Versionsverwaltungssystem

Was ist das?

Versionsverwaltung ist ein System, welches die Änderungen an einer oder einer Reihe von Dateien über die Zeit hinweg protokolliert, sodass man später auf eine bestimmte Version zurückgreifen kann.

https://git-scm.com/book/de/v1/Los-geht%E2%80%99s-Git-Grundlagen

Aber warum überhaupt?

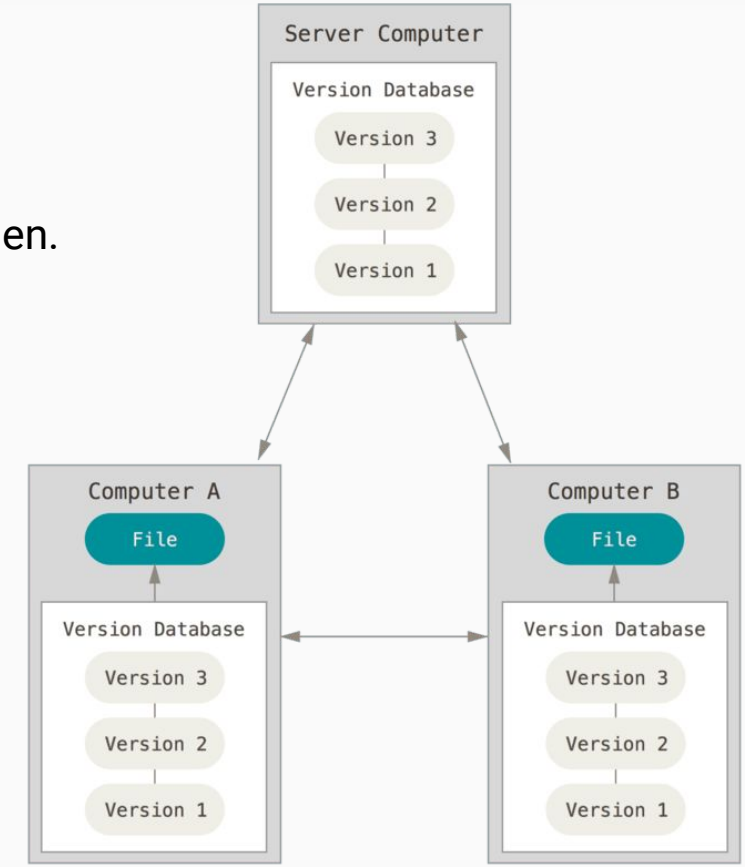
Aber warum überhaupt?

- Nachvollziehbares Archiv eines Projektes
- Unterschiede zwischen Dateien erkennen
- Erleichtert das Arbeiten im Team
- Einheitliche Workflows
- Qualitätssicherung

Git nutzt verteilte Repositories

Ein **Repository** ist ein Archiv für ein Projekt und beinhaltet alle Änderungen und Versionen der Dateien.

- Es gibt mehrere Repositories
- Entwickler haben lokal ein komplett geklontes Repository
- Ein zentrales Repository ist möglich (meistens auch sinnvoll)
- Commits können ohne Verbindung zum zentralen Repository erledigt werden



Git nutzt verteilte Repositories

Vorteile:

