

## Versionsverwaltung am Beispiel von Git

### Einleitung

**Git** ist ein Versionsverwaltungstool. Es dient als Aufbewahrungsort (Repository) für Programmcode, der zusätzlich alte Versionen vorhält. Es werden alle Änderungen von Dateien die sogenannten **commits** in einem Logfile aufgezeichnet, sodass man zu jedem Commit-Zeitpunkt die entsprechende Dateiversion wiederherstellen kann.

Warum sollte man ein Versionskontrollsystem benutzen?

1. Es dient als Backup im Fall von Codeänderungen oder Speicherausfällen.
2. Es fungiert als Archiv aller Versionssprünge. Somit ist eine Rückkehr zu einer spezifischen Version möglich.
3. Es erleichtert die Zusammenarbeit zwischen Teammitgliedern und dient zugleich als Projektmanagementtool.
4. etc. ...

### Grundlagen

#### Anlegen eines Repositorys

Als Grundlage eines Git-Repositorys dient ein beliebiges Verzeichnis. In diesem Beispiel wird das bereits existierende Verzeichnis **hello-git** mit Datei **Main.java** zu einem Git-Repository.

Hierzu wechselt man lediglich in das entsprechende und führt den Befehl **git init** aus.

```
1 cd [path to]\hello-git
2 git init
```

Git quittiert den Erfolg mit der Meldung **Initialized empty Git repository in [path to]\hello-git\.git** und erstellt einen versteckten Ordner **.git**. Dort werden alle benötigten Daten für die Versionsverwaltung bereitgestellt.

```
27.01.2022 07:52      130 config
27.01.2022 07:52       73 description
27.01.2022 07:52      23 HEAD
27.01.2022 07:52    <DIR> hooks
27.01.2022 07:52    <DIR> info
27.01.2022 07:52    <DIR> objects
27.01.2022 07:52    <DIR> refs
                3 Datei(en),      226 Bytes
                4 Verzeichnis(se), 21.104.840.704 Bytes frei
```

## Übung

1. Installieren Sie Git auf Ihrem Rechner. <https://git-scm.com/download/win>
2. Überprüfen Sie Ihre Installation in dem Sie den Befehl `git --version` ausführen.
3. Legen Sie einen Ordner an der als `WorkingDirectory` dienen soll und weisen Sie git an diesen Ordner ab sofort zu überwachen.

## Notiz:

### Status überprüfen

Mit dem Befehl `git status` lässt sich der Zustand aller Dateien in einem Repository über-prüfen. Für unser Beispiel ergibt sich also folgende Ausgabe:

```
On branch master

No commits yet

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

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

Offensichtlich befinden wir uns im Branch `master` (dazu später mehr) und die Datei `Hello.java` ist nicht unter der Versionskontrolle von Git. Der Befehl `status` liefert außerdem hilfreiche Tipps. Hier wird erklärt wie man die Datei `Hello.java` zur Versionskontrolle von Git hinzufügen (engl. tracked) kann.

### Dateien zur Versionskontrolle hinzufügen

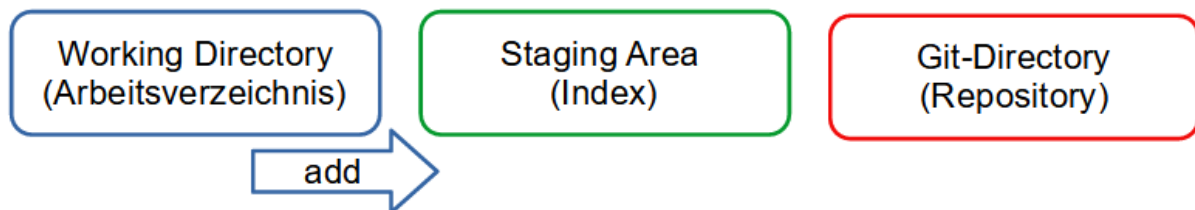
Wie der Befehl `status` schon vorschlägt wird als nächstes die Datei `Hello.java` in die `Staging Area` aufgenommen, also der Versionskontrolle hinzugefügt. Hierzu dient der Befehl `git add`.

```
P:\workspaces\git\hello-git>git add Main.java

P:\workspaces\git\hello-git>git status
On branch master

No commits yet

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



Der Status der Datei hat sich von `untracked` auf `added` bzw. `staged` geändert. Der Befehl `add` wird jedoch nicht nur dazu verwendet um *neue* Dateien hinzuzufügen, vielmehr können auch bestehende also schon von Git *getrackte* Dateien auf einen `commit` vorbereitet werden. Was ein `commit` ist klären wir gleich. Mit Verweis auf obige Grafik bedeutet ein `add` von Dateien, dass sie aus dem `Working Directory` in der `Staging Area` registriert werden.

## Übung

1. Erstellen Sie in dem Ordner der von *Git* überwacht wird zwei beliebige Dateien.
2. Fügen Sie beide Dateien zu `Staging Area` hinzu.
3. Überprüfen Sie den Zustand Ihres *Repositorys*

## Notiz:

## Git denkt in Änderungen (nicht in Dateien)

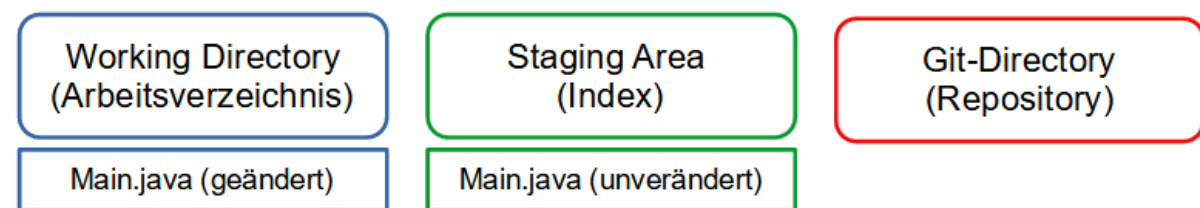
Wenn man eine Datei im **Staging**-Bereich registriert heißt das noch lange nicht, dass diese auch in das Repository aufgenommen (**committed**) wird. Zur Verdeutlichung verändern wir unsere Datei **Main.java** und lassen uns erneut den Status anzeigen.

```
P:\workspaces\git\hello-git>git status
On branch master
```

```
No commits yet
```

