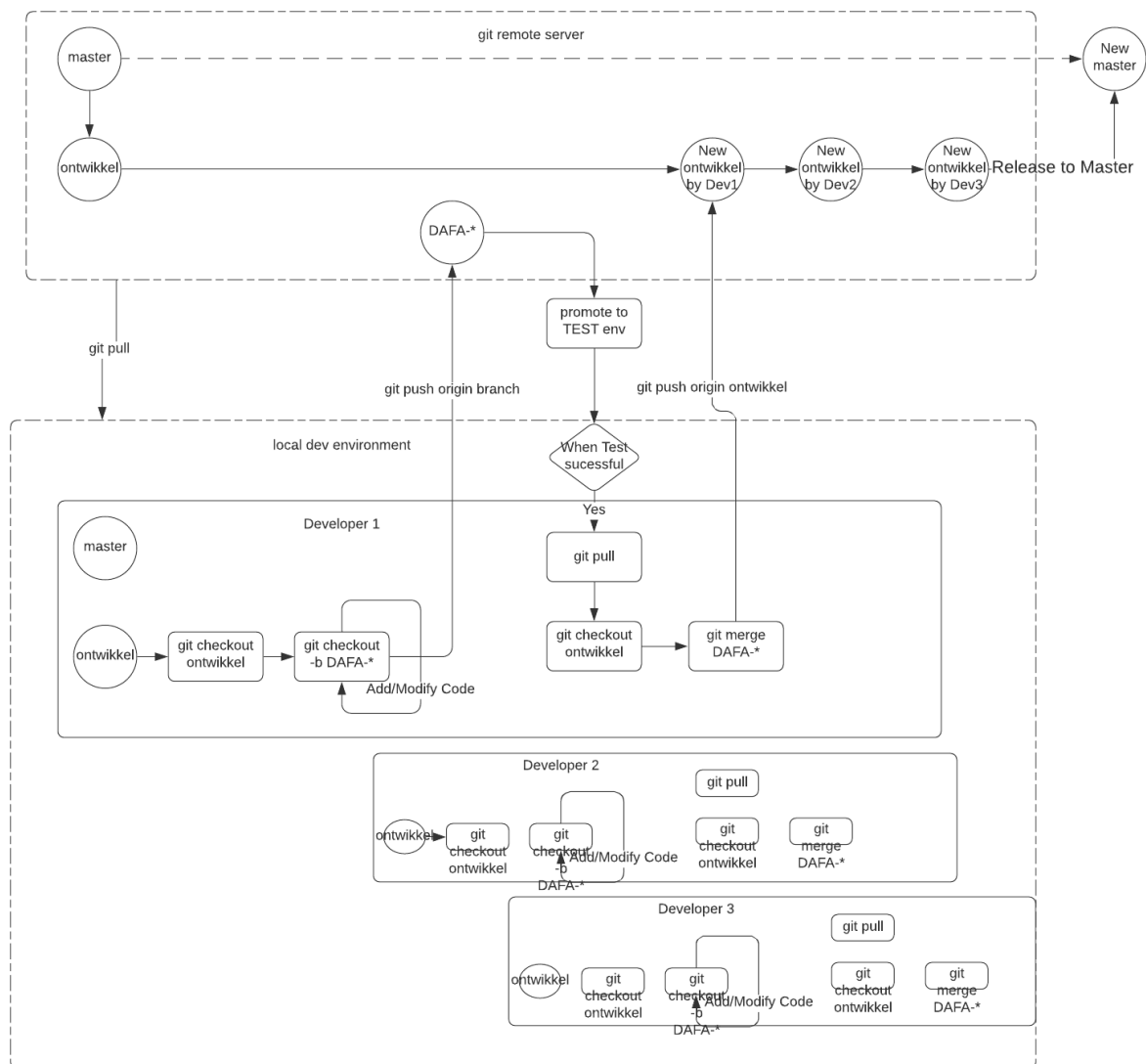
26 Commits 1 Branch

Branch: master DIM / Datastage / DIM_ontwikkel Nieuw bestand Upload bestand Geschiedenis

gni020 a3df50415c Initial Commit	22 uur geleden
..	
00 Schedule Initial Commit	22 uur geleden
10 Staging Initial Commit	22 uur geleden
20 Bronzone Initial Commit	22 uur geleden
30 Integratiezone Initial Commit	22 uur geleden
40 Businesszone Initial Commit	22 uur geleden
90 Generic Initial Commit	22 uur geleden
99 Zandbak Initial Commit	22 uur geleden
Readme.md Initial Commit	22 uur geleden

5 GIT Workflow en GIT richtlijnen.

5.1 GIT Workflow



Hierboven is een schematische weergave van het versiebeheerproces in GIT. Het versiebeheerproces bestaat uit de volgende stappen:

1. Voer '**GIT clone**' uit om de master repository te kopiëren naar lokale folder **V:\DWH\medewerkers\<puik>**. (Wanneer al eerder een 'clone' was uitgevoerd, dan wordt '**GIT pull**' gebruikt om de lokale repository te verversen met de meest actuele code in de remote repository).

2. Voer een '**checkout**' uit naar de Ontwikkel branch.
3. Maak een nieuwe lokale branch aan voor de software wijziging. De branch krijgt de naam van de user story, dus DAFA-*.
4. Voer een '**checkout**' uit naar deze branch. Vanaf dat moment worden wijzigingen aan de nieuwe branch toegevoegd.
5. Voer wijzigingen uit op de lokale files (toevoegen, wijzigen, verwijderen)
6. Voeg de gewijzigde files toe aan de GIT index middels '**GIT add**'. De index houdt de lijst van gewijzigde files vast die gecommit moeten worden. De GIT index wordt ook wel 'staging area' genoemd.
7. Voer een '**commit**' uit op de lokale branch, en geef daarbij een commit message mee (zie sectie 5.2). Elke commit wordt binnen GIT uniek geïdentificeerd met een SHA1 key, bv. a58bd5032df63af46107b700e4f6c2eb83b517d8
8. Wanneer de code klaar is om te worden ingezet in testomgeving, maak een manifest file (zie par. 5.3) in de Deployment/Test map en voer een '**push**' uit van de lokale branch naar de remote repository met **git push origin DAFA-***. De naam van de manifest file moet hetzelfde zijn als de branch naam en met de extensie <.manifest>
9. Een andere ontwikkelaar (wie??) deployt de branch naar de testomgeving.
10. Test van de code in de testomgeving.
11. Bij succesvolle test ververs de lokale repository met een '**GIT pull**'.
12. Voer een '**checkout**' uit naar de Ontwikkel branch.
13. Voer een '**merge**' uit van de lokale DAFA-* branch en de lokale ontwikkel branch.
14. Voer een '**push**' uit van de lokale ontwikkel naar de remote ontwikkel. Hiermee is een nieuwe baseline in de remote ontwikkel ontstaan.
15. Verwijder de lokale branch (deze heeft geen functie meer)
16. Verwijder de remote branch

(Deze werkwijze rond gebruik van branches kan nog worden aangepast n.a.v. de ervaringen die binnen het project worden opgedaan)

5.2 Gebruik van commit boodschappen en standaard boodschap formaat

Een commit boodschap dient in detail weer te geven wat de aard is van een wijziging, dus gebruik een zinvolle en voldoende uitgebreide tekst (zonder te verzanden in proza).

Gebruik in de tekst de gebiedende wijs en niet de voltooid verleden tijd; hiermee liggen de boodschappen in lijn met de automatische boodschappen die gegenereerd worden met 'GIT merge' en 'GIT revert'.

Voorbeeld: gebruik 'Herstel fout in tekst' in plaats van 'Fout in tekst hersteld'.

Gebruik het volgende formaat voor commit boodschappen:

- **Regel 1:** <Module of Applicatiegebied>: Een korte beschrijving van de wijziging. Gebruik hiervoor een enkele regel (maximaal 76 karakters).
- **Line 2:** Lege regel
- **Line 3+:** Uitgebreide beschrijving van de wijziging. Leg uit waarom de wijziging is gedaan, en hoe deze is uitgevoerd. Lever hierbij zoveel detail als nodig is voor anderen om de wijziging te begrijpen. Gebruik lijnlengtes van maximaal 76 karakters.

Gebruik van labels

Het gebruik van labels in een boodschap kan de boodschap verder verhelderen/typeren. Denk daarbij aan:

@correctie - wanneer er sprake is van een correctie.

@functionaliteit - wanneer er sprake is van een nieuwe feature of uitbreiding daarvan.

<bevinding>: XXXXX – wanneer een specifieke bevinding is opgelost.

Voorbeelden:

Een eenvoudige boodschap:

```
SRC_BAS: Wijzig het polling interval naar 120 minuten in het Polling script
Voor de bron BAS arriveren de files tussen 10.00 en 12:00 uur. Om deze periode af
te dekken is het polling interval gewijzigd van 60 minuten naar 120 minuten.
Afgestemd met de Bron.
```

Boodschap voor een functionele wijziging:

```
SRC_BAS: Voeg nieuwe kolom toe aan doel tabel t.b.v. berekening van afgedragen
belasting.
Nieuwe kolom is nodig omdat afnemer dit gegeven nodig heeft in nieuwe rapportage.
@functionaliteit
```

Bug fix Message voorbeeld:

```
SRC_BAS: Corrigeer berekening van afgedragen belasting
Afgedragen belasting is aangepast.
@correctie
bevinding: 123456
```

5.3 Manifest file

Een manifest file is een tekstbestand met een lijst van alle bestandsnamen die gedeployed moeten worden in een bepaalde user story samen met het pad waar ze gevonden kunnen worden in de GIT repository. Hieronder zie je een voorbeeld voor de verschillende soorten objecten en een voorbeeld van een manifest file in GIT.

Voor alle Datastage objecten geldt: Het pad start vanaf de DIM_ontwikkel project map (exclusief DIM_ontwikkel)

Bijv. \10 Staging\50 Jobs\DL_Bestand_Verwerking\3

Pakbon\job_DL_PakbonControlesMetadata.dsx

of \90 Generiek\10 Parameter Sets\par_DIM_Dbs.dsx

Let op dat het pad de windows pad conventies volgt.

Note: De dsxspliter (zie hoofdstuk 7) zal met het uitpakken van de dsx bestanden ook een manifest file creëren voor de uitgepakte dsx bestanden. Dit manifest kan je aanvullen met de nog missende objecten.

Voor alle Database objecten geldt: Het pad start vanaf de schema map in GIT. Als een bepaalde schema map nog niet bestaat in GIT, maak hem dan aan.

Bijv. \DIM_LOGGING\cre_DL_Logging_Sequence.sql

Let op dat de sql scripts uitgevoerd zullen worden in de volgorde waarin ze in de manifest file staan. Dus let goed op de volgorde bij het maken van een manifest file.