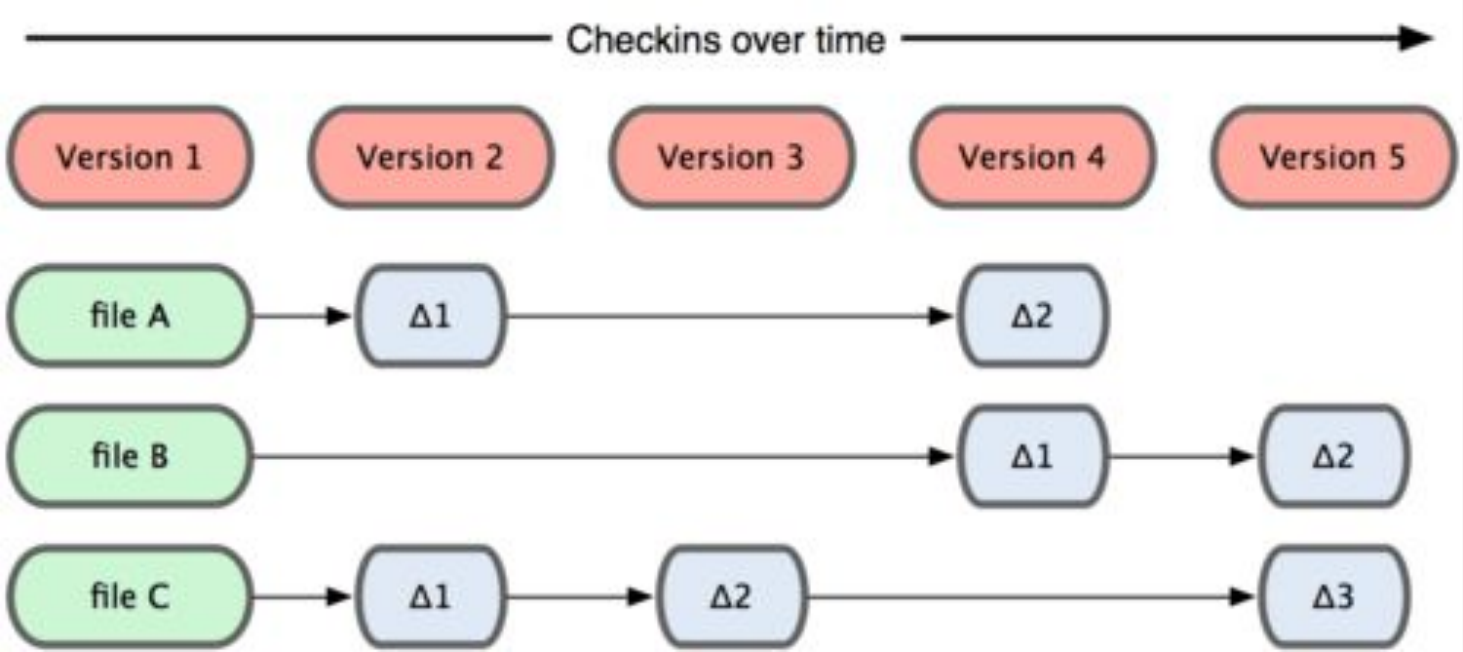
- Schnelleres Arbeiten im Projekt, da keine Verbindung zum Server (außer push/pull)
- Änderungen können lokal bearbeitet werden, bevor ins Hauptrepo gepushed wird
- Hauptrepository kann sauber gehalten werden (frei von Entwicklerbranches, nur relevante Branches werden veröffentlicht)

Nachteile

- Große Projekte mit vielen Änderungen/Branches benötigen viel Platz
- Erster Clone kann lange dauern