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

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
       modified:   Main.java
```



Nun wird die Datei **Main.java** zweimal aufgelistet. Das Hinzufügen der Datei wurde **gestaged** und ist somit bereit für einen **commit**. Die Änderung des Dateiinhalts hingegen ist noch nicht **gestaged**.

## Übung

1. Verändern Sie den Inhalt der beiden Dateien.
2. Überprüfen Sie erneut den Zustand Ihres *Repositorys*

## Notiz:

## Einen commit erzeugen

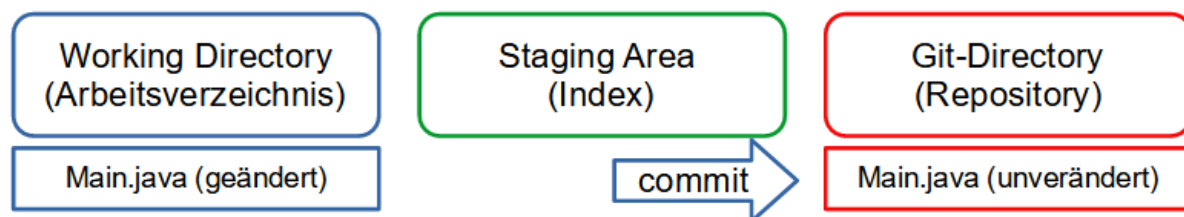
Wenn man im Bezug auf *Git* von **committen** spricht, meint man den aktuellen Projektstatus dauerhaft im *Git Repository* zu speichern. Hier werden jedoch nur die Dateien berücksichtigt die sich zum Zeitpunkt des **commits** in der *Staging Area* befinden.

Für unser Projekt liefert der erste Commit folgendes Ergebnis.

```
P:\workspaces\git\hello-git>git commit -m "First commit."
[master (root-commit) daff6e] First commit.
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 Main.java

P:\workspaces\git\hello-git>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Main.java

no changes added to commit (use "git add" and/or "git commit -a")
```



## Übung

1. Führen Sie Ihren ersten **commit** aus und übernehmen Sie Ihre Dateien aus der *Staging Area* ins *Repository*

## Notiz:

## Dateien entfernen

**Dateien aus der Staging Area entfernen** Um eine Datei aus der *Staging Area* zu entfernen ohne sie im Arbeitsverzeichnis zu löschen muss der Parameter `--cached` nach dem Befehl `rm` benutzt werden. Auf das bestehende Repository angewandt sieht das Ganze wie folgt aus.

```
P:\workspaces\git\hello-git>git rm --cached Main.java
rm 'Main.java'

P:\workspaces\git\hello-git>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    Main.java

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

Es wird bestätigt, dass die Datei `Main.java` aus der *Staging Area* entfernt wurde und demnach nicht mehr im Arbeitsverzeichnis *getrackt* wird. Beim nächsten *commit* würde die Datei demnach aus dem Repository entfernt werden.

**Datei im Working Directory und in der Staging Area löschen** Als nächstes versuchen wir die Datei `Main.java` sowohl aus der *Staging Area* als auch aus dem *Working Directory* zu löschen. Hierzu müssen wir sie natürlich zuerst wieder in die *Staging Area* aufnehmen. Benutzt man den Befehl `rm` ohne Parameter so wird versucht die Datei sowohl aus der *Staging Area* als auch aus dem *Working Directory* zu löschen. *Git* warnt in diesem Fall jedoch das die zu löschende Datei noch nicht *committed* wurde und somit Daten verloren gehen. Wenn man sich dessen bewusst ist benutzt man wie vorgeschlagen den Parameter `-f`.

```
P:\workspaces\git\hello-git>git add Main.java

P:\workspaces\git\hello-git>git rm Main.java
error: the following file has changes staged in the index:
        Main.java
(use --cached to keep the file, or -f to force removal)
```

Mit dem nachfolgenden *commit* wurde die Datei noch gleich aus dem Repository entfernt.

```
P:\workspaces\git\hello-git>git rm -f Main.java
rm 'Main.java'

P:\workspaces\git\hello-git>git commit -m "Forced Delete of Main.java"
[master 391357f] Forced Delete of Main.java
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 Main.java
```

## Übung

1. Nehmen Sie erneut Änderungen in Ihren Dateien vor.
2. Versuchen Sie die Dateien in den verschiedenen Bereichen von Git zu löschen.
3. Überprüfen Sie den Zustand von *Git* nach den jeweiligen Lösversuchen.

## Notiz:

### Die commit history

Nun könnte man davon ausgehen, dass die Datei `Hello.java` unwiderruflich aus dem *Repository* entfernt wurde. Das ist glücklicherweise nicht der Fall. Ein Blick in die *Commit-Historie* zeigt mehr. Sie kann mit dem Befehl `log` angezeigt werden.

