GIT

In der Regel wird der Code in größeren Projekten nicht nur von einer Person bearbeitet, sondern teilweise arbeiten mehrere Entwickler u.U. zeitgleich daran.

Dadurch ist eine Versionsverwaltung des Codes unabdingbar, um beispielsweise ein Überschreiben/Löschen des Codes anderer Personen (oder auch des eigenen Codes) zu verhindern. Das weitverbreitetste System zur Versionsverwaltung ist *Git* und wurde 2005 von Linus Torvalds, dem "Erfinder" des Linux-Kernels, entwickelt.

Projekte werden dabei innerhalb sogenannter *"Repositories"* verwaltet.

Verteilter Ansatz

Git ist im Gegensatz zu einigen anderen Versionsverwaltungssystemen verteilt (distributed). Das bedeutet, dass es keinen zentralen Server gibt, auf dem die gesamte Codebasis verwaltet wird, sondern jeder Nutzer eine lokale Kopie des Repositories besitzt, an dem er arbeitet und anschließend lediglich die Änderungen an den Server schickt ("push"), sodass sich die anderen Mitglieder diese Änderungen jeweils in ihr lokales Repository herunterladen können ("pull").

Workflow

Erstellen/Klonen eines Repositories

Soll eine neues Projekt erstellt werden, so wird dazu ein neues Repository im entsprechenden Projektverzeichnis initialisiert:

```
git init
```

Soll dagegen eine lokale Kopie eines bereits bestehenden Repositories erstellt werden, so kann dieses Projekt *"geklont"* werden:

```
git clone <URL>
```

Hinzufügen von Änderungen

Nun können in diesem Projektverzeichnis beliebige Dateien erstellt/hinzugefügt werden. Um diese Änderungen zu "speichern", müssen die Dateien zunächst zum *Index* hinzugefügt werden (*"Staging"*):

`git add <dateiname>` (lediglich die Änderungen der entsprechenden Datei werden gestaged) oder `git add *` (alle Änderungen werden gestaged)

Dadurch sind die Änderungen aber noch nicht abgespeichert, sondern lediglich "vorgeschlagen", also zum Index hinzugefügt. Um diese Änderungen letztendlich zu bestätigen, müssen diese durch einen *"Commit"* bestätigt werden. Diese Commits werden später auch den anderen Mitgliedern im Repository angezeigt. Aus diesem Grund muss mit jedem Commit auch eine kurze Nachricht mit angegeben werden, was mit diesem Commit geändert/hinzugefügt/gelöscht wurde:

```
git commit -m "Kurze Beschreibung der Änderungen"
```

Nun wurden die Änderungen, die zuvor mit `git add` gestaged wurden, committed, also vollzogen. Damit befinden sich die Änderungen im sogenannten `HEAD`. Dabei handelt es sich um eine Referenz, die jeweils auf den letzten Commit verweist.

So wurden die Änderungen zwar lokal im Repository gespeichert, die Änderungen wurden jedoch noch nicht hochgeladen, sodass die anderen Mitglieder keine Kenntnis über diesen Commit besitzen. Um die Änderungen also an einen Git-Server zu übertragen, müssen diese Commits *"gepusht"* werden:

```
git push origin master
```

`master` bezeichnet dabei den *Zweig*, dessen Änderungen an das entfernte Repository gepusht werden sollen. Standardmäßig finden alle Änderungen auf dem `master`-Branch statt, es können aber auch weitere Zweige erstellt werden. `origin` beschreibt, an welches entfernte Repository die Änderungen gesendet werden sollen. Falls das Repository jedoch mit `git init` neu erstellt wurde, weiß Git nicht, unter welcher URL der Git-Server zu erreichen ist. In diesem Fall muss die Origin einmalig hinzugefügt werden, bevor die Änderungen an das Remote Repository gepusht werden können:

```
git remote add origin <URL>
```

Wurde das Projekt jedoch nicht neu erstellt sondern geklont, so verweist die `origin` automatisch auf die URL, von der geklont wurde, der o.g Befehl ist dann also nicht notwendig.