Snapshots, nicht Diffs

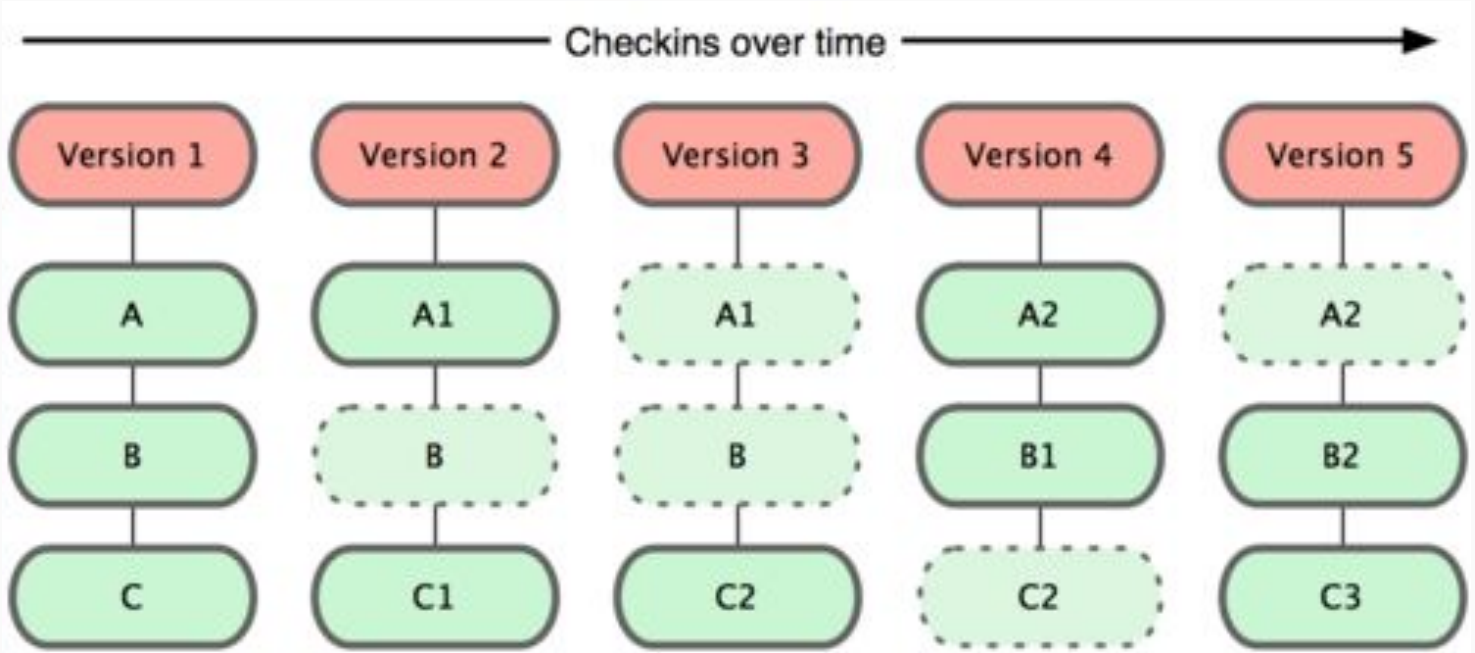
Andere Systeme speichern Daten als Änderungen an einzelnen Dateien einer Datenbasis



Quelle: https://git-scm.com/book/de/v1/Los-geht%E2%80%99s-Git-Grundlagen

Snapshots, nicht Diffs

Git speichert die Daten-Historie eines Projekts in Form von Snapshots



https://git-scm.com/book/de/v1/Los-geht%E2%80%99s-Git-Grundlagen

Technisches

- Git stellt Integrität sicher (Prüfsummen der Daten werden mit gespeichert)
- Git hängt normalerweise nur Daten an, und löscht sie nicht
- Sobald man einen Schnappschuss in Git eingecheckt hat, ist es sehr schwierig, diese Daten wieder zu verlieren, insbesondere wenn man regelmäßig seine lokale Datenbank auf ein anderes Repository hochlädt

https://git-scm.com/book/de/v1/Los-geht%E2%80%99s-Git-Grundlagen

Begriffsklärung

Branch Ein Branch beschreibt zusammenhängende Änderungen in einem Projekt. Es gibt mindestens einen Branch und es kann beliebig viele in einem Projekt geben.

Commit Ein Commit ist ein einzelner Änderungseintrag in einem Branch. Es beschreibt mindestens eine Änderung an einer Datei und enthält eine Beschreibung der Änderung.

Begriffsklärung

Clone	Clone beschreibt den Download eines gesamten Repositorys.
Push	Ein Push schiebt lokale Änderungen in einem Branch in ein Ziel-Repository.
Pull	Ein Pull lädt Änderungen eines Branches von einem Repository in das lokale Repository.
Fetch	Ein Fetch holt die Informationen zu Änderungen von einem Repository.
Checkout	Ein Checkout liest einen bestimmten Commit aus.