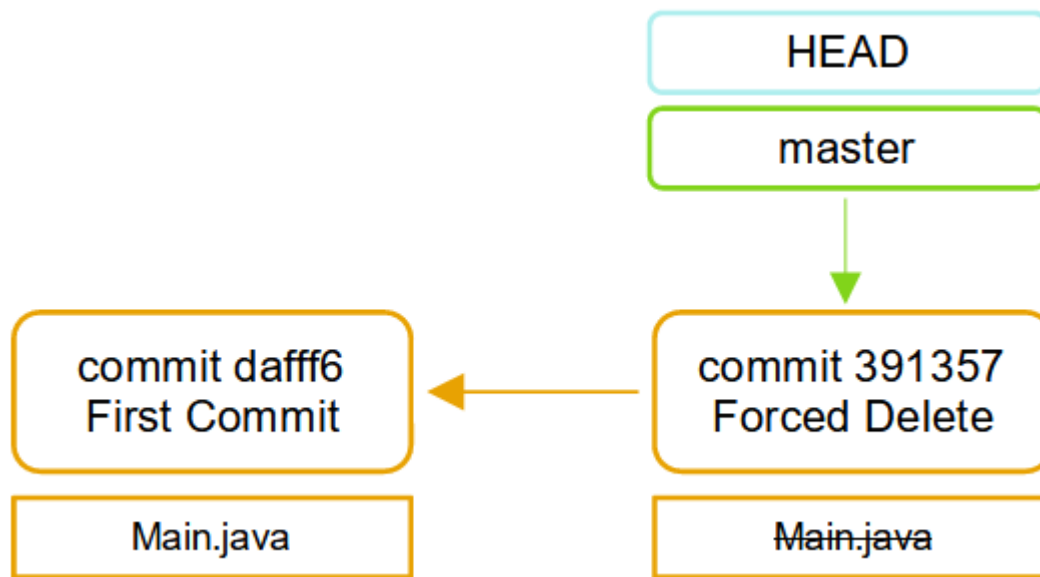
```
P:\workspaces\git\hello-git>git log
commit 391357fd6d00cc805569b9f0720ac89a8a12a82e (HEAD -> master)
Author: Mr. Go <go@bsinfo.eu>
Date:   Mon Jan 31 12:25:45 2022 +0100

    Forced Delete of Main.java

commit dafff6ebb02b862ba460cc9cdadd25dc553b158
Author: Mr. Go <go@bsinfo.eu>
Date:   Mon Jan 31 12:05:41 2022 +0100

    First commit.
```

Wie man sieht existieren zwei *commits*, die jeweils einen *Snapshot* des gesamten Projekts darstellen. Somit befindet sich die Datei `Main.java` noch im ersten 'commit'. Der erzeugte *Hashwert* wird von *Git* benutzt um die Informationen über die Dateien zu referenzieren. *Git* arbeitet nicht mit Dateinamen. Eine Grafik soll diesen Umstand verdeutlichen.



Der **HEAD** ist eine Referenz auf den aktuellen *Branch* (dazu gleich mehr). Aktuell haben wir nur einen *Branch* mit dem Bezeichner *master*.

### Formatierung der git history

Der Befehl `log` ermöglicht verschiedene Ausgaben, indem man ihn mit unterschiedlichen Parametern ausführt. Hierzu einige Beispiele:

`-pretty=oneline` (Ein `commit` pro Zeile)

```
P:\workspaces\git\hello-git>git log --pretty=oneline
391357fd6d00cc805569b9f0720ac89a8a12a82e (HEAD -> master) Forced Delete of Main.java
dafff6ebb02b862ba460cc9cdaddd25dc553b158 First commit.
```

`-pretty="%h - %an, %ar : %s"` (Benutzerdefiniert: %h: Hashwert(kurz), %an: Autor, %ar: Autor Datum (relativ), %s: commit subject)

```
P:\workspaces\git\hello-git>git log --pretty="%h - %an, %ar : %s"
391357f - Mr. Go, 38 minutes ago : Forced Delete of Main.java
dafff6e - Mr. Go, 58 minutes ago : First commit.
```

`-p -1` (Änderungen in jedem `commit` hier nur der letzte)



```
P:\workspaces\git\hello-git>git log -p -1
commit 391357fd6d00cc805569b9f0720ac89a8a12a82e (HEAD -> master)
Author: Mr. Go <go@bsinfo.eu>
Date:   Mon Jan 31 12:25:45 2022 +0100

    Forced Delete of Main.java

diff --git a/Main.java b/Main.java
deleted file mode 100644
index e69de29..0000000
```

-stat (Änderungen mit Statistik)

```
P:\workspaces\git\hello-git>git log --stat
commit 391357fd6d00cc805569b9f0720ac89a8a12a82e (HEAD -> master)
Author: Mr. Go <go@bsinfo.eu>
Date:   Mon Jan 31 12:25:45 2022 +0100

    Forced Delete of Main.java

Main.java | 0
1 file changed, 0 insertions(+), 0 deletions(-)

commit dafff6ebb02b862ba460cc9cdaddd25dc553b158
Author: Mr. Go <go@bsinfo.eu>
Date:   Mon Jan 31 12:05:41 2022 +0100

    First commit.

Main.java | 0
1 file changed, 0 insertions(+), 0 deletions(-)
```

## Übung

1. Recherchieren Sie auf <https://devhints.io/git-log-format> und Erstellen Sie sich ein Gitlog-Format nach Ihren Wünschen.