Pullen & Merge-Konflikte

Um die Änderungen von anderen Bearbeitern des Repositories in das eigene lokale Repository herunterzuladen, müssen die Commits gepullt werden.

```
git pull
```

Dabei werden die Änderungen zunächst heruntergeladen (`fetch`) und anschließend in die eigene Arbeitskopie bzw. den Verlauf des lokalen Repositories eingearbeitet (`merge`).

Das Mergen (Zusammenführen) der Änderungen geschieht dabei in der Regel automatisch und bedarf keines weiteren Eingriffs durch den Nutzer. Wenn jedoch eine Datei sowohl im lokalen Repository als auch durch einen anderen Nutzer geändert wurde, können *"Konflikte"* auftreten, da Git nicht weiß, welche der beiden Änderungen übernommen werden soll. Diese Konflikte müssen dann manuell behoben werden (Git kennzeichnet die Konflikte direkt in den entsprechenden Dateien). Die Änderungen können mit

```
git diff
```

angezeigt werden. Nachdem die Konflikte behoben wurden, müssen diese Änderungen erneut gestaged und committed werden.

Wechseln zwischen Commits

Um zu einer früheren Version des Projekts zu wechseln, kann einfach zwischen den einzelnen Commits gewechselt werden. Jeder Commit besitzt einen eindeutigen Hash-Wert, der aus den Inhalten des Commits generiert wird, z.B. `4a73b89bfbe2bd2fea05be33cd1b281c49720bcd`. Dieser Hash-Wert kann genutzt werden, um zu diesem Commit zurückzuspringen. Da der Wert jedoch sehr lang ist und nur umständlich einzutippen ist, können die Hash-Werte abgekürzt werden, indem lediglich die ersten 7 Stellen berücksichtigt werden, also `4a73b89`. Dies sollte ausreichen, um einen Commit innerhalb eines Repositories eindeutig zu identifizieren (16^7 = 268.435.456 mögliche Hash-Werte).

Um zu diesem Commit zu springen, kann der `checkout`-Befehl genutzt werden:

```
git checkout 4a73b89
```