Let ook op dat database scripts waarbij tabellen worden verwijderd middels een DROP statement in een aparte branche aangeboden moeten worden vanwege back-up procedures.

Voor alle Linux script objecten geldt: Het pad start vanaf de Linux map in GIT (exclusief de Linux map)

Bijv. \10staging\00scripts\DetecteerPakbon.sh

Zie bijgevoegd voorbeeldmanifestbestand voor een user story DAFA-100



6 Werken met GIT Gui en GIT Bash

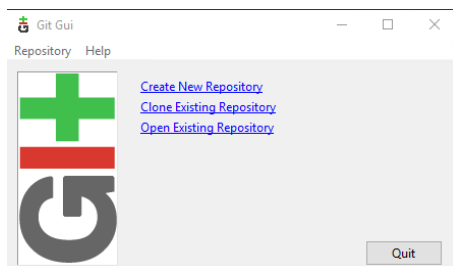
In dit hoofdstuk worden de belangrijkste met GIT uit te voeren acties beschreven. Hierbij worden zowel de werkwijze met GIT Gui als met GIT Bash toegelicht.

6.1 ...clone een nieuwe werk directory?

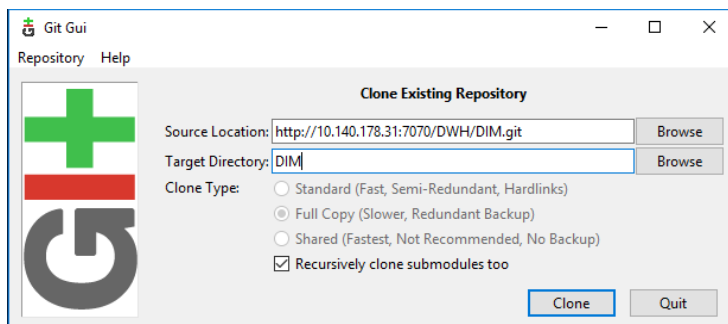
In GIT is elke werk directory in zichzelf een complete repository, en bevat de volledige historie van alle applicatie componenten. De eerste stap in het werken met GIT is het uitvoeren van een 'clone' van de centrale repository.

GIT Gui

Ga naar je persoonlijke directory **V:\DWH\medewerkers\<puik>**. Klik de rechtermuis, en selecteer de optie 'Git GUI Here'. GIT start op met het volgende scherm:

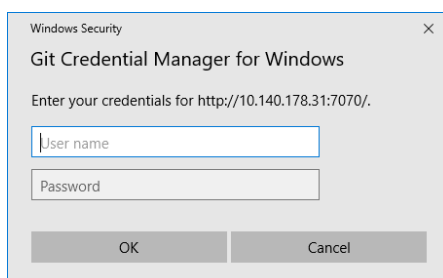


Selecteer 'Clone Existing Repository'.



Vul het adres in van het DIM project binnen de GIT repository:
`http://10.140.178.31:7070/DWH/DIM.git`

Initieel kan GIT om je user-id en wachtwoord vragen; gebruik hier je <puik>.



GIT Bash

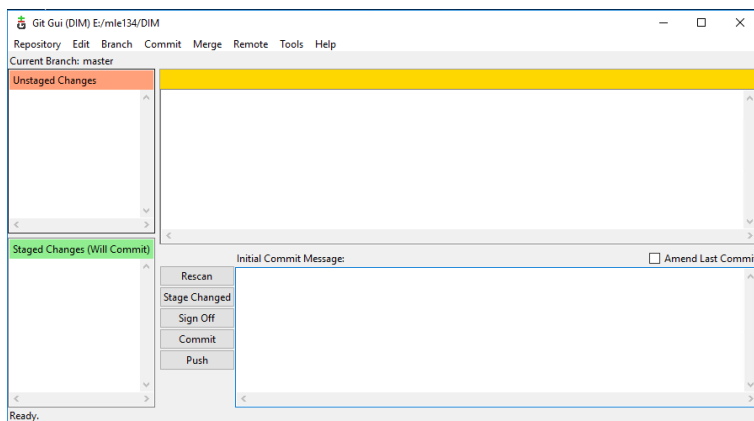
Navigeer met het 'cd' commando naar directory **V:\DWH\medewerkers\<puik>** en eventueel daarbinnen naar een subdirectory die je wilt gebruiken voor de lokale DIM repository. Voer vervolgens het clone commando uit. Hiermee wordt folder DIM aangemaakt met daarin alle geversioneerde DIM applicatie componenten.

```
$ git clone http://10.140.178.31:7070/DWH/DIM.git
```

6.2 ...wat is de status van mijn werk directory?

GIT Gui

Na het uitvoeren van de clone, wordt het volgende venster geopend:



GIT Gui toont hier de status van de werkdirectory.

- In het venster 'Unstaged Changes' staan de gewijzigde bestanden die nog niet aan de index zijn toegevoegd.
- In het venster 'Staged Changes' staan de gewijzigde bestanden die in de index zijn opgenomen, en meegenomen worden in de volgende commit.
- Bij 'Current Branch' staat de naam van de branch waarin je je bevindt.

Gebruik de knop 'Rescan' om de gegevens in de vensters te verversen.

GIT Bash

Voer het command 'git status' uit. Deze levert de volgende output:

```
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "GIT rm --cached <file>..." to unstage)
#
#       new file:   robin
#
# Changed but not updated:
```

```
# (use "GIT add <file>..." to update what will be committed)
#
#       modified:   batman
#
# Untracked files:
# (use "GIT add <file>..." to include in what will be committed)
#
#       alfred
```

'Changes to be committed' toont de lijst van gewijzigde bestanden die worden meegenomen bij de volgende commit.

'Changed but not updated' toont de lijst van bestanden die zijn gewijzigd, maar die niet worden ge-commit, tenzij expliciet aangegeven.

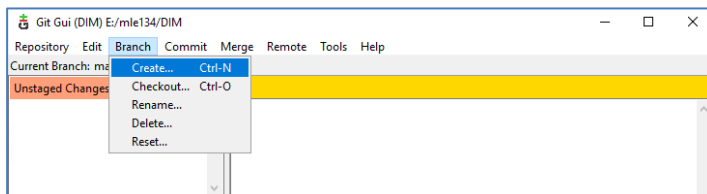
'Untracked files' toont de lijst van bestanden die nog niet in de index staan.

6.3 ...aanmaken van een nieuwe branch?

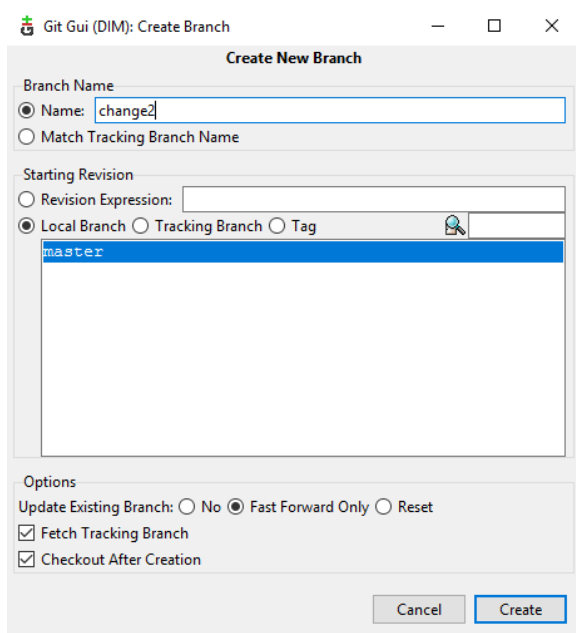
Wijzigingen worden nooit direct op de master branch uitgevoerd, maar worden altijd op aparte branches uitgevoerd.

GIT Gui

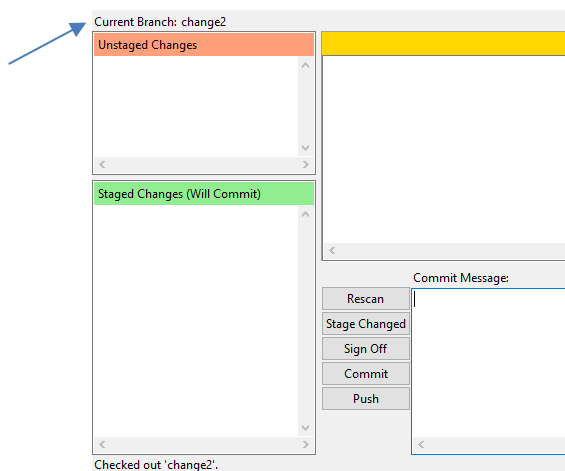
Maak een nieuwe branch aan door in menu 'Branch' optie 'Create' te selecteren:



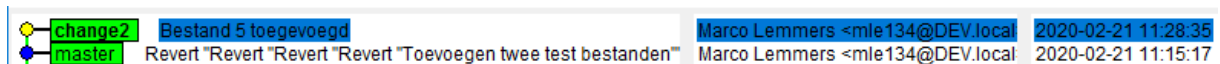
Vul de naam van de branch in, en selecteer de branch waarop de nieuwe branch gebaseerd moet zijn (in ons geval altijd de master branch):



Met de optie 'Checkout After Creation' wordt na het aanmaken van de branch meteen naar die branch overgestapt:



Wanneer vervolgens wijzigingen worden uitgevoerd (bv het toevoegen van een extra bestand), dan ziet dat er na commit als volgt uit:



GIT Bash

Het alleen aanmaken van een branch gaat als volgt:

```
$ git branch <branch name>
```

Je stapt in dit geval niet over naar e context van de branch.

Je kunt als volgt een branch aanmaken en overstappen naar de context van de branch:

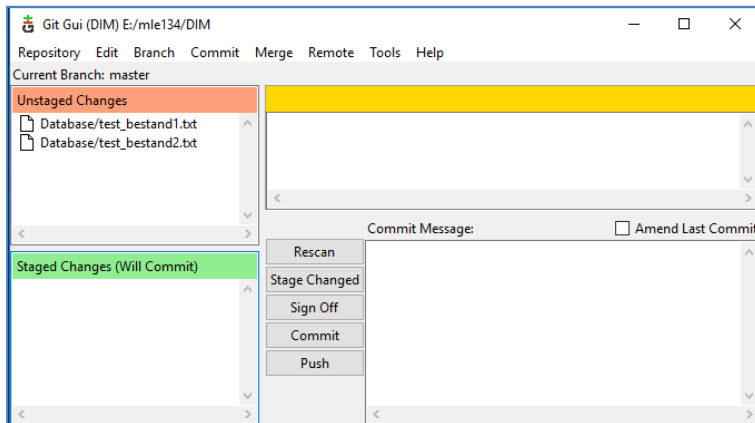
```
$ git checkout -b <branch name>
```

6.4 ...voeg bestanden die gecommited moeten worden toe aan de index?