## Notiz:

## Dateien aus einem *Repository* wiederherstellen

Mit dem Befehl `checkout` ist es möglich einen beliebigen `commit` in das Arbeitsverzeichnis zu kopieren. Hierzu werden die ersten fünf Stellen des `Commit`-Hashes benötigt, die man mit dem Befehl `status` ermitteln kann. Die gelöschte Datei `Main.java` aus dem ersten `commit` kann so wiederhergestellt werden.

```
P:\workspaces\git\hello-git>git checkout dafff
Note: switching to 'dafff'.
```

```
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.
```

```
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:
```

```
git switch -c <new-branch-name>
```

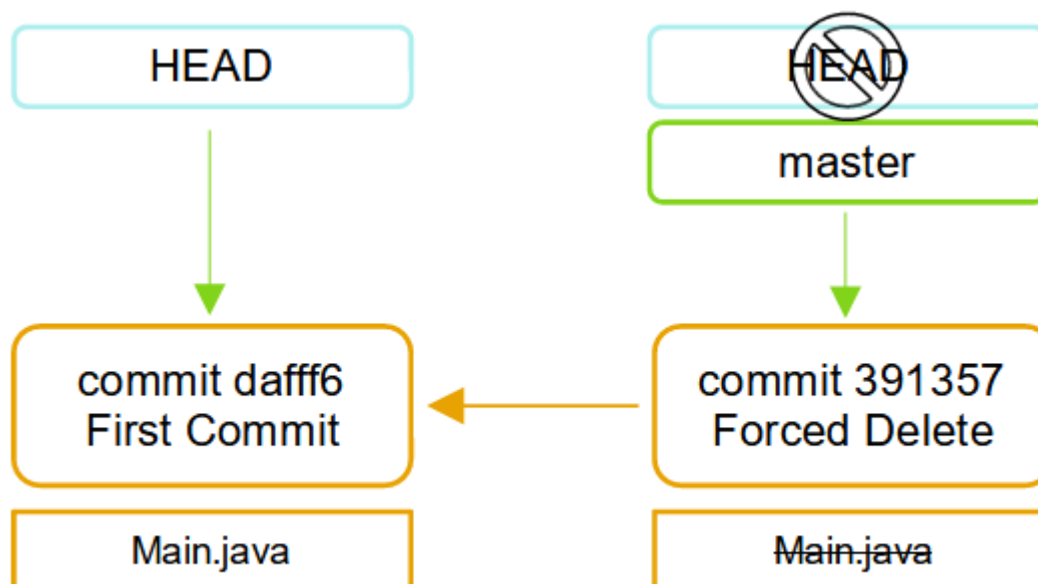
```
Or undo this operation with:
```

```
git switch -
```

```
Turn off this advice by setting config variable advice.detachedHead to false
```

```
HEAD is now at dafff6e First commit.
```

Die Datei `Main.java` liegt nun zwar wieder im `Working Directory`, allerdings hat sich der Zeiger (HEAD) auf den aktuellen Branch vom `master` gelöst. Er ist also wie *Git* mitteilt im `detached state`.



Da die `commits` bei losgelösten *Branches* keinem Namen zugeordnet werden können, machen

wir das ganze wieder rückgängig und holen uns explizit die Datei `Main.java` aus dem ersten `commit` und lassen somit den `HEAD` beim Branch `master`. Danach befindet sich die Datei wieder im Arbeitsverzeichnis und ist zugleich in der `Staging Area` als `added` bzw. `new` markiert.

```
P:\workspaces\git\hello-git>git checkout master
Previous HEAD position was dafff6e First commit.
Switched to branch 'master'

P:\workspaces\git\hello-git>git checkout dafff Main.java
Updated 1 path from ced46f7

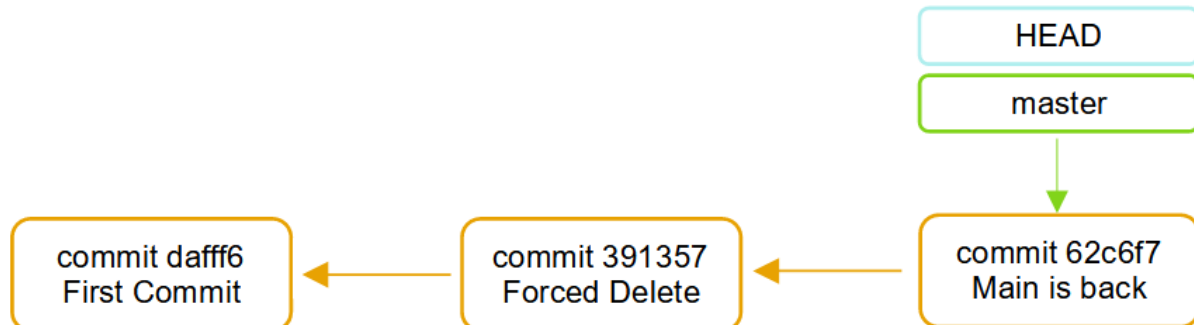
P:\workspaces\git\hello-git>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   Main.java
```

Bevor wir sie erneut `committen` fügen wir eine Zeile hinzu, und speichern Sie ab.

```
P:\workspaces\git\hello-git>git add Main.java

P:\workspaces\git\hello-git>git commit -m "Main.java is back!"
[master 62c6f7a] Main.java is back!
 1 file changed, 3 insertions(+)
 create mode 100644 Main.java
```

Die `Git-History` sieht nun so aus.

