dataport

anderScore
trust in competence

Tag 1: Einführung in Git und GitLab

08.07.2024, Daniel Krämer

HECKER
CONSULTING

Agenda

- **Tag 1 – Einführung in Git und GitLab**
  - Einführung & Kursüberblick
  - Grundlagen von Git
  - Git Rebase und Merge-Strategien
  - Git Remote
  - Grundlagen von GitLab
- **Tag 2 – Git-Workflows, CI/CD, GitLab CI**
  - Git-Workflow im Team
  - Gitflow-Workflow
  - Tags, Releases & deren Verwaltung
  - Einführung in GitLab CI/CD & gitlab-ci.yml
  - GitLab Runner
- **Tag 3 – Docker, GitOps, Deployment-Strategien**
  - Entwicklung mit Docker
  - Container/Docker-Registry
  - Erstellen von Release- und Tagged-Images
  - GitOps Grundlagen
  - Möglichkeiten des Deployments & Verwaltung von Konfiguration
  - Abschlussübung & Diskussion

Agenda

- **Tag 1 – Einführung in Git und GitLab**
  - Einführung & Kursüberblick
  - Grundlagen von Git
  - Git Rebase und Merge-Strategien
  - Git Remote
  - Grundlagen von GitLab
- **Tag 2 – Git-Workflows, CI/CD, GitLab CI**
  - Git-Workflow im Team
  - Gitflow-Workflow
  - Tags, Releases & deren Verwaltung
  - Einführung in GitLab CI/CD & gitlab-ci.yml
  - GitLab Runner
- **Tag 3 – Docker, GitOps, Deployment-Strategien**
  - Entwicklung mit Docker
  - Container/Docker-Registry
  - Erstellen von Release- und Tagged-Images
  - GitOps Grundlagen
  - Möglichkeiten des Deployments & Verwaltung von Konfiguration
  - Abschlussübung & Diskussion

Inhalt

- Einführung
  - Version Control System
  - Zentrale und dezentrale Versionsverwaltung
- Grundlagen von Git
  - Was ist Git?
  - Konfiguration
  - Projekte und Repositories
  - Commits
  - Branches
  - Zurücksetzen von Commits

Grundlagen und Konzepte eines

Version Control System

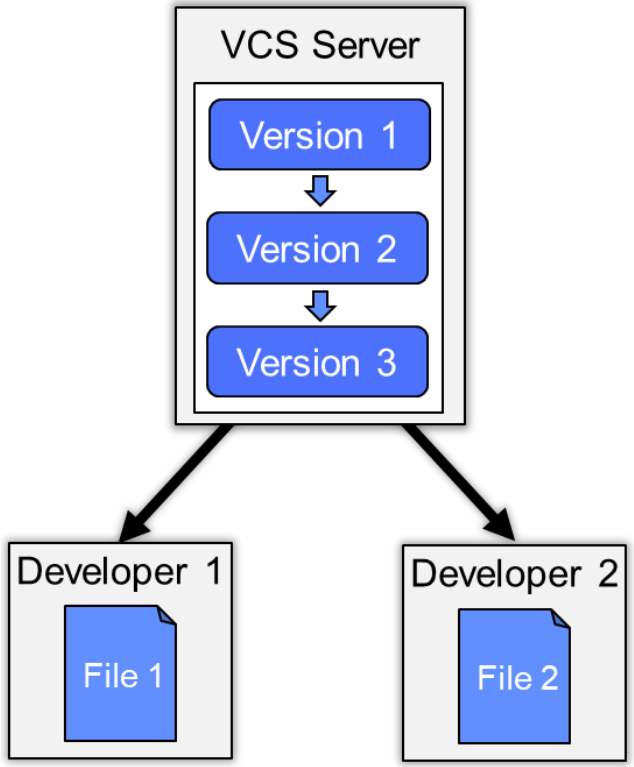
Version Control System

- Erfassung von Änderungen an Dateien und Dokumenten
- Protokollierung: Änderung, Zeitstempel, Autor
- Häufiger Use Case: Verwaltung von Quellcode
- Unterscheidung zwischen
  - (Lokaler Versionsverwaltung)
  - Zentraler Versionsverwaltung
  - Verteilte Versionsverwaltung