In GIT bevat de index / staging area de lijst van bestanden die gecommited moet worden.

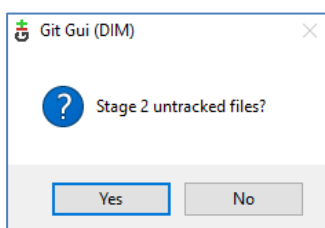
GIT Gui

Wanneer een bestand is gewijzigd of een nieuw bestand is aangemaakt, zie je deze na een 'Rescan' bij de 'Unstaged Changes staan':

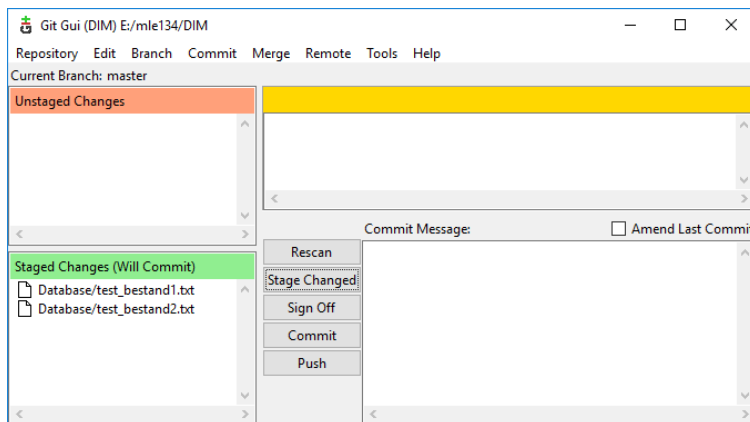


Kies de optie 'Stage Changed' om de file in de index op te nemen. Hiermee worden alle 'Unstaged Changes' in de index geplaatst. Het is niet mogelijk om hier individuele bestanden te selecteren.

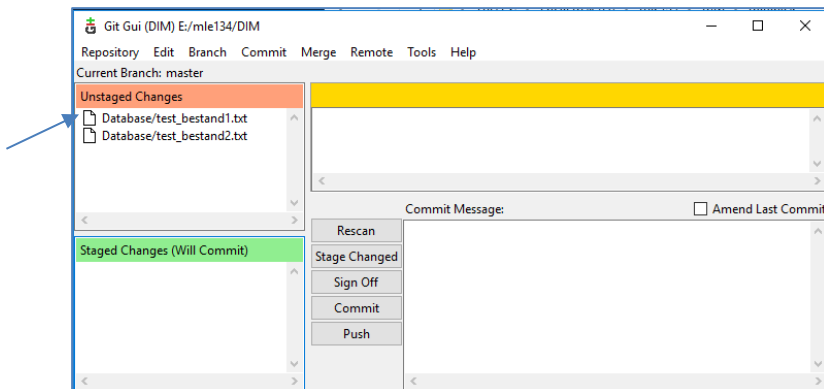
Bevestig de actie:



Dit geeft het volgende resultaat:



Hetzelfde kan ook sneller worden bereikt door met de linkermuis te klikken op het file-icoon van het betreffende bestand:



Hiermee heb je de controle over individuele files.

GIT Bash

Gebruik **add** om files toe om gewijzigde bestanden toe te voegen:

```
$ git add <file1> <file2>
```

Het toevoegen van alle bestanden in de werk directory:

```
$ git add .
```

6.5 ...verwijderen van bestanden uit de index/staging area?

GIT Gui

Door met de linkermuis op het file icoon van het bestand in de 'Staged Changes' te klikken, verwijder je het bestand uit de index, en komt het weer in de 'Unstaged changes' te staan.

GIT Bash

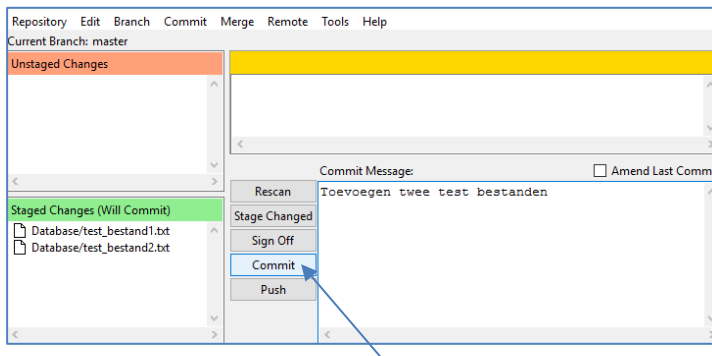
Verwijder bestanden uit de index op de volgende manier:

```
$ git rm <file>
```

6.6 ...commit van wijzigingen naar mijn lokale repository?

GIT Gui

Geef in het scherm 'Commit message' een commit boodschap, en selecteer 'Commit' om de 'Staged Changes' te committen.



GIT Bash

Commit een enkele file als volgt:

```
$ git commit <file> -m "commit boodschap"
```

Zowel 'Changes to be committed' als 'Changed but not updated' wijzigingen worden gecommit.

Voeg altijd een commit boodschap toe. Als de -m optie niet wordt meegegeven, start GIT een editor voor het invoeren van de commit boodschap (zoals gespecificeerd in de EDITOR omgevingsvariabele). De default hier is een Unix vi editor.

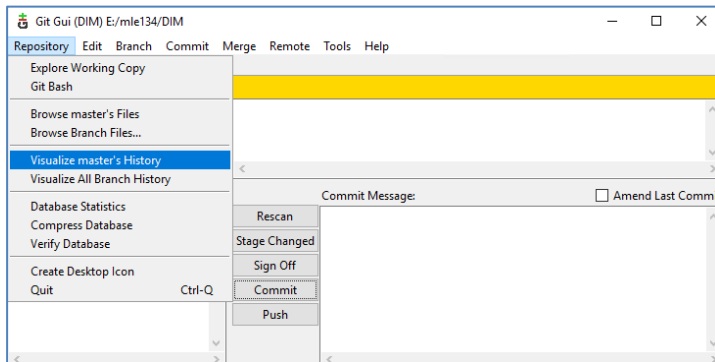
Commit alle wijzigingen in de index door de <file> weg te laten:

```
$ git commit -m "commit boodschap"
```

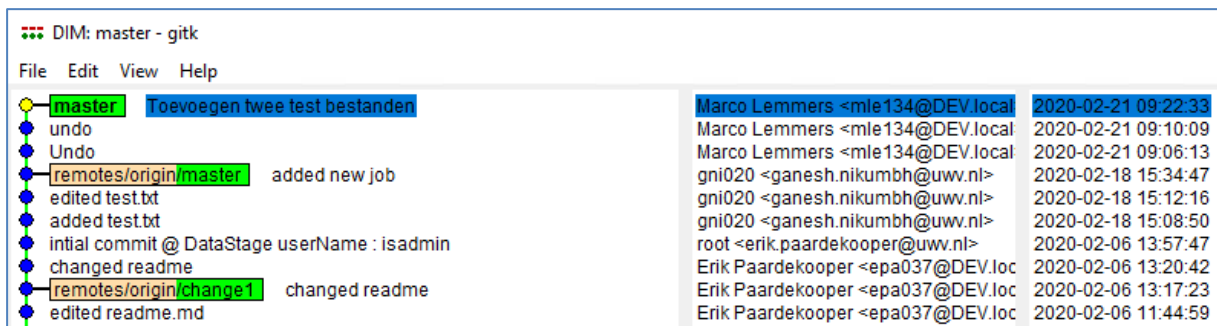
6.7 ...tonen van commits in grafische weergave

GIT Gui

In het menu 'Repository' zijn de opties 'Visualize master's History' en 'Visualize All Branch History'. Met deze opties wordt de lokale commit historie getoond.



Hier zien we ook de laatst gedane commit terug:



GIT Bash

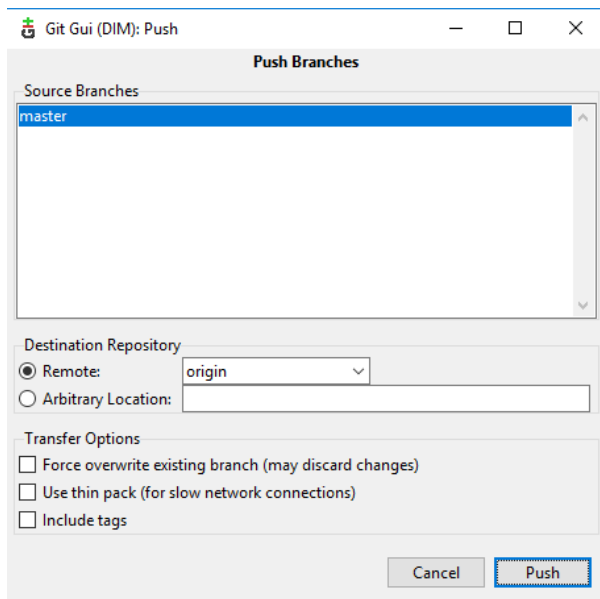
Binnen GIT BASH kun je de commit geschiedenis op een eenvoudige grafische manier weergeven:

```
git log --all --decorate --oneline --graph
```

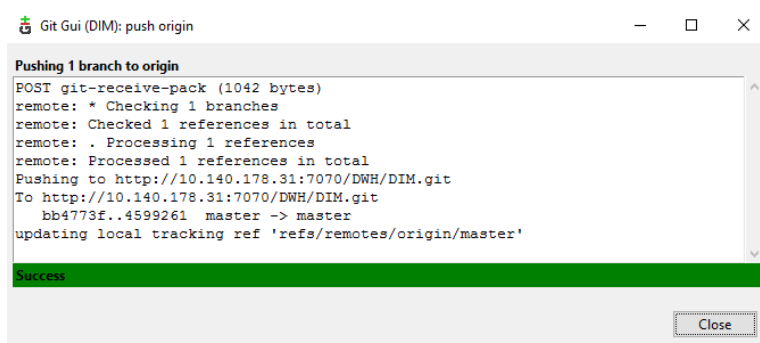
6.8 ...push mijn wijzigingen naar de centrale repository?