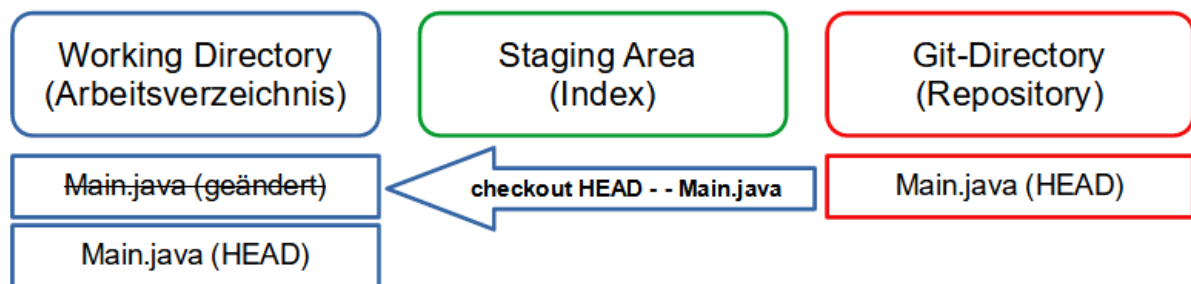


## Übung

1. Versuchen Sie Dateien aus einem *Repository* wiederherzustellen.

**Notiz:****Änderungen im Working Directory rückgängig machen**

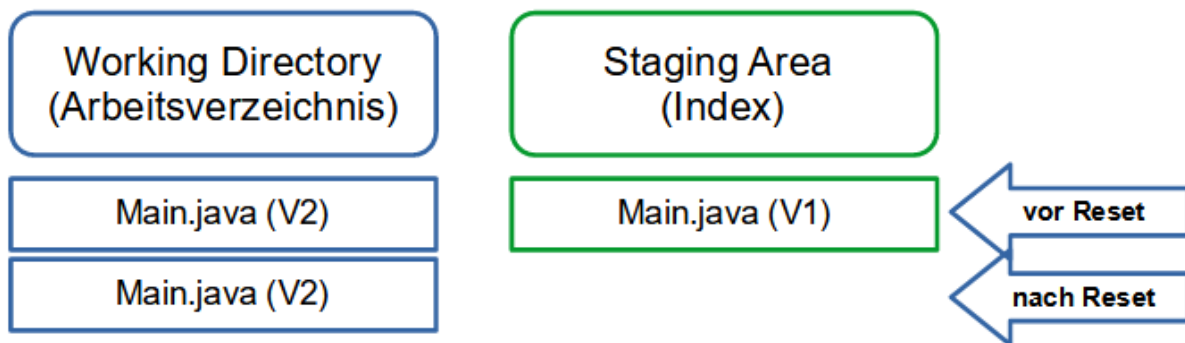
Hat man im **Working Directory** Änderungen gemacht, die man wieder verwerfen möchte, kann man den Zustand des letzten **commits** (also vom **HEAD**) wiederherstellen.



```
P:\workspaces\git\hello-git>git checkout HEAD -- Main.java  
  
P:\workspaces\git\hello-git>git status  
On branch master  
nothing to commit, working tree clean
```

**Änderungen in der Staging Area rückgängig machen**

Die Datei `Main.java` wurde verändert und der **Staging Area** hinzugefügt. Dies soll nun rückgängig gemacht werden. Der Befehl `add` soll also widerrufen gemacht werden.



```
P:\workspaces\git\hello-git>git add Main.java

P:\workspaces\git\hello-git>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Main.java

P:\workspaces\git\hello-git>git reset -- Main.java
Unstaged changes after reset:
M       Main.java

P:\workspaces\git\hello-git>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Main.java

no changes added to commit (use "git add" and/or "git commit -a")
```

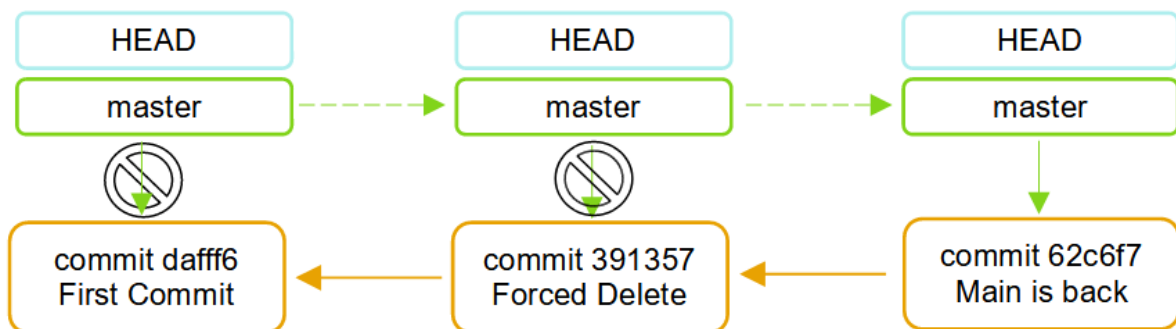
Möchte man alle Dateien aus der Staging Area entfernen verwendet man den Befehl 'reset' ohne Parameter.

## Übung

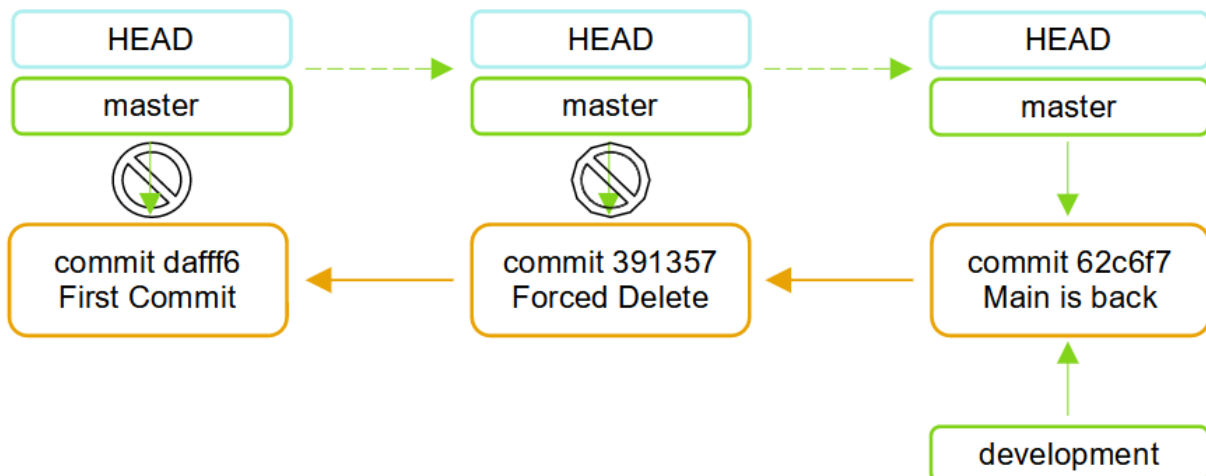
1. Machen Sie Änderungen im **Working Directory** rückgängig.
2. Machen Sie Änderungen in der **Staging Area** rückgängig

**Notiz:****Branches**

Wie bereits erwähnt ist ein Branch ein Zeiger auf einen `commit`. Mit dem ersten `commit` erstellt *Git* den Branch mit dem Namen *master*. Bei jedem weiteren Commit bewegt sich der Zeiger automatisch vorwärts.