Version Control System

Zentrale Versionsverwaltung

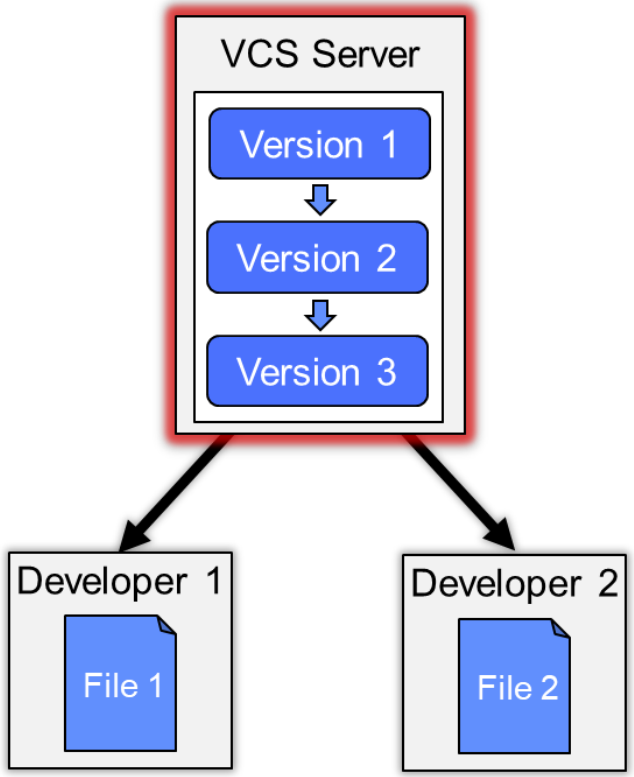
- CVCS (Centralized Version Control System)
- Client-Server Architektur
- Versionierung von vollständigen Projekten
- Zugriff auf zentralen Server über Netzwerk
- Populäre Umsetzungen
  - Concurrent Versions System (CVS), 1990 (seit 2008 keine Weiterentwicklung)
  - Apache Subversion (SVN), 2000



Version Control System

CVCS – zentraler Server

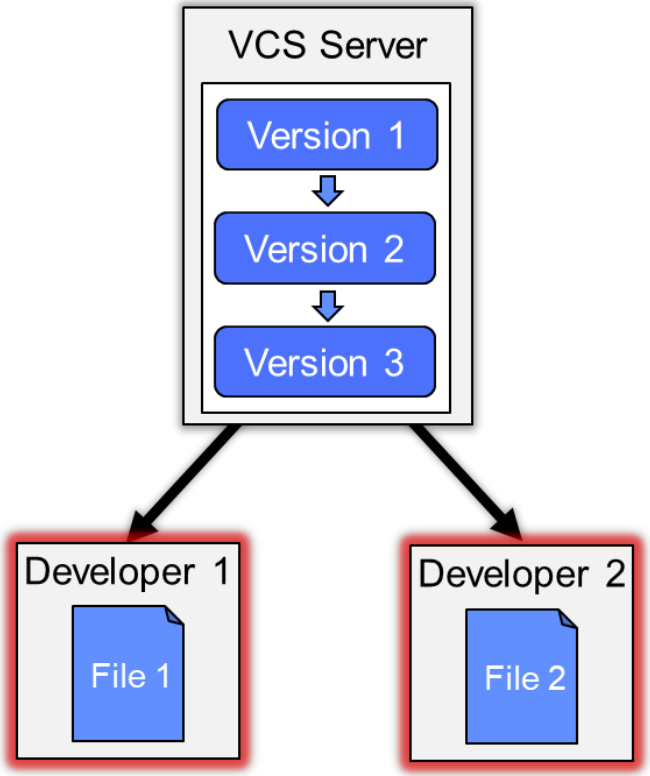
- Ermöglicht gemeinsame Arbeit an Projekten
- Speicherort der Versionshistorie
  - Single Source of Truth
  - Single Point of Failure
- Ermöglicht Zugriffkontrolle
  - Berechtigungen und Autorisierung



Version Control System

CVCS – Client

- Entwickler können Dateien zur Bearbeitung „auschecken“ und „einchecken“
- Benötigt Netzwerkverbindung