Die drei Zustände

Git definiert drei Hauptzustände, in denen sich eine Datei befinden kann:

modified

- eine Datei wurde geändert, aber noch nicht in die lokale Datenbank eingecheckt

staged

- eine geänderte Datei ist in ihrem gegenwärtigen Zustand für den nächsten Commit vorgemerkt

committed

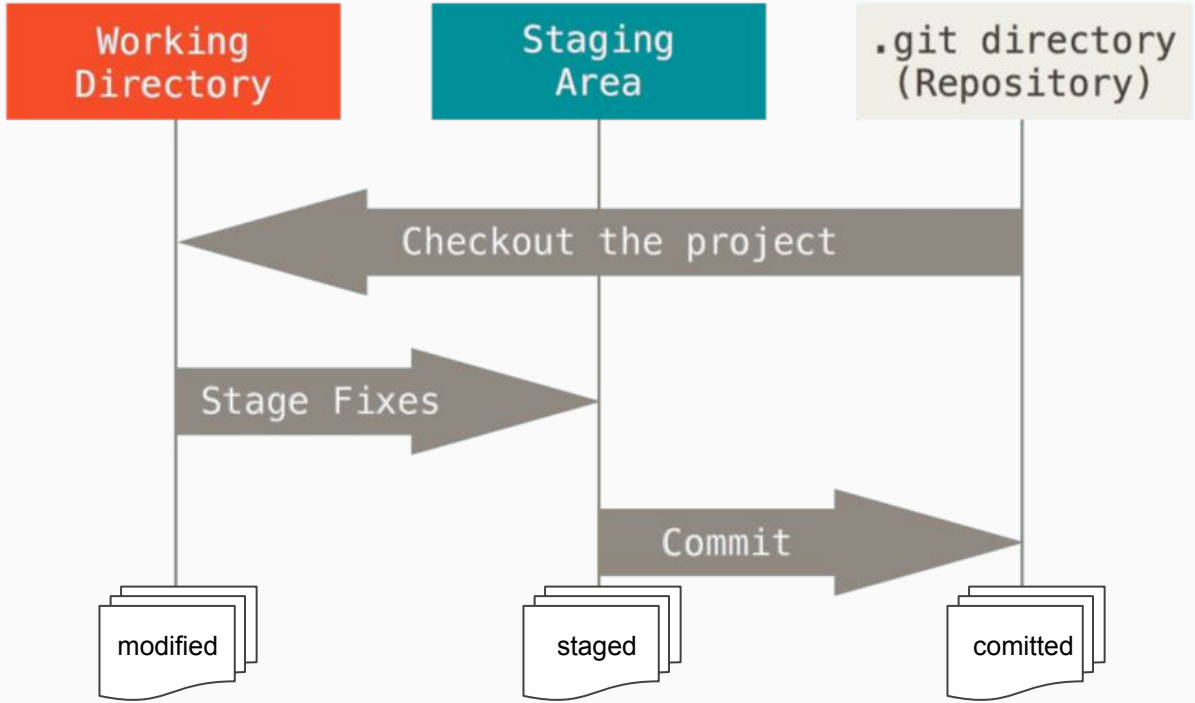
- Daten sind sicher in der lokalen Datenbank gespeichert