GIT Gui

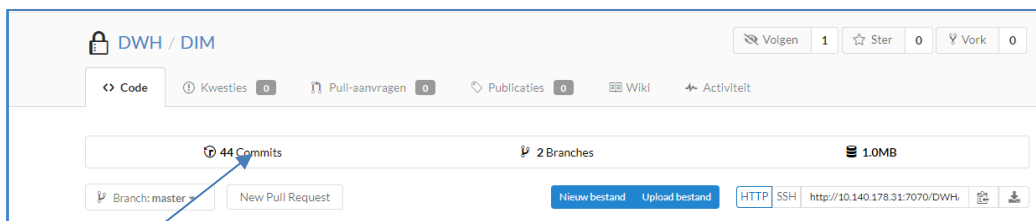
Met de optie 'Push' in het hoofdscherm worden de lokale commits gepushed naar de remote repository. Bij de push wordt het volgende scherm getoond; selecteer hier de juiste branch:



Gebruik de default waardes en selecteer 'Push'. Het volgende log scherm verschijnt. De push is succesvol uitgevoerd.



In de remote repository is de commit zichtbaar onder 'Commits':



En de commit:

Auteur	SHA1	Bericht
Marco Lemmers	4599261ec0	Toevoegen twee test bestanden

GIT Bash

Gebruik de volgende syntax om een branch te pushen naar de remote repository.

```
$ git push origin <branch name>
```

GIT Bash

In de command line push je als volgt:

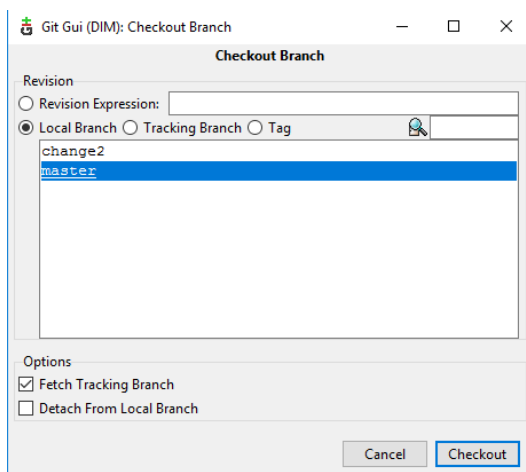
```
$ GIT push origin
```

6.9 ...hoe te mergen?

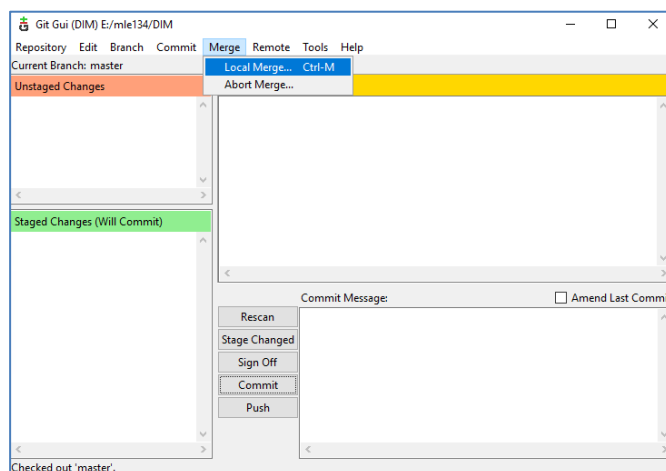
Een merge met de master branch wordt uitgevoerd wanneer de software een productie rijpe situatie heeft bereikt.

GIT Gui

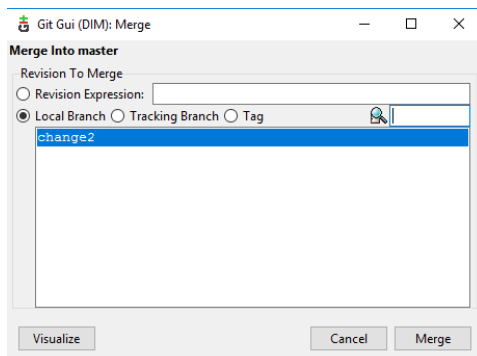
Wanneer een branch gemerged moet worden met de master branch, stap dan eerst over naar de master branch. Selecteer hiervoor in menu 'Branch' de optie 'Checkout'. Selecteer hier de master branch:



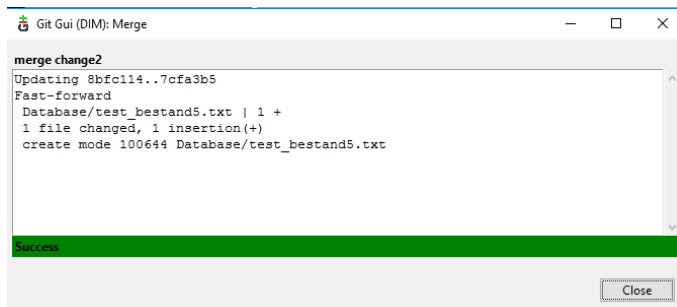
Selecteer vervolgens in menu 'Merge' de optie 'Local Merge':



Selecteer hier de te mergen branch, in dit geval branch 'change2', en klik op 'Merge':



Het resultaat:



In de commit history zie je dat de 'master' branch en de 'change2' branch nu naar hetzelfde commit punt verwijzen:



GIT Bash

Wanneer een lokale branch samengevoegd moet worden met de lokale 'master' (een merge), voer dan eerst een checkout uit naar de 'master' branch en voer dan een merge uit:

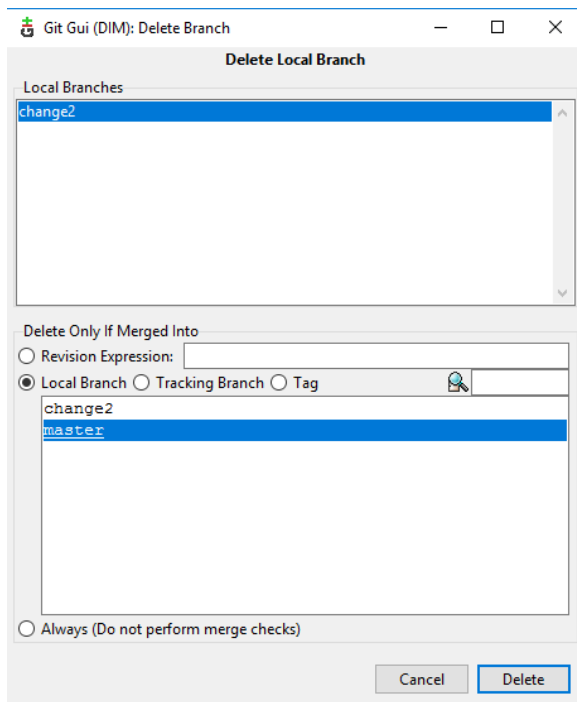
```
$ git checkout master  
$ git merge <branch name>
```

6.10 ...verwijderen van een branch?

Wanneer een branch is gemerged met de 'master' branch, of als de branch niet meer van toepassing is, verwijder deze dan uit de lokale repository:

GIT Gui

Selecteer voor het verwijderen van een lokale branch de optie 'Delete' in menu 'Branch'. Zorg er daarbij eerst voor dat je niet in de betreffende branch zit (als dat wel het geval is doe dan een checkout naar de 'master' branch):



Kies in het bovenste venster de te verwijderen branch. Geef vervolgens op wat de conditie is waaronder de merge mag plaatsvinden. In ons geval zal een branch alleen verwijderd mogen worden als deze lokaal gemerged in met de master branch.

Klik vervolgens op de 'Delete' knop. De branch is nu verwijderd.

GIT Bash

Verwijder een branch als volgt:

```
git branch -d <branch name>
```

The -d optie zorgt ervoor dat de branch alleen wordt verwijderd als alle commits op de branch zijn gepushed en gemerged met de remote branch.

```
git branch -D <branch name>
```

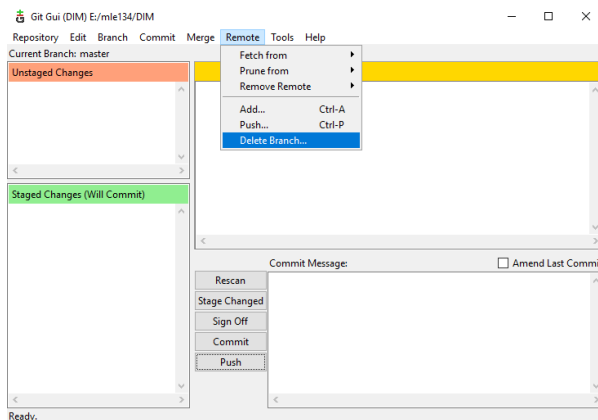
Met de -D optie wordt de branch verwijderd, ook al is deze nog niet gepushed en gemerged.

6.11 ...verwijderen van een remote branch?

Wanneer de lokale branch is verwijderd, kan ook de remote branch verwijderd worden. Coördineer echter het gebruik van branches; als iemand deze branch echter nog lokaal heeft staan, en deze pusht, dan zal de branch weer terugkeren in de remote repository !

GIT Gui

Kies in menu 'Remote' voor optie 'Delete Branch':



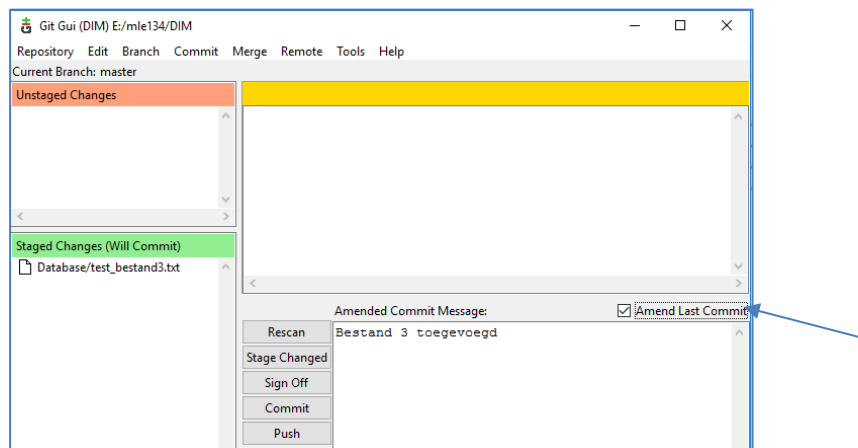
Selecteer hierna de te deleten branch en de conditie waaronder deze verwijderd mag worden:

6.12 ...wijzigen van commits die nog niet zijn gepushed?

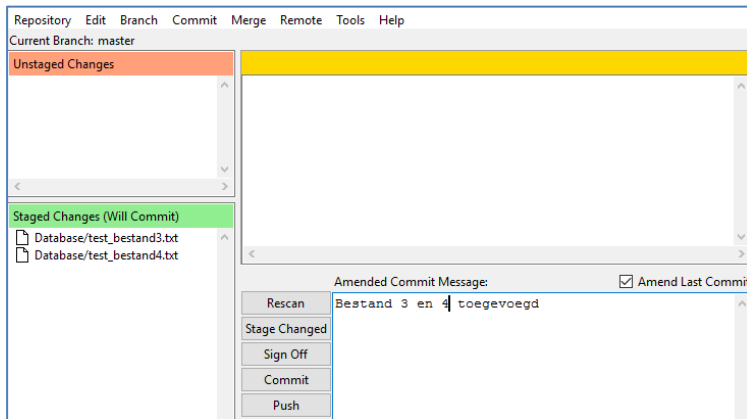
Stel dat er net een commit is gedaan voor bestand 'test_bestand3', maar deze blijkt incompleet en je wilt dat de laatste commit ook deze aanpassingen bevat. Dan kun je een 'amend' uitvoeren.

