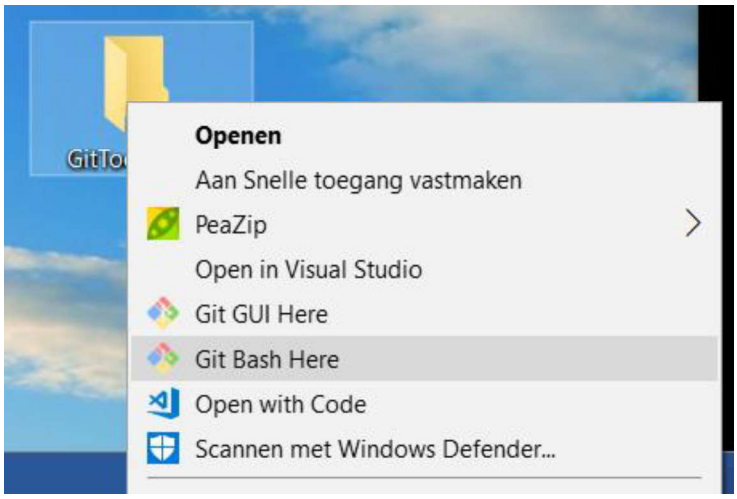
7 VAN START MET GIT

7.1 EEN EERSTE KENNISMAKING

Eenmaal Git geinstalleerd is, kan je hiervan gebruik maken vanuit de mappenstructuur binnen Windows. Je maakt een map aan en kunt hier dan de git functionaliteiten op toepassen, gaande van initialiseren tot bepaling welke files je wenst op te volgen tot het maken van een online repository, die als back-up kan dienen of de startfolder waarvan je dan later kunt gaan samenwerken voor eenzelfde project. Dit deel leert je de basis voor het aanmaken van dergelijke file. Programma's zoals Visual Studio en Visual Studio Code hebben dergelijke functionaliteit standaard ingebakken, waardoor ze achter de schermen eigenlijk dezelfde code gaan gebruiken.

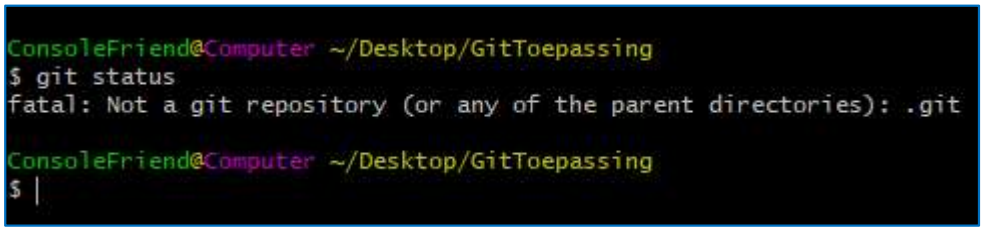
7.1.1 EEN GIT REPOSITORY AANMAKEN

- Maak een Mapje met de naam **GitToepassing** aan *(dit mag gerust op je bureaublad)*
- Rechtsklik op dit mapje en kies vervolgens **Git Bash Here**



- Je opent hierdoor een **Git Bash console** en bevind je meteen in de werkmap **GitToepassing** op de Desktop.

- Wanneer we even het commando `git status` ingeven kunnen we meteen zien dat we in deze directory nog **niet met git** aan de slag kunnen.



- We moeten eerst een **git repository** aanmaken. Dit doen we door het commando `git init` te gebruiken.

```
ConsoleFriend@Computer ~/Desktop/GitToepassing
$ git init
Initialized empty Git repository in C:/                    /GitToepassing/.git/

ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$
```

- Laten we even de inhoud bekijken van onze folder.
  We doen dit door middel van het commando `ls -a`
  We zien dat we een nieuwe *(verborgen)* folder erbij gekregen hebben genaamd **.git**

```
ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$ ls -a
./  ../  .git/

ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$
```

  ! Let op het commando wat we net gebruikten is **geen git commando** maar een **bash** commando!
  Dit kan je zelf al afleiden omdat dit commando **niet met het woordje git** begint!

7.1.2 BESTANDEN TOEVOEGEN AAN JE REPOSITORY (STAGEN)

- We voegen een aantal zaken toe aan onze **GitToepassing** folder. We voorzien een tekstbestand genaamd **test01.txt** en een folder **testFolder**. In de folder **testFolder** maken we een tweede tekstbestand aan genaamd **test02.txt**.
  Het aanmaken van deze bestanden kan je doen door middel van onderstaand commando
  `touch test01.txt`
  Het is uiteraard ook prima als je deze bestanden aanmaakt door middel van de rechtermuisknop en de keuze te maken voor **Nieuw -> tekstbestand**

- We controleren even of we nu al iets in onze git zitten hebben door het commando **git status** in te geven.

```
ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        test01.txt
        testFolder/

nothing added to commit but untracked files present (use "git add" to track)

ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$
```

- We zien onze bestanden staan, echter wordt er op dit ogenblik nog niks 'bijgehouden' door git. Hiervoor dienen we eerst onze bestanden te gaan '**stagen'**. Dit doen we door volgende commando's in te geven:
  `git add test01.txt`
  `git add testFolder/`

```
ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$ git add test01.txt

ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$ git add testFolder/

ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$
```