https://git-scm.com/book/de/v1/Los-geht%E2%80%99s-Git-Grundlagen

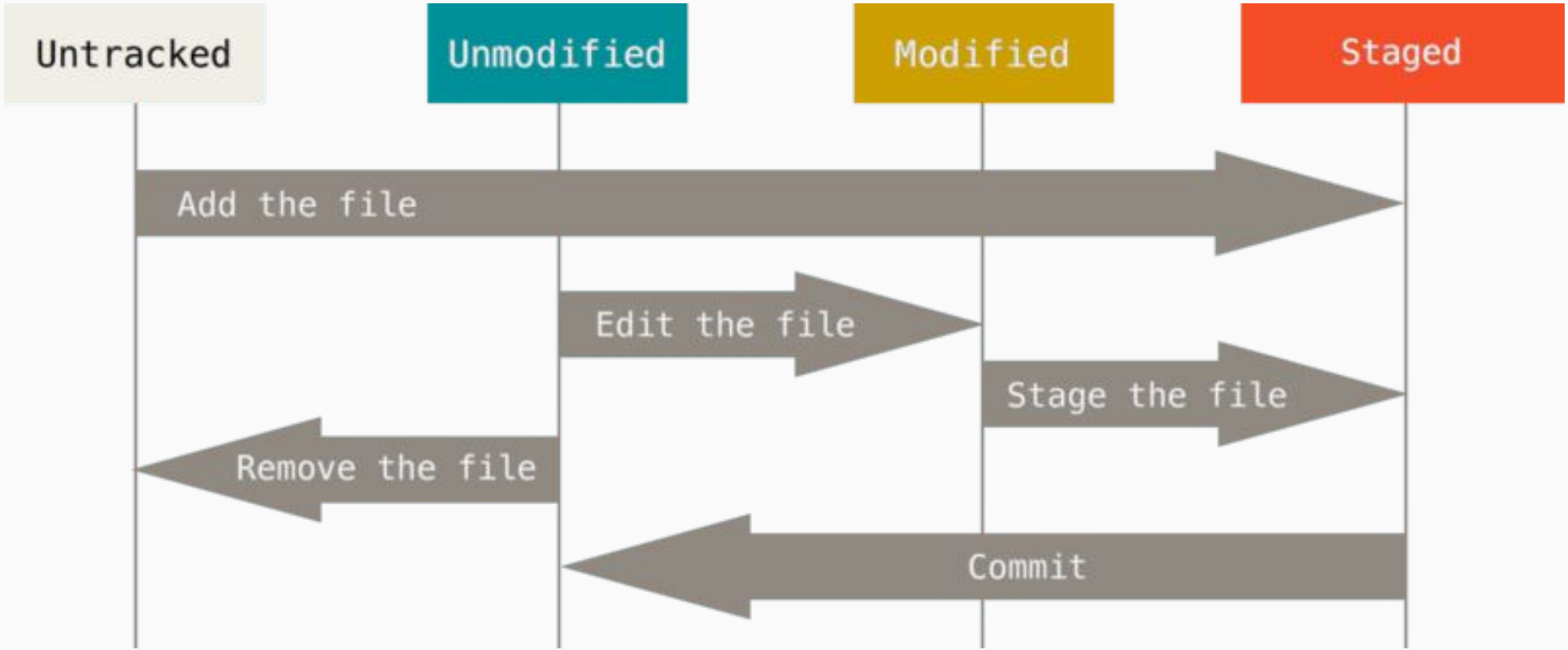
Git: Änderungen

Workflow

1. Datei ändern
2. Zum index hinzufügen (stagen)
3. commiten

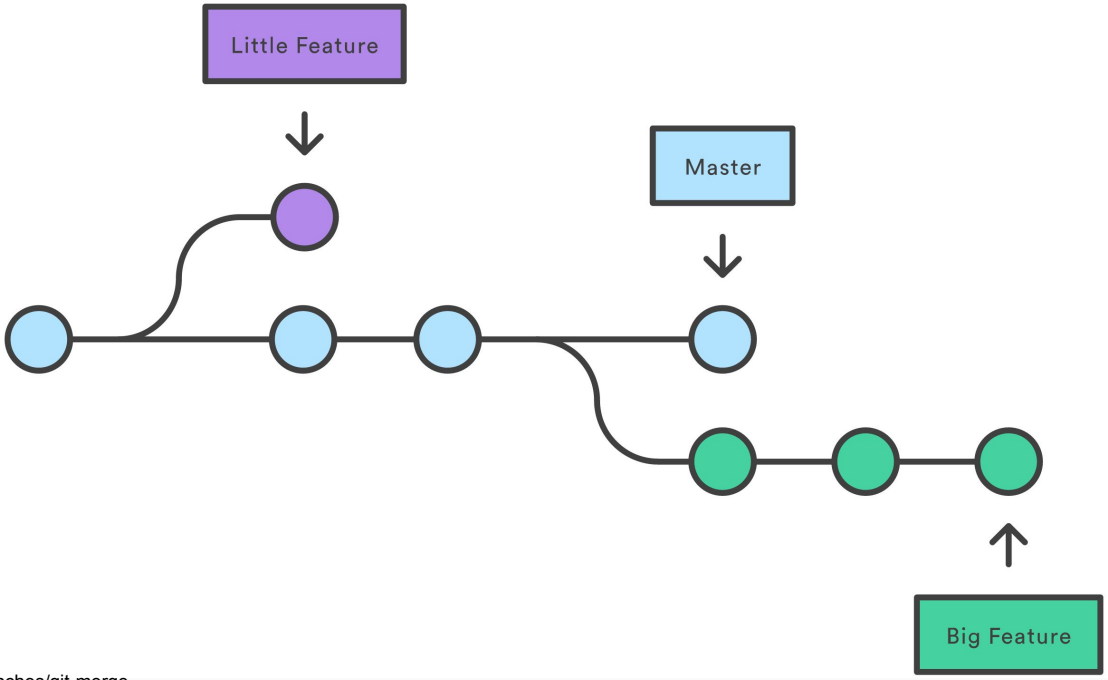


Lifecycle von Dateien



Git: Branches

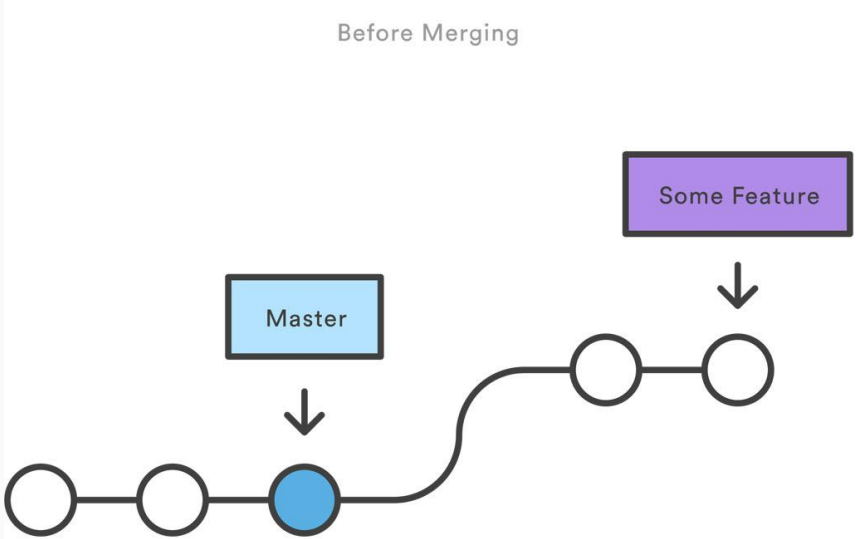