GIT Gui

Met gebruik van checkbox 'Amend last commit' verschijnt de laatste commit weer in bij de 'Staged Changes' en zie je ook de laatste commit boodschap:



Nu kunnen extra wijzigingen gedaan worden (bv de creatie van een nieuw bestand) en de commit boodschap worden aangepast.



Wanneer nu een commit wordt uitgevoerd, ziet de commit historie er als volgt uit:



Voer de gewenste file wijzigingen uit en commit als volgt:

```
$ git commit --amend
```

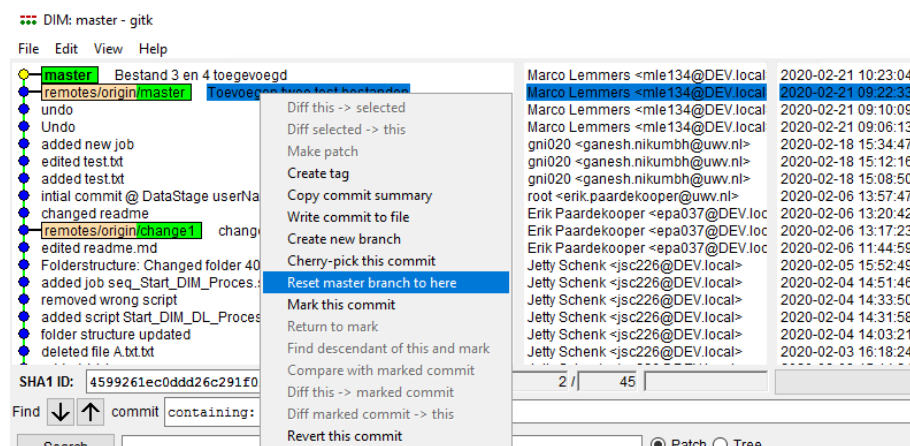
(Hier kan ook de -m optie gebruikt worden om de commit-boodschap aan te passen).

6.13 ...terugkeren naar een vorige commit (ongedaan maken van wijzigingen)

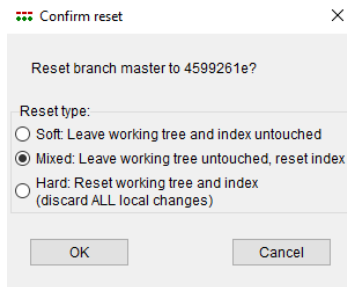
In extreme situaties kan het voorkomen dat je terug wilt keren naar een vorig commit punt en alle wijzigingen die daarna zijn uitgevoerd ongedaan wilt maken.

GIT Gui

In het overzicht van de commit historie is het mogelijk om terug te keren naar een vorig commit punt. Klik daarvoor op het gewenste commit boodschap en selecteer de optie 'Reset <branchnaam> branch to here':



Hierna krijg je de keuze uit drie opties:



- **soft**
Index- en werkomgeving blijven ongewijzigd. Alle gewijzigde bestanden blijven in status 'Changes to be committed', zoals te zien met 'GIT status' (1).
- **mixed**
Reset de index, maar niet de werk omgeving (gewijzigde files blijven bewaard, maar verdwijnen uit de index). Dit is de default optie.
- **hard**
Synchroniseert de werkomgeving en de index met het commit punt waarheen gereset wordt.

GIT Bash

Voer om terug te keren naar een vorig commit-punt de volgende actie uit:

- Zoek de naam van de GIT commit waarnaar je terug wilt keren (hetzij gebruik makend van de SHA-1 key van de commit, of middels HEAD^, HEAD~3, etc.):

```
$ git reset 0f4b0427731755476ca9b18ea337efae531e151
```

Geeft hierbij de optie --soft of --hard mee als je de default waarde (mixed) niet wilt gebruiken.

6.14 ...ongedaan maken / rol back van wijzigingen die zijn gepushed?

Wijzig nooit (!) de centrale wijzigings-historie in de remote repository omdat anderen deze waarschijnlijk al gebruiken voor nieuwe ontwikkelingen. Als anderen hun wijzigingen dan weer committen en pushen in de centrale repository, dan komen de eerder uitgevoerde verwijderingen weer terug.

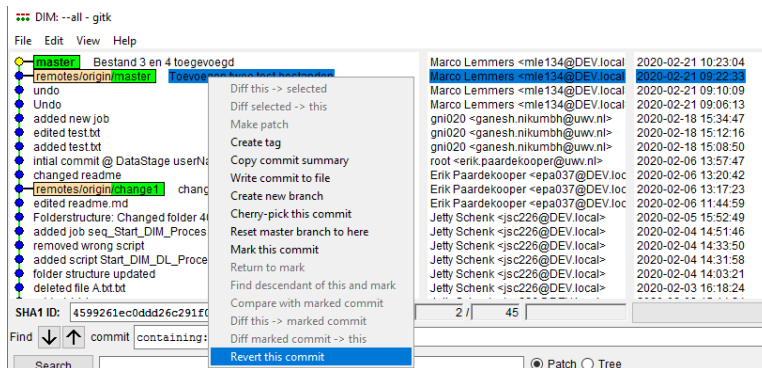
Dus: "eerst denken, dan pushen"!

6.15 ...ongedaan maken van een commit?

Met 'git revert' worden de wijzigingen die met een commit zijn gedaan, ongedaan gemaakt en wordt deze aanpassing ge-commit. Het is mogelijk om een willekeurige commit ongedaan te maken, maar dit kan resulteren in versie conflicten die opgelost moeten worden. Wees hier dus zeer voorzichtig mee!

GIT Gui

Open de commit history en klik op de commit (op de boodschap) die ongedaan moet worden gemaakt. Selecteer daarvoor de optie 'Revert this commit'.



Deze 'revert' resulteert in een nieuwe commit:



GIT Bash

Voer in de command line het volgende commando uit:

```
$ git revert a58bd5032df63af46107b700e4f6c2eb83b517d8
Finished one revert.
Created commit 8fecae8: Revert "edit robin"
1 files changed, 0 insertions(+), 3 deletions(-)
```

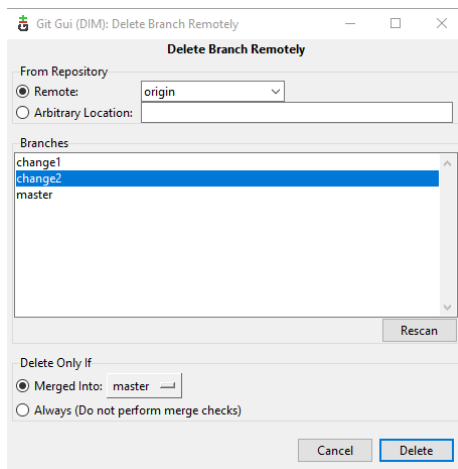
6.16 ...hoe merge conflicten op te lossen?

Het oplossen van merge conflicten is een handmatige actie, waarbij handmatig bestanden moeten worden aangepast en de index wordt gewijzigd met 'GIT add' en 'GIT rm'. Als het conflict is opgelost gebruik dan 'GIT commit'.

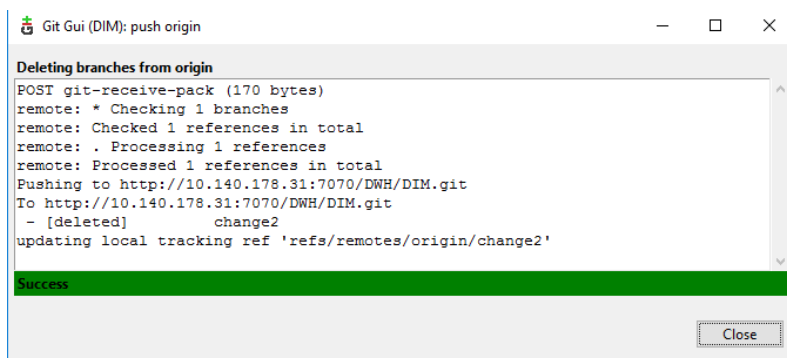
Met '**GIT mergetool**' wordt de default keuze voor een 'diff' tool gestart, waarmee verschillen tussen bestanden kunnen worden vergeleken en opgelost. Wanneer de resultaten hiervan worden gesaved, worden alle tijdelijke bestanden die zijn gegenereerd weer verwijderd.

GIT geeft normaal gesproken aan wanneer er een merge conflict dreigt op te treden.

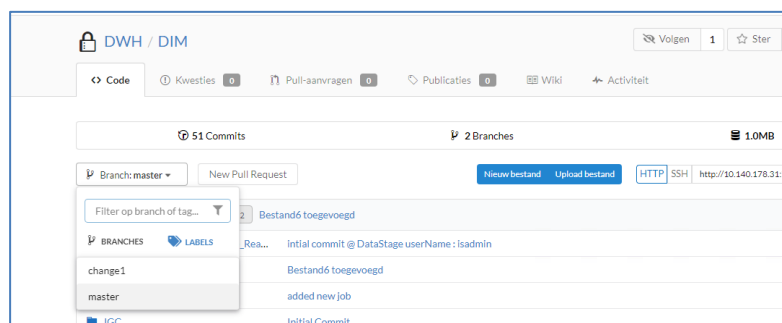
NB. In het geval van DataStage componenten, praten we over binaire objecten, en is het handmatig wijzigen van bestanden geen optie.



En klik hier 'Delete'.



In de remote repository is branch2 niet langer aanwezig:



GIT Bash

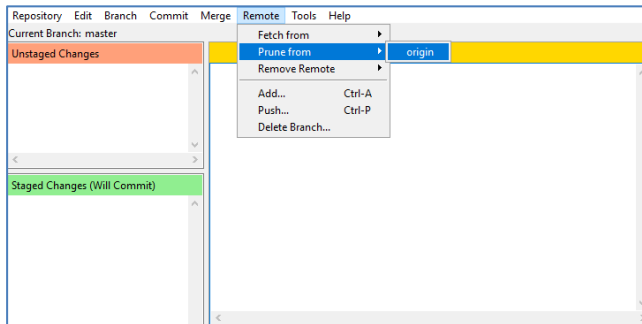
Verwijder een remote branch als volgt:.

```
git push origin --delete <branch name>
```

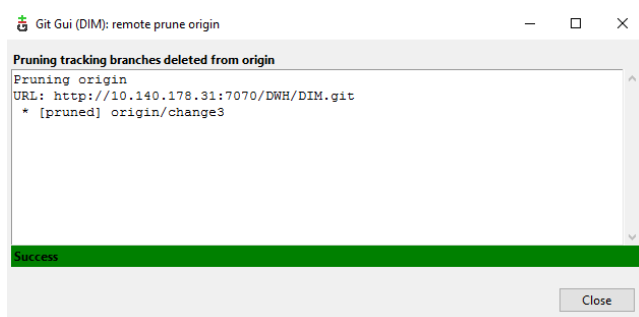
6.17 ...verwijderen van lokale branches die niet meer bestaan in de remote repository?