Beim Erstellen eines neuen Branches wird ein neuer Zeiger auf dem `commit` erstellt auf dem man sich aktuell befindet. An welchem Branch aktuell gearbeitet wird erkennt Git am 'HEAD'-Zeiger.



```
P:\workspaces\git\hello-git>git branch development
```

Git hat nun zwar einen neuen Branch erzeugt, jedoch wurde er noch nicht gewechselt.

## Übung

1. Erstellen Sie einen neuen Branch.

## Notiz:

### Branch wechseln

Der Befehl 'checkout' ermöglicht das Wechseln eines Branches.

```
P:\workspaces\git\hello-git>git checkout development
Switched to branch 'development'
M       Main.java

P:\workspaces\git\hello-git>git status
On branch development
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Main.java

no changes added to commit (use "git add" and/or "git commit -a")
```

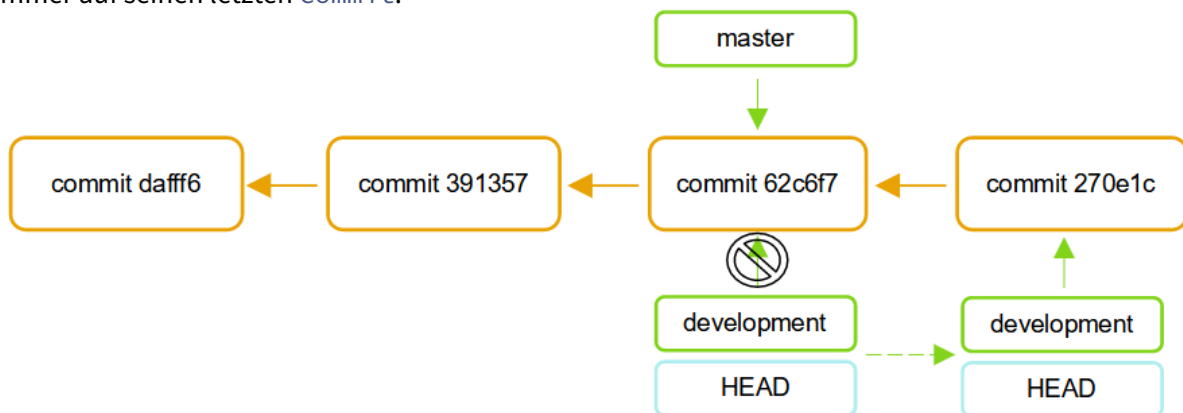
Nun wurde der Branch nach *development* gewechselt. Da er noch auf den gleichen *commit* wie der *master-Branch* zeigt existiert die lokale Datei immer noch und bleibt im Status *modified*.

Zum besseren Verständnis von Branches erstellen wir eine neue Datei *Feature.java* und *committen* diese ins Repository.

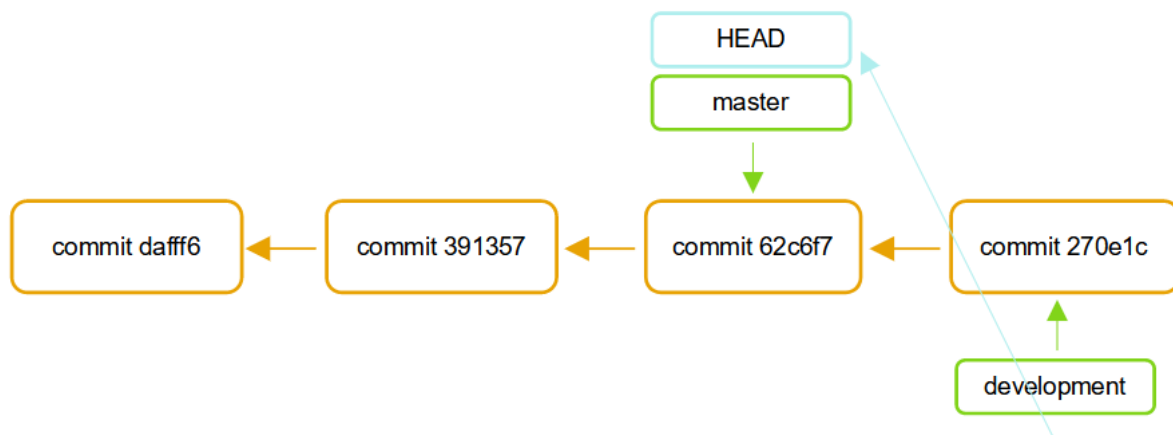
```
P:\workspaces\git\hello-git>git add Feature.java

P:\workspaces\git\hello-git>git commit -m "new Feature"
[development 270e1cf] new Feature
 1 file changed, 3 insertions(+)
 create mode 100644 Feature.java
```

Durch den `commit` hat sich der *HEAD*-Zeiger vorwärts bewegt. Der *master*-Branch zeigt jedoch noch immer auf seinen letzten `commit`.



Als nächstes wechseln wir zurück zum *master*-Branch.



```
P:\workspaces\git\hello-git>git checkout master
Switched to branch 'master'
M    Main.java
```

**Achtung:** Alle Dateien aus dem *Working directory* sind nun auf dem Stand des letzten `commits` des *master*-Branches. Somit ist die Datei `Feature.java` aus dem *development*-Branch nicht mehr im *Working Directory* vorhanden.