Ein **Branch** beschreibt zusammenhängende Änderungen in einem Projekt. Es gibt mindestens einen Branch und es kann beliebig viele in einem Projekt geben.

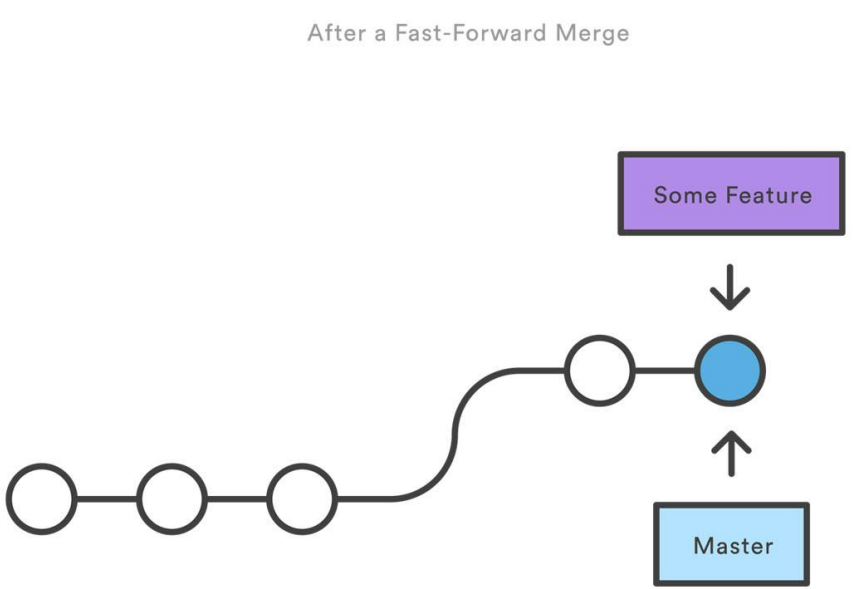


https://de.atlassian.com/git/tutorials/using-branches/git-merge

Git: FastForwardMerge

Ein **Merge** beschreibt das Zusammenführen von Änderungen aus zwei unterschiedlichen Branches.