GIT Gui

De GUI ondersteunt alleen het maken van een overzicht van lokale branches die die niet meer in de remote voorkomen. Selecteer onder menu item 'Remote' de optie 'Prune from' -> 'origin'.



En het resultaat na uitvoering:



GIT Bash

In de command line is het mogelijk om de lokale branches die niet meer voorkomen in de remote ook werkelijk te verwijderen:

```
git remote prune origin
```

Wanneer je de optie `--dry-run` toevoegt, krijg je het overzicht van lokale branches die lokaal niet meer bestaan.

7 Het gebruik van GIT voor DataStage objecten

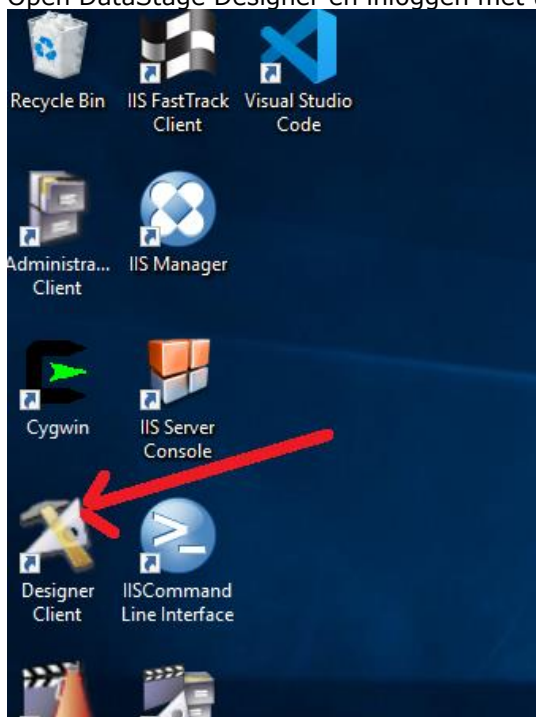
- Het GIT versiebeheer van DataStage objecten vindt plaats op basis van DataStage export (.dsx) bestanden. Deze bestanden worden in de GIT werkdirectory geplaatst (in de DataStage project mappenstructuur).
- Eerst voert de ontwikkelaar een pull uit vanuit de remote repository om de meest recente .dsx bestanden op te halen. Daarnaast maakt de ontwikkelaar een nieuwe branch aan voor de uit te voeren wijzigingen.
- Vervolgens importeert de ontwikkelaar de .dsx bestanden in de DataStage omgeving en worden de wijzigingen in DataStage uitgevoerd.
- Wanneer de wijzigingen gereed zijn en de unit test is afgerond, worden de gewijzigde DataStage componenten geëxporteerd en opgeslagen in de lokale GIT omgeving.

Onderstaande secties beschrijven het DataStage export- en import proces middels de DataStage Designer client.

7.1 Exporteren van DataStage objecten (componenten) uit DataStage Designer

In de standaard manier van werken worden alle DataStage objecten geëxporteerd, inclusief "executables" uit DataStage Designer.

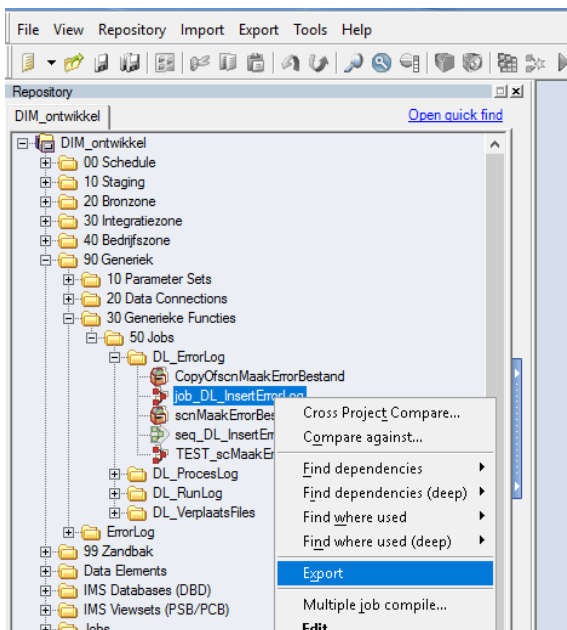
- Open DataStage Designer en inloggen met uw gegevens



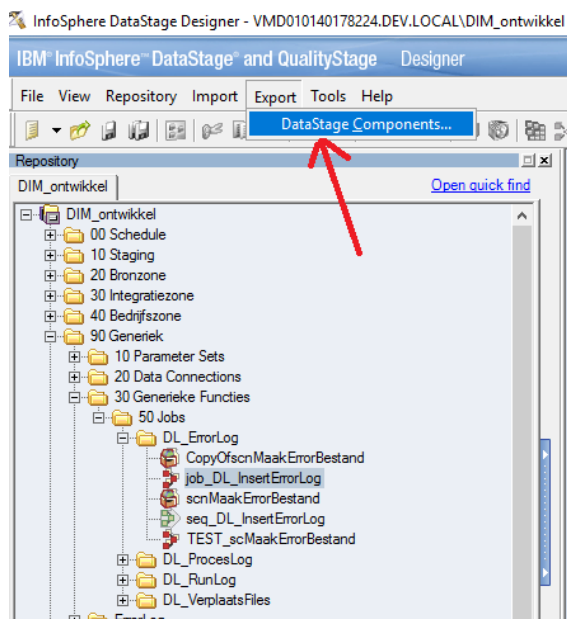


The image shows a screenshot of the 'Attach to Project' dialog box in IBM InfoSphere DataStage and QualityStage Version 11.7. The dialog box has a blue header with the IBM logo. Below the header, the text 'IBM® InfoSphere™ DataStage® and QualityStage Version 11.7' is displayed. The dialog contains several input fields: 'Host name of the services tier:' with a dropdown menu showing 'vmd010140178223.dev.local:9446'; 'User name:' with a text field containing 'gni020'; 'Password:' with an empty text field; and 'Project:' with a dropdown menu showing 'VMD010140178224.DEV.LOCAL/DIM_ontwikkel'. At the bottom, there are two buttons: 'Login' and 'Cancel'.

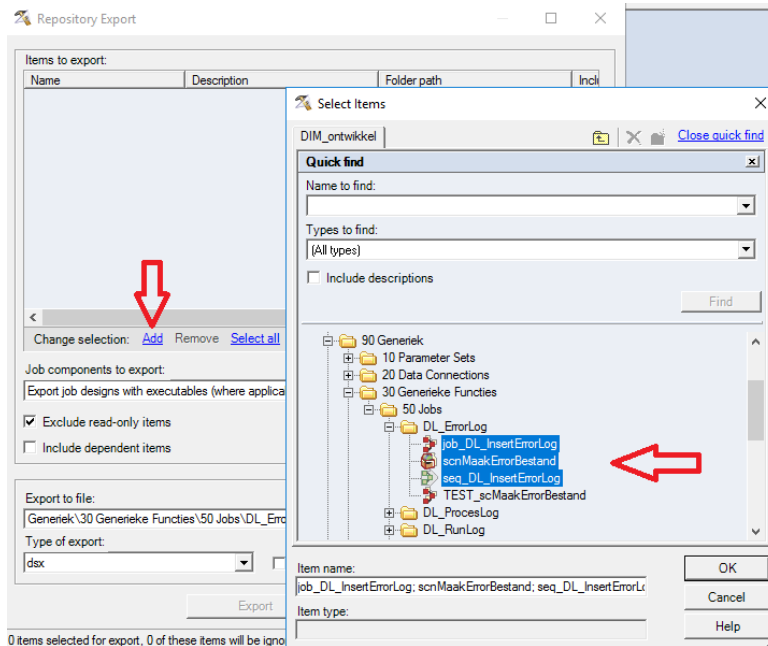
- selecteer het DataStage-object dat moet worden geëxporteerd. Er zijn 2 manieren om de objecten te exporteren:
 1. Selecteer een of meer objecten en klik met de rechtermuisknop op de geselecteerde objecten. Selecteer de optie "Export"



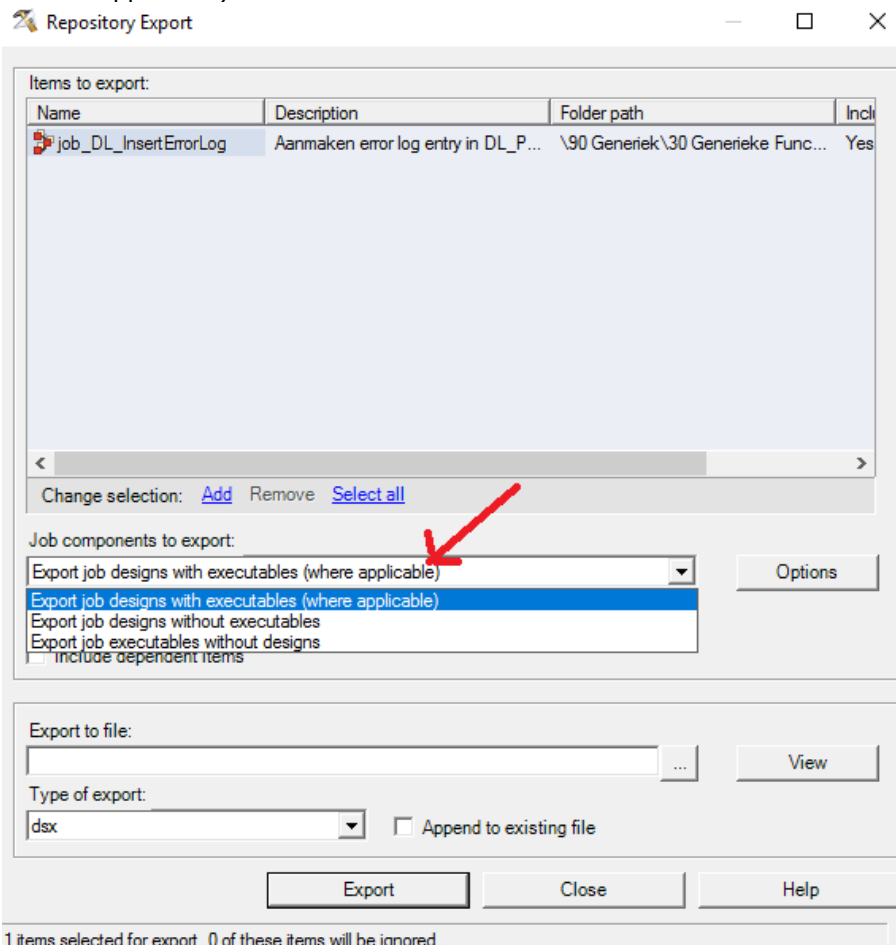
2. Klik in het menutabblad op "Export" en selecteer "DataStage Components"