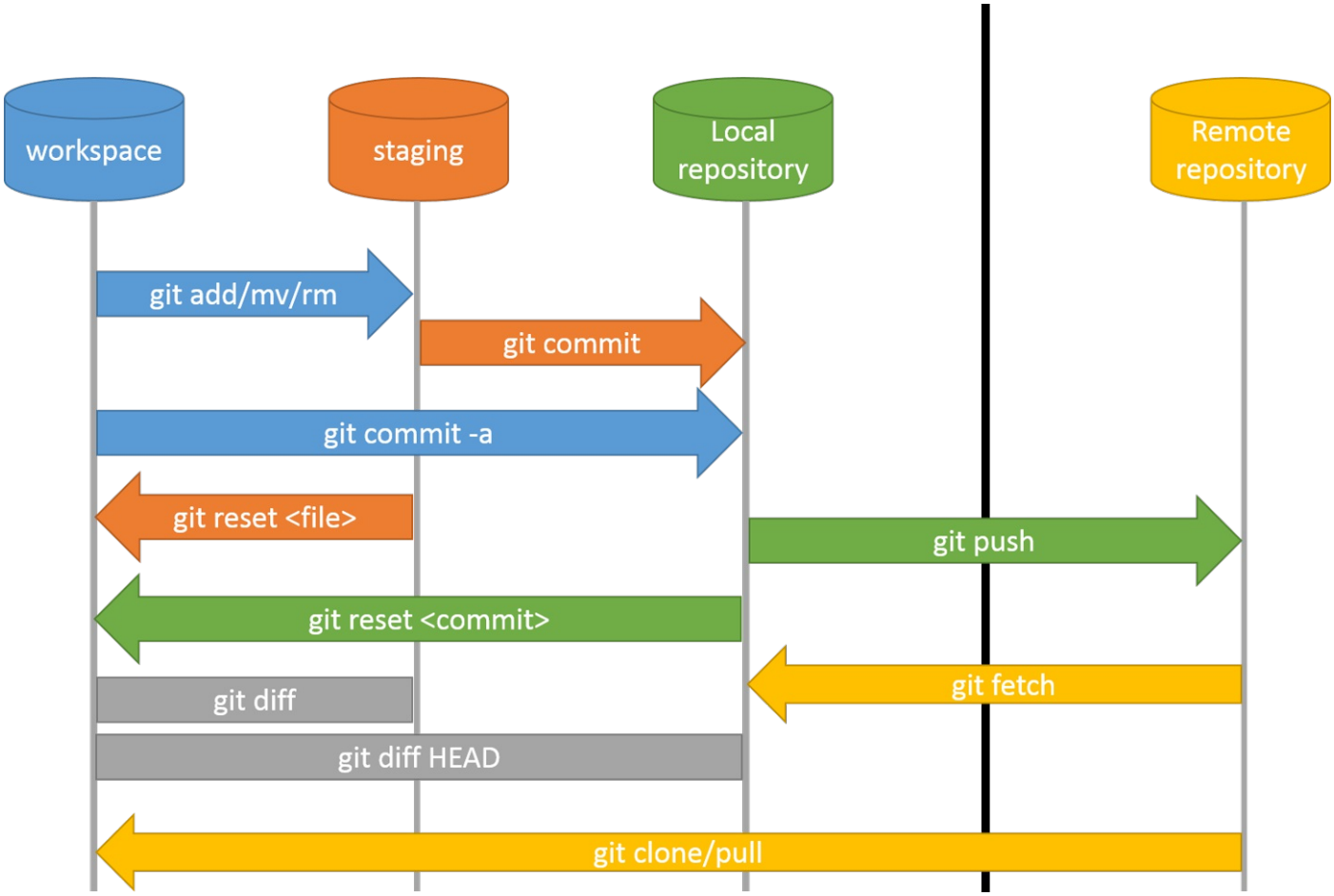
Hierdurch verweist die aktuelle Referenz (HEAD) auf diesen Commit, alle Dateien werden auf den Status dieses Commits zurückgesetzt.

Wichtig: Hierdurch ändert sich die History des Repositories nicht. Commits, die nach dem jeweiligen Commit erstellt wurden, bleiben erhalten.

Hinweis: Um alle bisherigen Commits anzuzeigen, kann der Befehl `git log --graph --oneline --all` genutzt werden.

Zusammenfassung

Git erlaubt es mehreren Entwicklern gleichzeitig an einem Projekt zu arbeiten, ohne dass es dadurch zu Problemen wie unbeabsichtigtes Überschreiben/Löschen oder kompliziertes Verwalten mehrerer Versionen kommt.



(Quelle: https://twiki.cern.ch/twiki/pub/BL4S/BL4SGitGuide/)

Aufgabe:

Teamaufgabe: Bildet 2er Teams

1. Person A initialisiert ein neues lokales Repository
2. Person A erstellt in diesem Repository zwei verschiedene Textdateien (*file1.txt* und *file2.txt*), fügt diese zum Repository hinzu und erstellt durch eine Änderung der Dateien mehrere Commits
3. Person A pusht diese Commits an ein Remote-Repository (GitHub-Account erforderlich!)
4. Person B klont dieses Repository auf seinen Rechner
5. Person A ändert die Datei *file1.txt*, Person B bearbeitet gleichzeitig die Datei *file2.txt*. Beide Personen pushen diese Änderungen an das Remote-Repository und beide Personen pullen die Änderungen des jeweils anderen
6. Person A und B arbeiten gleichzeitig an der Datei *file1.txt* und pushen diese Änderung an die Remote, der jeweils andere pullt diese Änderungen. Beide beheben die Merge-Konflikte