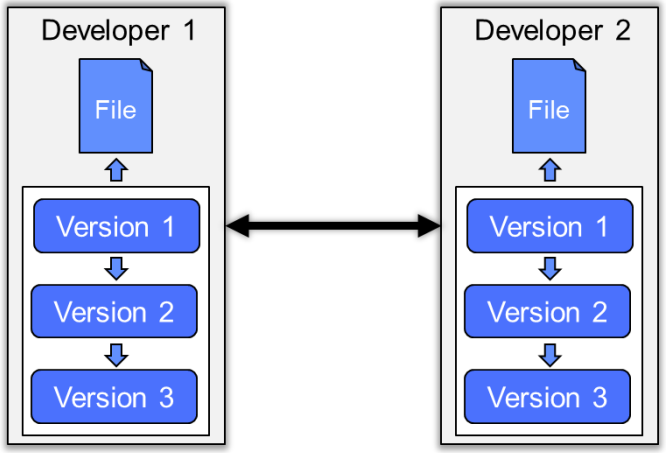


Version Control System

anderScore trust in competence

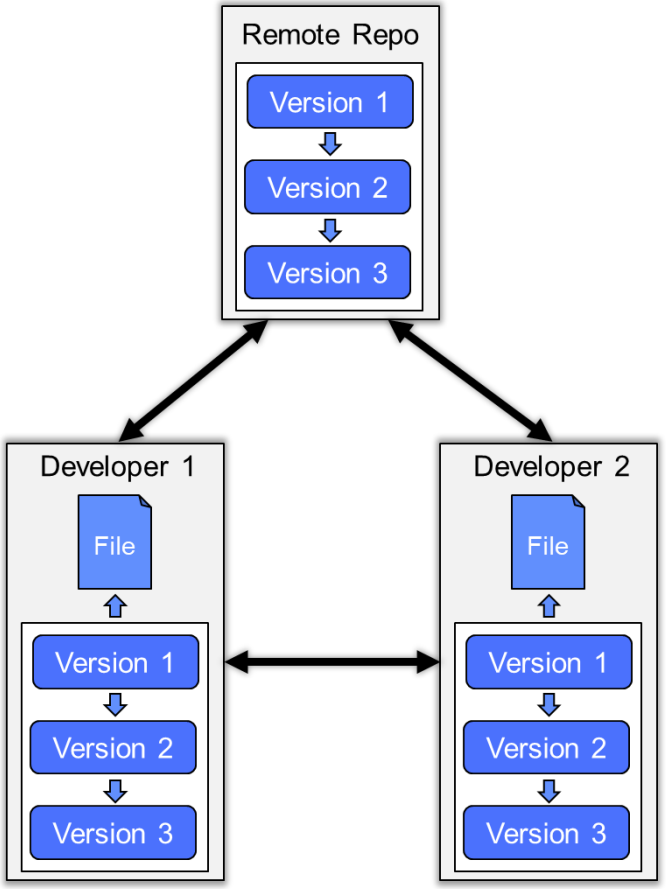
Dezentrale Versionsverwaltung

- DVCS (Distributed Version Control System)
- Kein zentrales Repository
  - Jeder Teilnehmer hat eigenes lokales Repository
- Populäre Umsetzungen
  - BitKeeper, 2000
  - Darcs, 2003
  - Mercurial, 2005
  - **Git, 2005**



DVCS – Remote Repository

- Oft existiert ein Remote Repository
  - Übernimmt einzelne Konzepte des Servers aus der CVCS
  - Für Betrieb des DVCS nicht unbedingt notwendig
- Vereinfacht Zusammenarbeit
- Ermöglicht zentrale Zugriffskontrolle



Einführung und

Konfiguration von Git

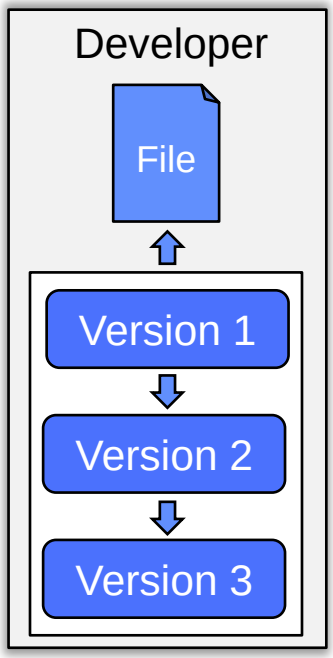
Git – Einführung

Was ist Git?