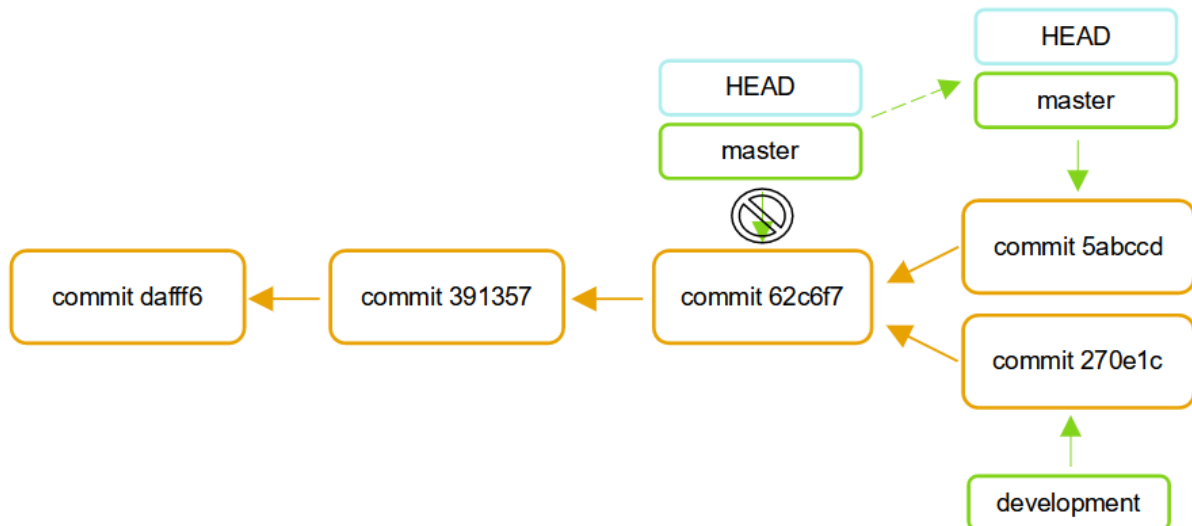
**Damit lokale Änderungen nicht verloren gehen empfiehlt es sich vor jedem Branchwechsel zu `committen`!**

Nun kann jedoch auf dem *master*-Branch entwickelt werden und jederzeit zum *development*-Branch gewechselt werden.

Ein `commit` auf dem *master*-Branch mit einer Änderung verändert das *Repository* wie folgt:



```
P:\workspaces\git\hello-git>git add Main.java  
  
P:\workspaces\git\hello-git>git commit -m "Hello World"  
[master 5abccd8] Hello World  
1 file changed, 4 insertions(+), 1 deletion(-)
```



## Übung

1. Wechseln Sie zwischen den Branches hin und her und beobachten Sie die Veränderungen.

## Notiz:

## Mergen

Führt ein **mergen** zu Unstimmigkeiten wird kein *Merge-Commit* erstellt, sondern in der entsprechenden Datei ein Konfliktmarker erstellt. Bei uns wurde auch im **master-Branch** eine Datei `Features.java` **committed**, die sich jedoch von der im *development-Branch* unterscheidet.


```
P:\workspaces\git\hello-git>git add Main.java

P:\workspaces\git\hello-git>git commit -m "Hello World"
[master 5abccd8] Hello World
 1 file changed, 4 insertions(+), 1 deletion(-)

P:\workspaces\git\hello-git>git add Feature.java

P:\workspaces\git\hello-git>git commit -m "feature dev"
[master fc4aba4] feature dev
 1 file changed, 5 insertions(+)
 create mode 100644 Feature.java

P:\workspaces\git\hello-git>git merge development
Auto-merging Feature.java
CONFLICT (add/add): Merge conflict in Feature.java
Automatic merge failed; fix conflicts and then commit the result.
```

 Feature.java - Editor

Datei Bearbeiten Format Ansicht Hilfe

```
class Feature{
```

```
<<<<<< HEAD
    //fancy features here!
```

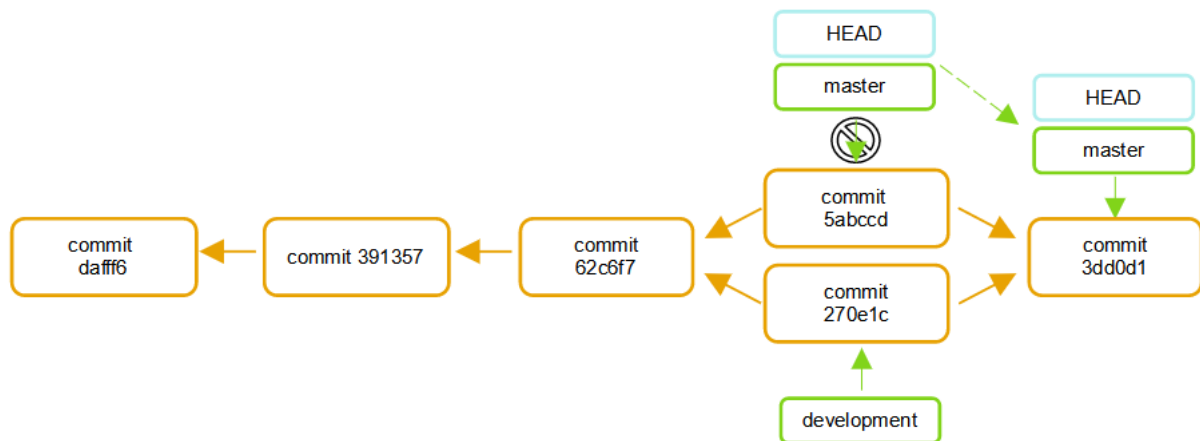
```
=====
>>>>>> development
}
```

Um den *Merge-Prozess* erfolgreich zu beenden entfernt man die Marker und verändert die Datei so, wie sie letztendlich aussehen soll. Nach dem `stagen` und anschließenden `committen` wird der `merge` schließlich abgeschlossen.

```
P:\workspaces\git\hello-git>git add Feature.java

P:\workspaces\git\hello-git>git commit -m "merge finally"
[master 3dd0d1e] merge finally

P:\workspaces\git\hello-git>git status
On branch master
nothing to commit, working tree clean
```