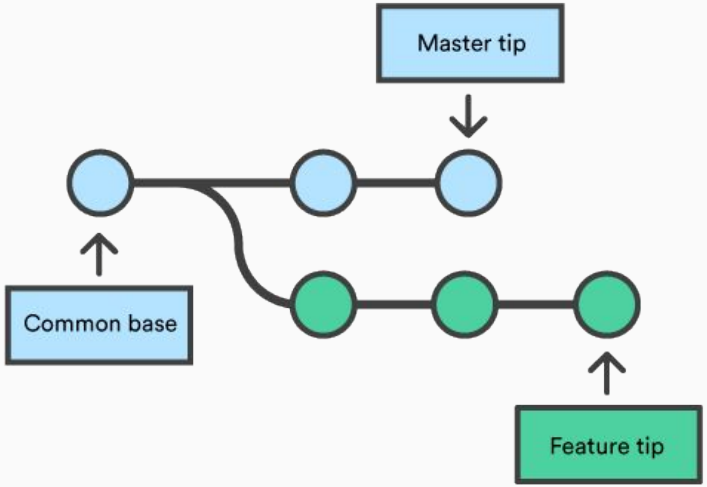


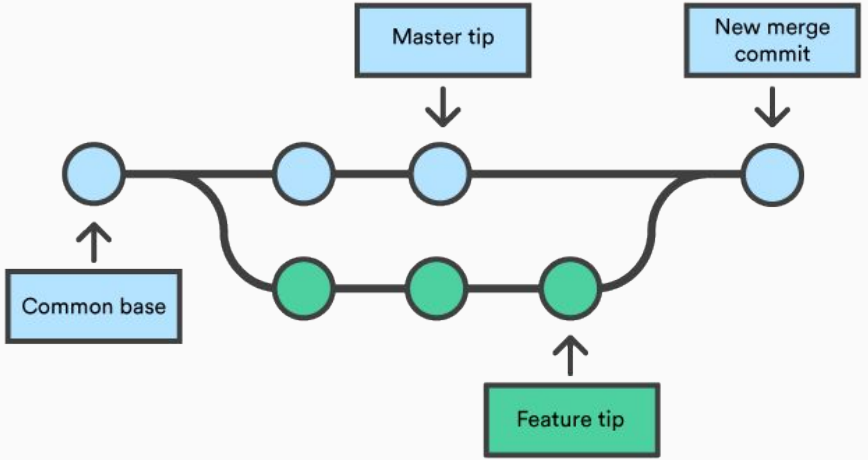


https://de.atlassian.com/git/tutorials/using-branches/git-merge

Git: MergeCommit

- git merge sucht zwischen zwei Commit-Pointern, einen gemeinsamen Basis-Commit.
- Sobald Git den gemeinsamen Basis-Commit gefunden hat, wird ein neuer "Merge-Commit" erstellt, um die Änderungen jeder Abfolge von Merge-Commits in der Warteschlange zusammenzuführen.