- Freie Software zu dezentralen Versionsverwaltung
- 2005 von Linus Torvalds entwickelt zur Nutzung beim Linux Kernel
- Heute die am weitesten verbreitete Versionsverwaltung
- Bedienung via Command Line
- Vielzahl von GUIs und
  IDE-Integrationen (Intellij, Eclipse uvm.)
- Website: https://git-scm.com/

Git – lokales Repository

- Git als DVCS auch ausschließlich lokal verwendbar
- Remote Repository und andere Mitwirkende entfallen



Installation von Git

- Aktuelle Version von Git prüfen

```
git -v
```

- Bei fehlender Installation Git installieren: https://git-scm.com/downloads

Konfiguration von Git

- Git bietet Vielzahl von Konfigurationsmöglichkeiten
- Beinhaltet Tool zur Konfiguration

```
git config
```

- Konfigurationsdateien können auf drei Ebenen gespeichert werden
  - **Systemkonfiguration**
    - /etc/gitconfig (Linux)
    - C:\ProgrammData\Git\config oder C:\Programm Files\Git\etc\gitconfig (Windows, Git Version 2.x oder neuer)
  - **Userkonfiguration**
    - ~/.gitconfig oder ~/.config/gitconfig (Linux)
    - C:\User\<User>\.gitconfig (Windows)
  - **Projektkonfiguration**
    - <Pfad zum Projekt>/.git/config
- Nachfolgende Level überschreiben vorherige

Git – Einführung

- Anzeigen der aktuellen Konfiguration mittels

```
git config --list --show-origin
```

- Identität konfigurieren
  - Notwendig, um mit Git arbeiten zu können
  - Informationen, die Git jedem Commit beifügt

```
git config --global user.name "<Name>"
git config --global user.email <Email>
```

Konzepte und

Grundlagen von Git

Git Projekte

- Sammlung von Dateien, die als Projekt zusammengehören und von Git versioniert werden
- Hauptordner wird auch als *Workspace* bezeichnet
- Neues Projekt anlegen

```
git init
```

- Unterordner **.git** wird angelegt, in welchem sich das *Repository* befindet

Beispiel: Anlegen eines Git Projektes

```
$ ls -la
total 12
drwxr-xr-x  3 gituser gituser 4096 May  9 22:32 .
drwxr-x--- 15 gituser gituser 4096 May  9 22:32 ..
-rw-r--r--  1 gituser gituser    0 May  9 22:30 file1.txt
-rw-r--r--  1 gituser gituser    0 May  9 22:30 file2.txt
drwxr-xr-x  2 gituser gituser 4096 May  9 22:31 subdir

$ git init
Initialized empty Git repository in /home/gituser/example/.git/

$ ls -la
total 16
drwxr-xr-x  4 gituser gituser 4096 May  9 22:32 .
drwxr-x--- 15 gituser gituser 4096 May  9 22:32 ..
drwxr-xr-x  7 gituser gituser 4096 May  9 22:32 .git
-rw-r--r--  1 gituser gituser    0 May  9 22:30 file1.txt
-rw-r--r--  1 gituser gituser    0 May  9 22:30 file2.txt
drwxr-xr-x  2 gituser gituser 4096 May  9 22:31 subdir
```

Status des Workspaces

- Status eines Workspaces anzeigen

```
git status
```

- Zeigt Informationen über Dateien im Workspace
- Beispiel

```
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt
        subdir/

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

- Dateien müssen explizit zur Versionierung mit Git hinzugefügt werden
- Sogenannte „Tracked Files“ werden auf Änderungen überwacht

Übungsaufgabe 1: Installation

1. Überprüfen Sie, ob eine aktuelle Version von Git auf Ihrem Rechner installiert ist mit dem Befehl `git -v`

```
$ git -v
git version 2.34.1
```

2. Falls keine aktuelle Version installiert ist, folgen Sie der Anleitung auf https://git-scm.com/book/de/v2/Erste-Schritte-Git-installieren für Ihr jeweiliges System
3. Falls noch nicht geschehen, setzen Sie mit dem folgenden Befehl Ihren globalen Namen sowie Ihre E-Mail