- We controleren even met het commando **git status** hoe het er nu uitziet. We zien onze bestanden nu staan onder '**Changes to be comitted**' en dat is prima.

```
ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   test01.txt
        new file:   testFolder/test02.txt


ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$
```

7.1.3 WIJZIGINGEN COMMITTEN

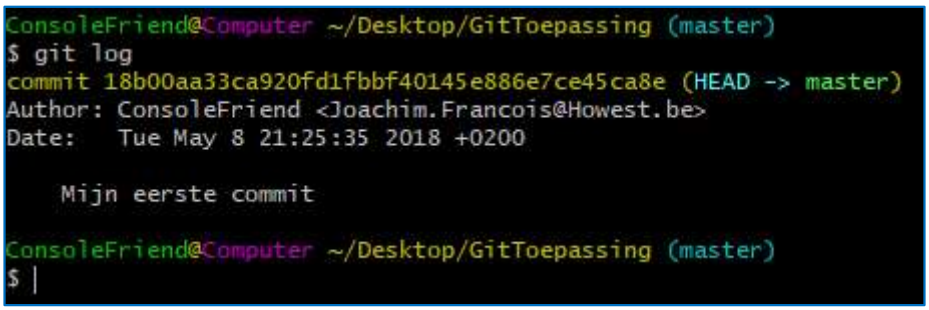
- Nu onze bestanden gestaged zijn kunnen we ze gaan commiten. Hiervoor maken we gebruik van het commando `git commit -m "Mijn eerste commit"`

```
ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$ git commit -m "Mijn eerste commit"
[master (root-commit) 18b00aa] Mijn eerste commit
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test01.txt
 create mode 100644 testFolder/test02.txt

ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$
```

- Als we nu terug gaan kijken naar de huidige status via `git status`, zien we dat alles gecommit is en er geen openstaande wijzigingen meer zijn.

```
ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$ git status
On branch master
nothing to commit, working tree clean

ConsoleFriend@Computer ~/Desktop/GitToepassing (master)
$
```

- Willen we even gaan kijken naar wat er nu precies gecommit werd dan kunnen we dit uiteraard ook via het command **`git log`**.



Tip! Eenmaal je meerdere commits hebt kan je deze in verkort formaat weergeven doormiddel van de parameter **`--oneline`**. Het volledige commando wordt vervolgens **`git log --oneline`**.

7.1.4 EEN BESTAND GESTAGED DAT JE NIET WOU STAGEN?

Het kan al eens gebeuren dat je een bestand gestaged hebt via het **`git add`** command maar dat je dit niet meteen mee wou nemen naar de staging. Hoe krijg je deze er dan terug uit?

Het commando **`git reset`** kan je hierbij helpen. Let echter wel op: het gebruik van git reset zorgt ervoor dat alle gestagede bestanden opnieuw 'unstaged' zullen worden.

Heb je nood om enkel 1 van meerdere bestanden te unstagen?
Dan kan je aan de slag met **`git reset -- naamVanBestand`**

Bijvoorbeeld: **`git reset -- verkeerd.txt`**

Wees wel steeds aandachtig als je met het git reset commando aan de slag gaat. Dit commando wordt namelijk ook gebruikt *(in andere vorm)* om commits ongedaan te maken. Hier komen we later nog op terug.

7.2 HOE WERKT GIT NU?

Alles in Git krijgt een controlegetal ('checksum') voordat het wordt opgeslagen en er wordt later met dat controlegetal ernaar gerefereerd. Dat betekent dat het onmogelijk is om de inhoud van een bestand of map te veranderen zonder dat Git er weet van heeft. Deze functionaliteit is in het diepste deel van Git ingebouwd en staat centraal in zijn filosofie. Je kunt geen informatie kwijtraken als het wordt verstuurd en bestanden kunnen niet corrupt raken zonder dat Git het kan opmerken.

Het mechanisme dat Git gebruikt voor deze controlegetallen heet een **SHA-1**-hash. Dat is een tekenreeks van 40 karakters lang, bestaande uit hexadecimale tekens (0–9 en a–f) en wordt berekend uit de inhoud van een bestand of directory-structuur in Git. Een SHA-1-hash ziet er ongeveer zo uit:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

Je zult deze hashwaarden overal tegenkomen omdat Git er zoveel gebruik van maakt. Sterker nog, Git bewaart alles niet onder een bestandsnaam maar in de database van Git met de hash van de inhoud als sleutel. De 40 gebruikte karakters zijn (veel!) meer dan voldoende om ervoor te zorgen dat elk onderdeel van je repository een unieke hashcode krijgt. Sterker nog: doorgaans volstaan hoogstens 8 tot 10 karakters. Git laat dan ook toe om slechts een prefix te gebruiken van eender welke hashcode, zolang die nog uniek is binnen de repository. Je zal dus vaak kortere codes tegenkomen zoals bv.

```
24b9da65
```

Zelfs in gigantische repositories met honderdduizenden lijnen code volstaan hooguit de 12-tal eerste karakters om elke hashcode uniek te beschrijven. We kunnen er dus wel veilig van uitgaan dat we nooit in de problemen zullen komen met de totale 40 karakters, die een astronomisch groot aantal mogelijkheden bieden. In de praktijk gebruik je slechts zoveel karakters als nodig, dit kan mee groeien naarmate de repository uibreidt. Git zal je vriendelijk laten weten wanneer je onvoldoende karakters gebruikt, d.w.z. wanneer er meerdere hashcodes in de repoistory voorkomen met dezelfde opgegeven prefix.