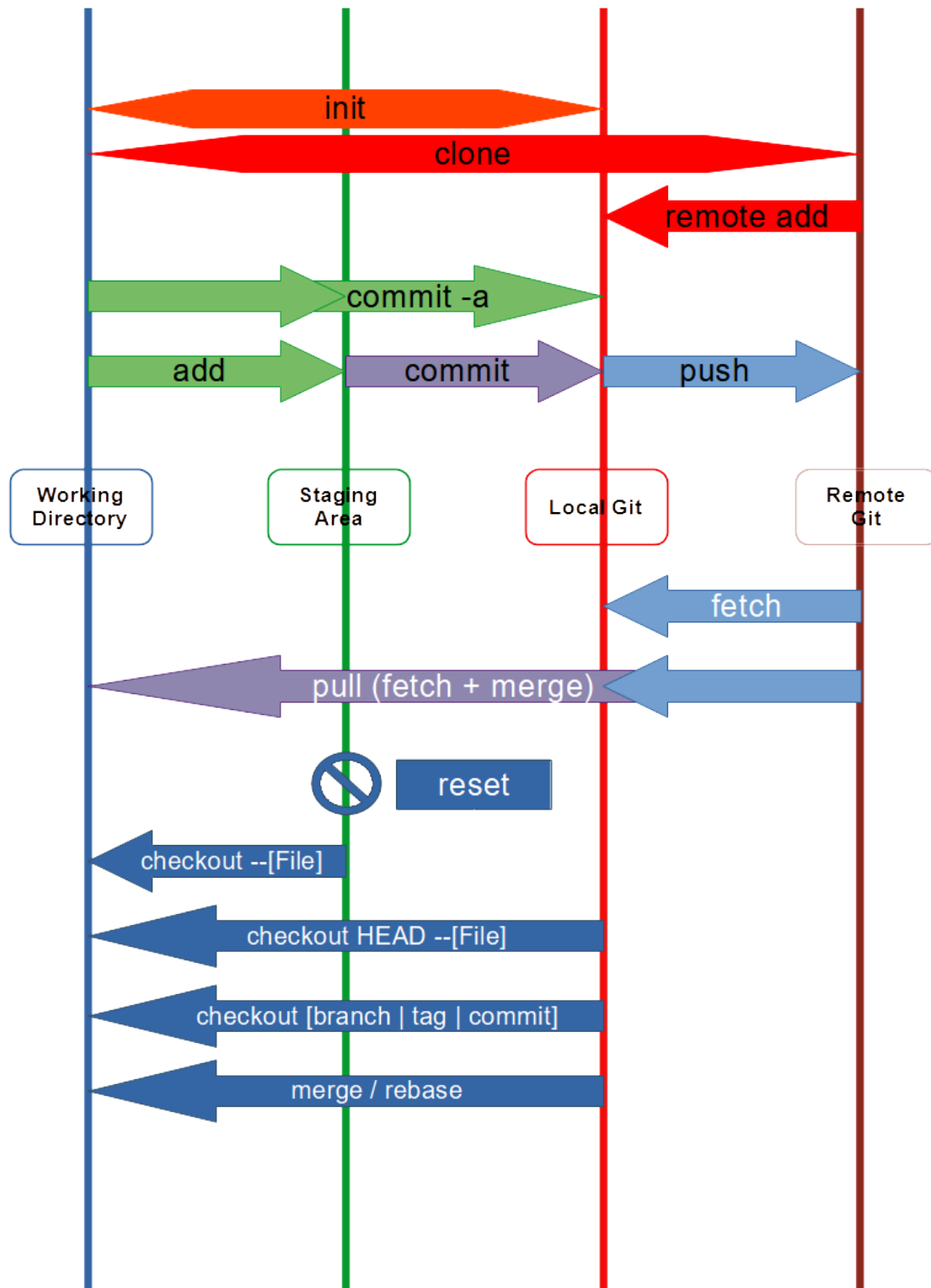
## Übung

1. Führen Sie Ihre Branches wieder zusammen.

## Notiz:



## Git mit Remote Repository (Überblick)



## Git WorkFlow

Während der Entwicklungsarbeit wird der Workflow in Git meistens so aussehen:

1. Aktuellen master-Branch vom Server downloaden (vorher mit git status sicherstellen, dass man sich auf dem master-Branch befindet. Wenn nicht: git checkout master):  
**git pull**
2. Einen neuen Branch für das kommende Feature anlegen:  
**git checkout -b myfeature master**
3. [Änderungen am Code durchführen]
4. Geänderte und neue Dateien stagen:  
**git add**
5. Änderungen committen:  
**git commit -m „changes“**
6. Jetzt den Branch auf den Server laden, wenn gewünscht:  
**git push -u origin myfeature**
7. ... oder direkt in den master wechseln:  
**git checkout master**
8. (nochmals aktuellen Code ziehen – zur Sicherheit)  
**git pull**
9. Eigenen Code mit master zusammenführen:  
**git merge myfeature**
10. Nicht mehr benötigten Branch löschen:  
**git branch -d myfeature**
11. Aktualisierten master auf den Server pushen:  
**git push**
12. Fertig :)

**Abschlussübung**

1. Erstellen Sie sich einen Account bei einem Remote-Repository-Anbieter.
2. Installieren Sie sich ein entsprechendes Plugin in Ihrer Entwicklungsumgebung.
3. Testen Sie die Funktionalitäten des Plugins.

**Notiz:**