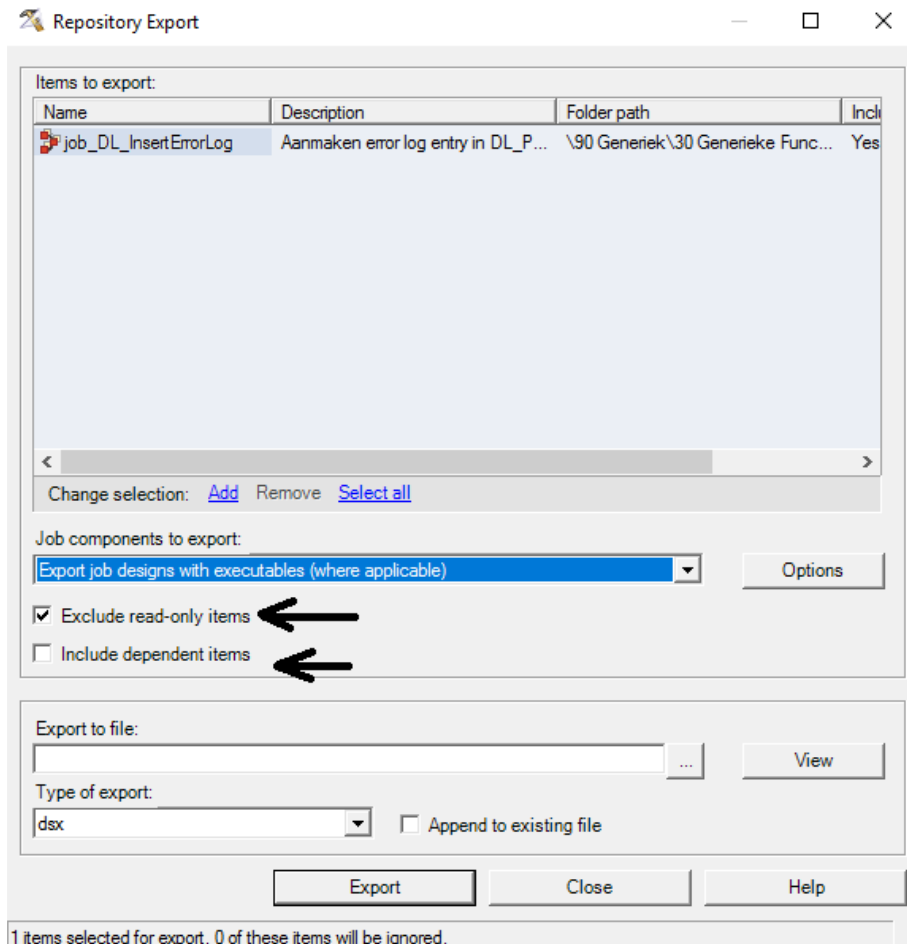
- Klik in het Exportvenster op de link "Add". Selecteer een of meer objecten en klik op "OK".



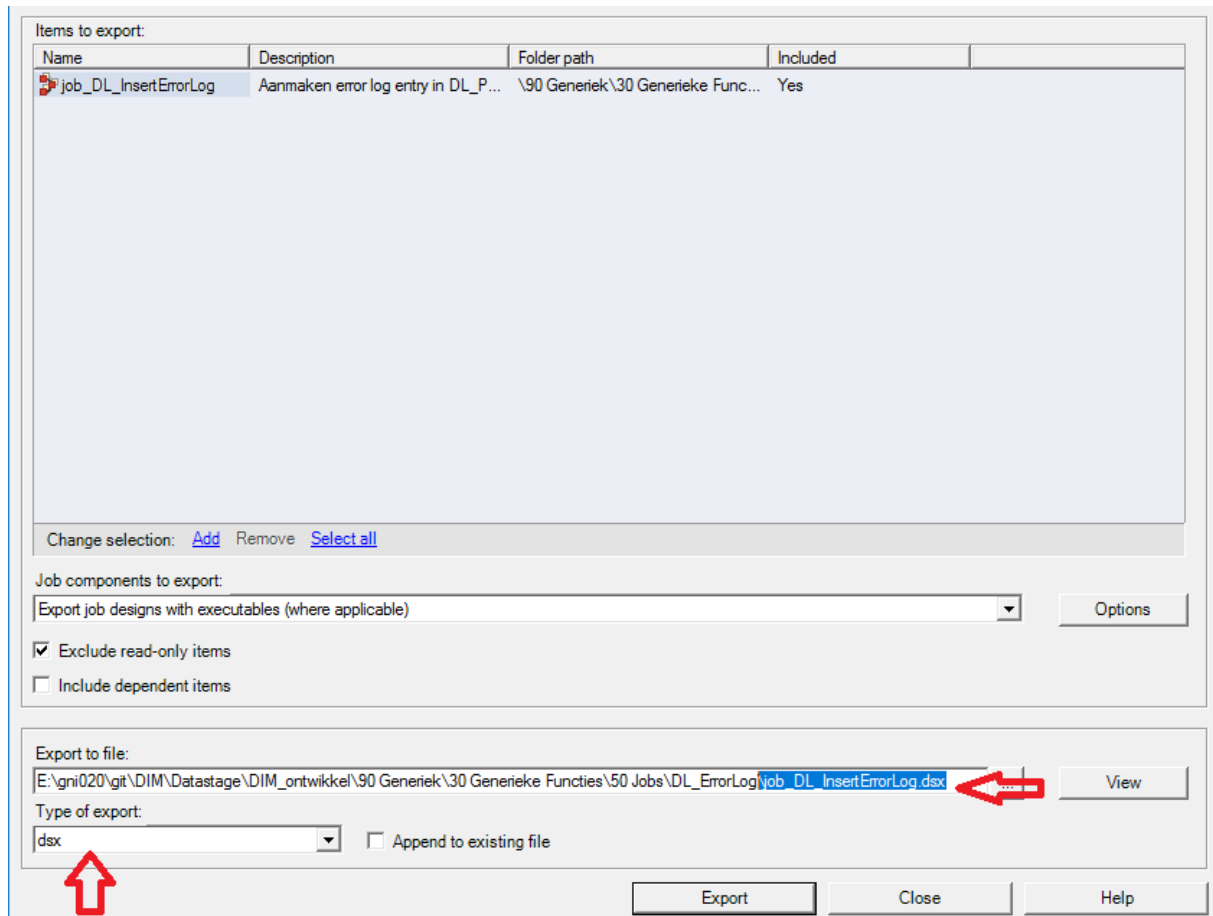
- Selecteer in het venster Exporteren de optie "Export Job designs with executables (where applicable)".



- Selecteer "Exclude read-only items" en deselecteer "Include dependant items".



- Geef het bestandspad en de bestandsnaam op in de prompt voor "Export to file" en selecteer **dsx** voor "Type of export".
 - Selecteer de lokale GIT-opslagmap waar het object moet worden opgeslagen.
 - Voor de export van een enkel object moet de bestandsnaam dezelfde naam hebben als de DataStage-objectnaam.
 - Als u meerdere objecten in één bestand exporteert, geef dan een betekenisvolle naam aan het bestand.
 - Gebruik het bestandstype ".dsx". Sla vervolgens het bestand op..



Name	Description	Folder path	Included
job_DL_InsertErrorLog	Aanmaken error log entry in DL_P...	\90 Generiek\30 Generieke Func...	Yes

Change selection: [Add](#) [Remove](#) [Select all](#)

Job components to export:
Export job designs with executables (where applicable) Options

☒ Exclude read-only items
☐ Include dependent items

Export to file:
E:\gni020\git\DIM\Datastage\DIM_ontwikkel\90 Generiek\30 Generieke Functies\50 Jobs\DL_ErrorLog\job_DL_InsertErrorLog.dsx View

Type of export:
dsx ☐ Append to existing file

Export Close Help

- Klik dan op de 'Export' knop.

7.2 dsxsplitter.py hulpprogramma voor het splitsen van multi object dsx bestand en creeren van een manifest bestand

Gezien de manier waarop individuele dsx objecten worden onderhouden en de Datastage-achtige mappenstructuur in GIT, wordt het al snel een vervelend proces voor een ontwikkelaar.

De dsxsplitter.py utility helpt bij het oplossen van onderstaande problemen.

- Het maken van de juiste bestandsnaam
- Verplaatsen van het bestand naar de juiste map / selecteren van de juiste map
- En dat voor elk onderdeel
- Aanmaken van een Manifest file met een overzicht van alle uitgepakte dsx bestanden

De ontwikkelaar kan alle Datastage objects in één groot dsx exportbestand exporteren, en kan het hulpprogramma dsxsplitter.py uitvoeren om het grote bestand op te splitsen in individuele Datastage-objecten.

Het hulpprogramma maakt ook het mappad aan volgens de Datastage mapstructuur in de GIT-directory, als die niet bestaat.

In de locatie van waaruit je je python .bat programma start (of in de folder waarin je het commando uitvoert in (cmd omgeving)) zal ook een manifest bestand gemaakt worden met de bestandsnaam naar keuze. (Let op dat er niet al eenzelfde manifest bestand staat met eenzelfde naam in deze folder. Deze zal worden overschreven.)