```
$ git config --global user.name "<Ihr Name>"
$ git config --global user.email "<Ihre E-Mail>"
```

Übungsaufgabe 2: Repository anlegen

1. Erstellen Sie einen neuen Ordner MyRepository und wechseln Sie in diesen.
2. Legen Sie im Ordner ein neues Git-Repository an mit dem Befehl `git init`
   - Beim ersten Anlegen eines Repositories zeigt Git einen Text bezüglich Benennung des Default Branches an
   - Möchten Sie den Default Branch global umbenennen, so können Sie dies mit dem Befehl `git config --global init.defaultBranch <Name>` machen
   - Im Folgenden wird der Default Branch als main bezeichnet
3. Schauen Sie sich über die Ausgabe von `git status` an, ob Ihr Repository erfolgreich angelegt wurde.

Lösung Übungsaufgabe 2: Repository anlegen

1.
```
$ mkdir MyRepository
$ cd MyRepository/
```

2.
```
$ git init
Initialized empty Git repository in /home/example/MyRepository/.git/
```

3.
```
$ git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Tracked Files

- Datei in die Git Versionierung aufnehmen

```
git add <file>
```

- Beispiel

```
$ git add */*.txt

$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt
        new file:   subdir/subdir_file1.txt
```

Tracking von Ordnern

- Git versioniert keine Ordner an sich
- Ordner werden indirekt über Dateiinformationen verwaltet und aufgenommen
- Dadurch können nur nicht-leere Ordner ins Repository aufgenommen werden

Beispiel: Kein Tracking von leeren Ordnern

```
$ mkdir empty_subdir

$ ls -la
total 20
drwxr-xr-x  5 gituser gituser 4096 May  9 23:07 .
drwxr-x--- 15 gituser gituser 4096 May  9 22:34 ..
drwxr-xr-x  7 gituser gituser 4096 May  9 23:06 .git
drwxr-xr-x  2 gituser gituser 4096 May  9 23:07 empty_subdir
-rw-r--r--  1 gituser gituser 0 May  9 22:30 file1.txt
-rw-r--r--  1 gituser gituser 0 May  9 22:30 file2.txt
drwxr-xr-x  2 gituser gituser 4096 May  9 22:54 subdir

$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt
        new file:   subdir/subdir_file1.txt
```

Beispiel: Explizites Tracking mit .gitkeep

```
$ touch empty_subdir/.gitkeep

$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt
        new file:   subdir/subdir_file1.txt

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

Commits

- Ein Commit speichert aktuellen Zustand zum Commit vorgemerkter Dateien im Repository (sog. *Snapshot*)
- Änderungen committen mittels

  `git commit`

- Beispiel:

```
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   subdir/subdir_file1.txt

$ git commit -m "Initial commit"
[main (root-commit) c61ef14] Initial commit
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
 create mode 100644 subdir/subdir_file1.txt
```

Commit Messages

- `git commit -m` oder `--message` um Beschreibung hinzuzufügen
- Vielzahl verschiedener Conventions und Guidelines
- Beispiele
  - https://www.baeldung.com/ops/git-commit-messages
  - https://www.gitkraken.com/learn/git/best-practices/git-commit-message
  - https://github.com/angular/angular/blob/main/CONTRIBUTING.md#commit
  - …
- Commit Message sollte kurze, aber aussagekräftige Auskunft geben

Inhalt eines Commits

- Änderungen innerhalb eines Commits sollten auf ein einzelnes Feature oder eine spezifische Änderung beschränkt sein
- Commits sollten nur Änderungen für jeweiliges Ziel beinhalten

Staging von Änderungen

- Änderungen an Dateien müssen für den nächsten Commit *gestaged* werden
- Datei stagen mittels

```
git add <file>
```

- Ohne Staging wird Datei beim nächsten Commit ignoriert

Beispiel:

```
$ echo test > file1.txt

$ git status
On branch main
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:   file1.txt