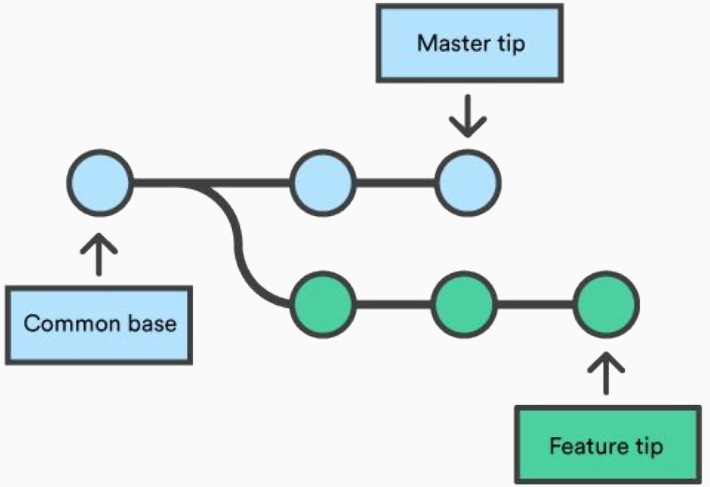


https://de.atlassian.com/git/tutorials/using-branches/git-merge

Git: Merge vorbereiten

Damit das Mergen problemlos funktioniert, sollte man vorbereitend einige Schritte durchführen.

- Den Merge-Ziel-Branch bestätigen (git status; git checkout master)
- Neueste Remote-Commits abrufen (git fetch; git pull)
- Verschmelzung mit “git merge <branchname>” (branchname --> Name des Branches, der in den Merge-Ziel-Branch gemergt wird)



https://de.atlassian.com/git/tutorials/using-branches/git-merge

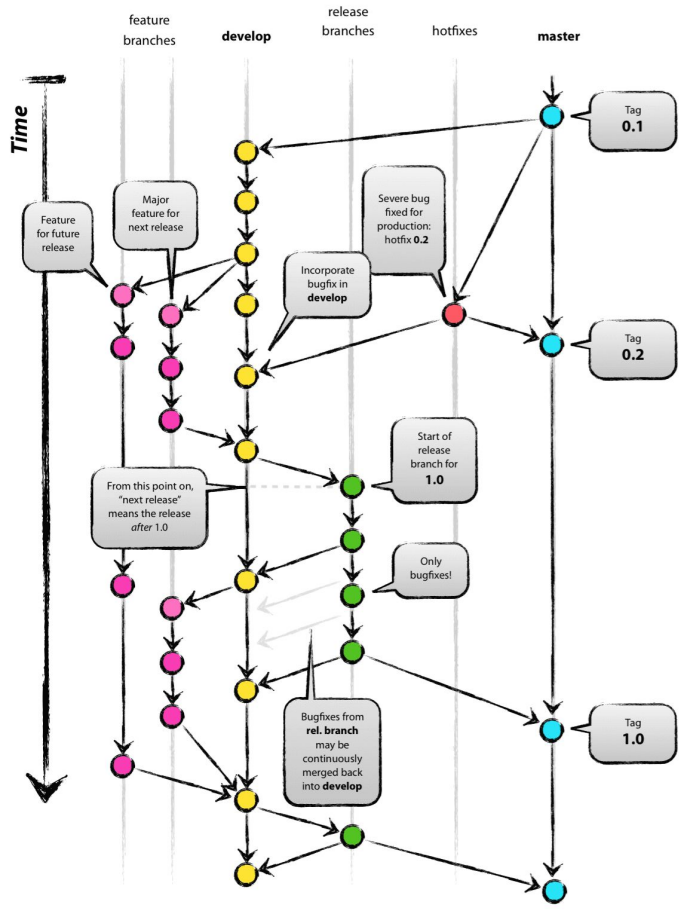
Git: Rebase

Ein **Rebase** wendet die Commits eines Branches auf den Basis-Branch an. Dadurch entstehen neue Commits und eine saubere lineare Commit-History

Der Projektverlauf wird beim Rebasing *neu geschrieben*, indem für jeden Commit im originalen Branch völlig neue Commits erstellt werden

Mehr Infos:
https://de.atlassian.com/git/tutorials/merging-vs-rebasing

Workflow



Quelle: https://nvie.com/posts/a-successful-git-branching-model/

Weiterführend

Informationen
https://de.atlassian.com/git/tutorials
https://git-scm.com/book
https://nvie.com/posts/a-successful-git-branching-model
https://rogerdudler.github.io/git-guide/index.de.html

Git GUIs
https://www.gitkraken.com
https://www.git-tower.com

Online Repos
https://bitbucket.org/
https://github.com
https://about.gitlab.com/