Het dsxsplitter.py programma staat momenteel in de V:\DWH\medewerkers\dim_utilities map op de OTAP windows server. De syntaxis voor het uitvoeren van het script is als volgt:

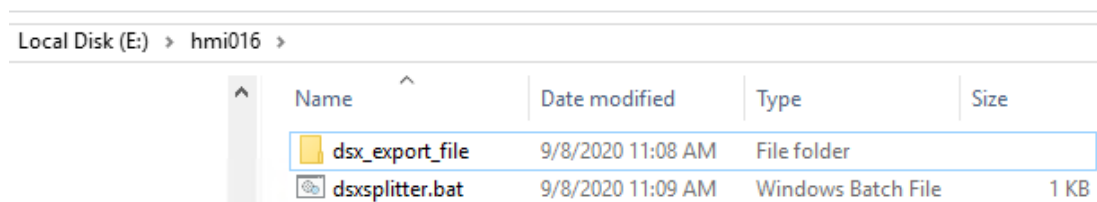
python <scripts path> /dsxsplitter.py <groot dsx-bestandspad> < directory waaronder de jobmappen en job dsx moeten worden aangemaakt > < Voer Y in als dsx uitvoerbare bestanden bevat, anders N >

e.g python V:\DWH\medewerkers\dim_utilities\dsxsplitter.py

V:\DWH\medewerkers\hmi017\Exports2

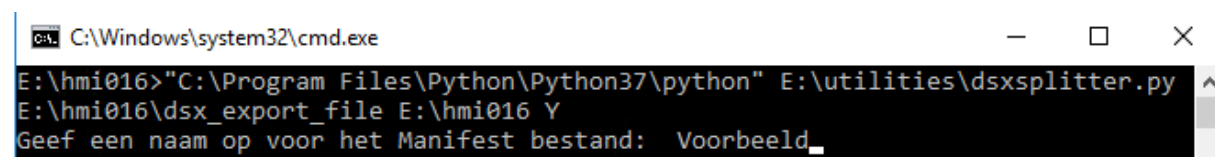
V:\DWH\medewerkers\hmi017\GIT\DIM\Datastage\DIM_ontwikkel Y

voordat het script wordt uitgevoerd :



Name	Date modified	Type	Size
dsx_export_file	9/8/2020 11:08 AM	File folder	
dsxsplitter.bat	9/8/2020 11:09 AM	Windows Batch File	1 KB

het script uitvoeren :



```
C:\Windows\system32\cmd.exe
E:\hmi016>"C:\Program Files\Python\Python37\python" E:\utilities\dsxsplitter.py
E:\hmi016\dsx_export_file E:\hmi016 Y
Geef een naam op voor het Manifest bestand: Voorbeeld_
```

Na opgeven naam voor manifest bestand:

```
manifest file Voorbeeld.manifest word aangemaakt in folder E:\hmi016
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Start verwerking export bestand :    parameters.dsx

bestand bevat Executables, deze zal eerst vergeleken worden of aantallen klopt

geen mismatch in DSJOB en Executables, programma loopt

Getting dsp block from match ordinals
Creating new file: par_DIM_Dbs.dsx in E:\hmi016\90 Generiek\10 Parameter Sets
Creating new file: par_DIM_Email_Admin.dsx in E:\hmi016\90 Generiek\10 Parameter Sets
Creating new file: par_Linux_Paden.dsx in E:\hmi016\90 Generiek\10 Parameter Sets
Creating new file: par_OracleParFile.dsx in E:\hmi016\90 Generiek\10 Parameter Sets
Creating new file: par_Teksten.dsx in E:\hmi016\90 Generiek\10 Parameter Sets

E:\hmi016>PAUSE
Press any key to continue . . .
```

na uitvoering van het script staat naast de uitgedaakte dsx folders er ook het manifest bestand:

Local Disk (E:) > hmi016 >				
Name	Date modified	Type	Size	
90 Generiek	9/8/2020 11:11 AM	File folder		
dsx_export_file	9/8/2020 11:08 AM	File folder		
dsxsplitter.bat	9/8/2020 11:09 AM	Windows Batch File	1 KB	
Voorbeeld.manifest	9/8/2020 11:11 AM	MANIFEST File	1 KB	

Ook kunnen de geëxporteerde dsx bestanden en mappen gekopieerd en geplakt moeten worden in de Datastage projectmap onder GIT indien je niet meteen naar de GIT branch folder verwijst.

Het manifest overzicht heeft de structuur zoals beschreven in 5.3:

	Voorbeeld.manifest
1	\90 Generiek\10 Parameter Sets\par_DIM_Dbs.dsx
2	\90 Generiek\10 Parameter Sets\par_DIM_Email_Admin.dsx
3	\90 Generiek\10 Parameter Sets\par_Linux_Paden.dsx
4	\90 Generiek\10 Parameter Sets\par_OracleParFile.dsx
5	\90 Generiek\10 Parameter Sets\par_Teksten.dsx

7.2.1 Creeer 'dsxsplitter.bat' bestand

Om makkelijk het python script te kunnen uitvoeren zou je gebruik kunnen maken van een 'dsxsplitter.bat' bestand. Maak in een eigen subfolder op de E-drive het bestand 'dsxsplitter.bat' aan, met de volgende inhoud:

```
python V:\DWH\medewerkers\dim_utilities\dsxsplitter.py V:\DWH\medewerkers\hmi017\Exports2  
V:\DWH\medewerkers\hmi017\GIT\DIM\Datastage\DIM_ontwikkel Y
```

PAUSE

In het **rood** vind je de export directory van waaruit de dsx file gelezen moeten worden. De tweede directory (in **blauw**) geeft aan waar de dsx files terecht moeten komen; Verwijs hier naar je GIT home directory waar je project staat.

De 'Y' geeft aan dat de dsx file ook executable code bevat. In theorie kun je ook dsx files zonder executabel code exporteren, voor de jobs die we naar GIT deployen word dit nooit gedaan. Gebruik altijd 'Y'.

Met de PAUSE wordt de DOS prompt nog even vastgehouden, en kun je de resultaten van het opsplitsen nog bekijken in het "cmd" commando scherm.

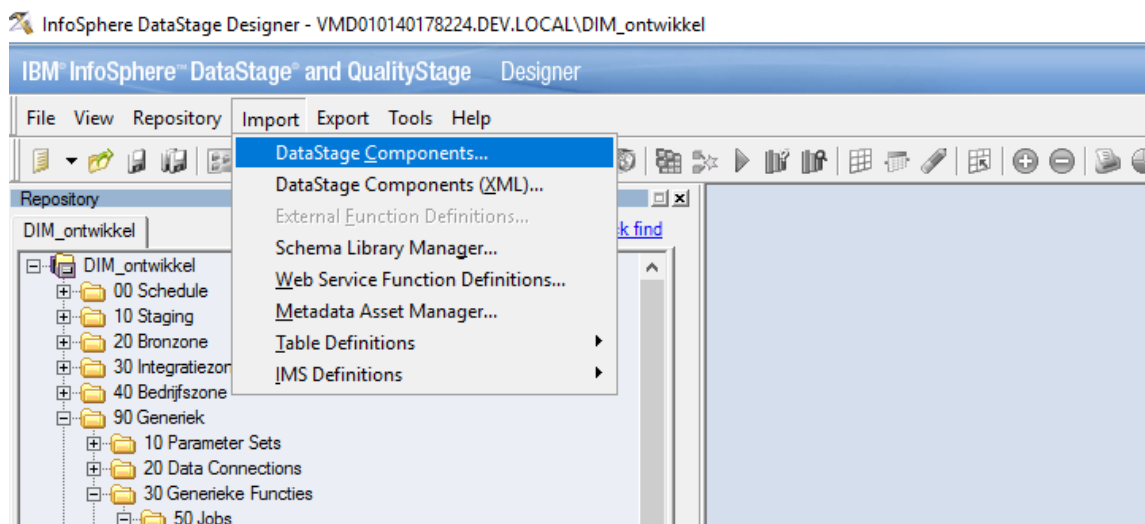
- Manifest bestand

De locatie waar je je 'dsxsplitter.bat' bestand uitvoert word ook het manifest bestand ook naar toegeschreven.

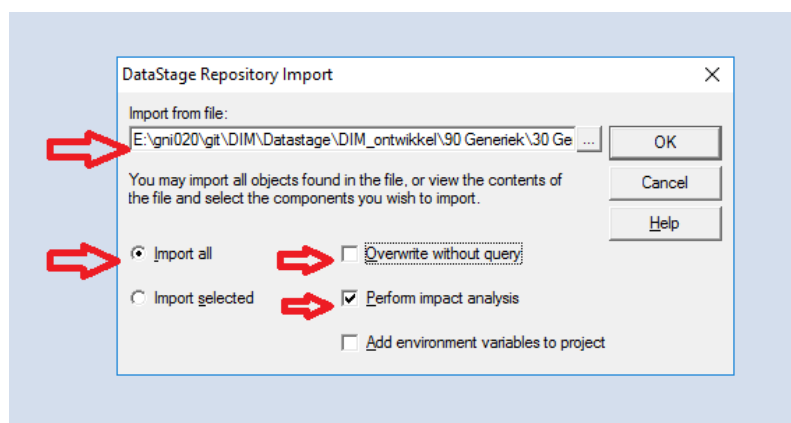
7.3 Importeren van DataStage objecten (componenten) met behulp van Datastage Designer

Volg de volgende stappen voor het importeren van DataStage objecten in Datastage Designer.

- Klik in het tabblad Datastage designer op Importeren en selecteer Datastage Componenten.



- Selecteer het te importeren bestand.
- Selecteer de optie "Import All".
- Deselecteer de optie "Overwrite without query".
- Selecteer de optie "Perform Impact Analysis".



- Klik dan op de 'OK' knop.

8 Scenario's

Dit hoofdstuk beschrijft het gebruik van Git Bash in de context van een aantal ontwikkel-scenario's.

8.1 Zet de componenten voor een nieuwe user story in Git

- In Git: doe een checkout naar de lokale ontwikkel branch
git checkout ontwikkel
- In Git: zorg ervoor dat je lokaal de laatste versie van de ontwikkel branch hebt
git pull
- In Git: creëer een nieuwe lokale branch voor de user story
git checkout -b DAFA-<user story nummer>
- In DataStage: exporteer de nieuwe/gewijzigde jobs in één export file
- OTAP Windows: gebruik de dsx splitter om de export file op te splitsen in files per datastage component en om de files naar de lokale Git werkomgeving te verplaatsen
- OTAP Windows: zet de nieuwe/gewijzigde database en/of linux scripts in de juiste locatie in de lokale Git werkomgeving.
- OTAP Windows: Verplaats het door de dsx splitter gemaakte manifest file DAFA-<user story nummer>.manifest naar de 'Deployment' / 'Test' folder van de lokale Git werkomgeving.
- In Git: voeg alle nieuwe/gewijzigde files toe aan de Git index
git add .
- In Git: check of de gewenste files in de index zijn opgenomen
git status
- In Git: commit de componenten
git commit -m "<commit boodschap>"
- In Git: push de componenten naar de remote
git push origin DAFA-<user story nummer>
- Stuur een mail naar Wouter/Jelmer en Ganesh/Tijmen met darin de melding dat de user story gereed is voor deployment naar TEST
- Neem de user story op in de DIM_Deployment_Tracker sheet in de sharepointfolder Knights/Components/Deployment

Op dit moment is de user story gereed voor deployment naar TEST.