Bijna alles wat je in Git doet, leidt tot toevoeging van data in de Git database. Deze database bevindt zich in de folder .git. Het is erg moeilijk om het systeem iets te laten doen wat je niet ongedaan kan maken of het de gegevens te laten wissen op wat voor manier dan ook. Zoals met elke VCS kun je veranderingen verliezen of verhaspelen als je deze nog niet hebt gecommit; maar als je dat eenmaal hebt gedaan, is het erg moeilijk om die data te verliezen, zeker als je de lokale repository regelmatig uploadt ('push') naar een online repository. Dit maakt het gebruik van Git net zo tof, omdat je weet dat je kunt experimenteren zonder het gevaar te lopen jezelf behoorlijk in de nesten te werken.

7.3 DE DRIE TOESTANDEN IN GIT

Dit is het belangrijkste dat je over Git moet weten als je wilt dat de rest van het leerproces van Git vlotjes verloopt. Git heeft drie hoofdtoestanden waarin bestanden zich kunnen bevinden: gecommit (‘**commited’**), aangepast (‘**modified’**) en voorbereid voor een commit (‘**staged’**). **Committed** houdt in dat alle data veilig opgeslagen is in je lokale database. **Modified** betekent dat je het bestand hebt gewijzigd maar dat je nog niet naar je database gecommit hebt. **Staged** betekent dat je al hebt aangegeven dat de huidige versie van het aangepaste bestand in je volgende commit meegenomen moet worden.

Dit brengt ons tot de drie hoofdonderdelen van een Gitproject: de **Git directory**, de werk-directory (‘**working directory’**), en de wachtrij voor een commit (‘**staging area’**).

7.3.1 WORKING DIRECTORY

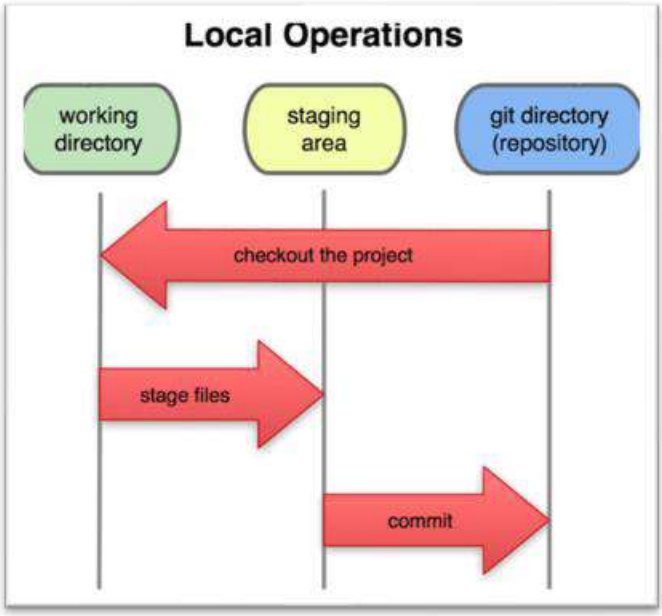
De werk directory is een checkout van een bepaalde versie van het project. Deze bestanden worden uit de gecomprimeerde database in de Git directory gehaald en op de harde schijf geplaatst waar jij ze kunt gebruiken of bewerken.

7.3.2 STAGING AREA

De wachtrij is een simpel bestand, dat zich normaalgesproken in je Git directory bevindt, waar informatie opgeslagen wordt over wat in de volgende commit meegaat. Het wordt soms de index genoemd, maar tegenwoordig wordt het de staging area genoemd.

7.3.3 GIT DIRECTORY (REPOSITORY)

De Git directory is waar Git de metadata en objectdatabase van je project opslaat. Dit is het belangrijkste deel van Git. Deze directory wordt gekopiëerd wanneer je een repository kloont vanaf een andere computer of online repository.



7.4 ALGEMENE WORKFLOW MET GIT

De algemene workflow met git gaat ongeveer zo:

1. Je bewerkt bestanden in je werk directory.
2. Je bereidt de bestanden voor (staged), waardoor momentopnames (snapshots) worden toegevoegd aan de staging area.
3. Je maakt een commit. Een commit neemt alle snapshots van de staging area en bewaart die voorgoed in je git directory.

Als een bepaalde versie van een bestand in de git directory staat, wordt het beschouwd als gecommit. Als het is aangepast, maar wel aan de staging area is toegevoegd, is het staged. En als het veranderd is sinds het was uitgechecked maar niet staged is, is het aangepast.