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

$ git add file1.txt

$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file1.txt

$ git commit -m "Update file1.txt"
[main e0b3cf3] Update file1.txt
 1 file changed, 1 insertion(+)
```

Übungsaufgabe 3: Erste Commits

1. Erstellen Sie eine leere Textdatei file1.txt.
2. Führen Sie den Befehl `git status` aus. Git zeigt Ihnen an, dass file1.txt aktuell nicht von Git versioniert wird.
   Lassen Sie Git diese Datei tracken.
3. Committen Sie nun Ihre hinzugefügte Datei mit der Commit-Nachricht „Added file1.txt file“.
4. Füllen Sie nun Ihre file1.txt mit dem Inhalt „file1 content“.
5. Führen Sie erneut den Befehl `git status` aus. Ihnen wird nun angezeigt, dass file1.txt modifiziert wurde.
   Fügen Sie file1.txt zur Staging Area hinzu.
6. Erstellen Sie einen zweiten Commit, um die Änderungen an file1.txt zu speichern. Wählen Sie dabei eine geeignete Commit-Message.

Lösung Übungsaufgabe 3: Erste Commits

1.
```
$ touch file1.txt
```

2.
```
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt

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

3.
```
$ git add file1.txt

$ git commit -m "Add file1.txt file"
[main (root-commit) 7aa4fee] Add file1.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
```

4.
```
$ echo "file1 content" > file1.txt
```

Lösung Übungsaufgabe 3: Erste Commits

5.
```
$ git status
On branch main
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:   file1.txt

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

$ git add file1.txt
```

6.
```
$ git commit -m "Add content to file1"
[main 1092ab2] Add content to file1
 1 file changed, 1 insertion(+)
```

Identifier von Commits

- Jeder Commit besitzt einen eindeutigen SHA-1 Identifier
- Commit IDs können über Log-Output angezeigt werden

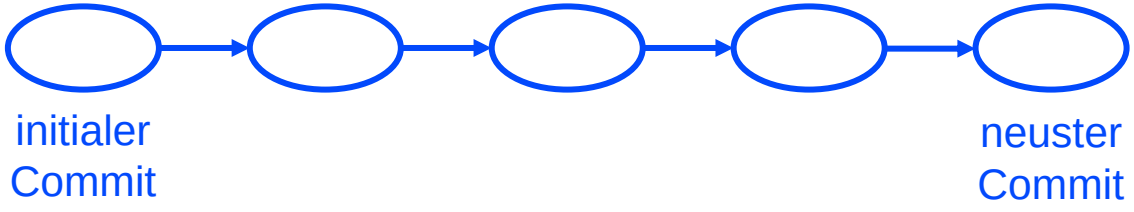
```
git log
```

- Beispiel:

```
$ git log
commit e0b3cf300585c6230d1abffafe37bb4b9d540247 (HEAD -> main)
Author: Git User <git.user@example.com>
Date:   Fri May 10 00:04:39 2024 +0200

    Update file1.txt

commit c61ef14e66353d2be1d7a20a3c1eaa73d78ffe71
Author:  Git User <git.user@example.com>
Date:   Thu May 9 23:40:52 2024 +0200

    Initial commit
```

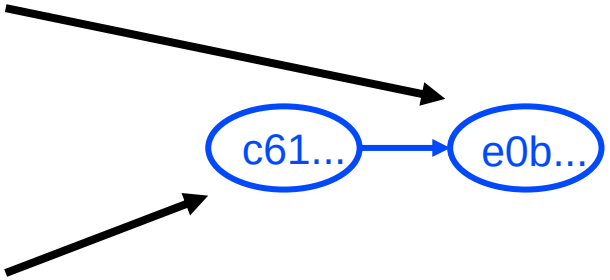
Visualisierung von Commits

- Abfolge von Commits häufig als gerichteter Graph dargestellt
- Knoten entspricht dabei einem Commit



- Beispiel:

```
$ git log
commit e0b3cf300585c6230d1abffafe37bb4b9d540247
(HEAD -> main)
Author: Git User <git.user@example.com>
Date:   Fri May 10 00:04:39 2024 +0200

    Update file1.txt

commit c61ef14e66353d2be1d7a20a3c1eaa73d78ffe71
Author:  Git User <git.user@example.com>
Date:   Thu May 9 23:40:52 2024 +0200

    Initial commit
```



Weitere Git Befehle

`git commit <files>`	Commitet selektiv ausgewählte Dateien, umgeht Staging
`git diff`	Zeigt Unterschiede der aktuellen Dateiinhalte im Vergleich zum letzten Commit
`git diff --staged`	Wie `git diff`, jedoch werden nur gestagete Dateien betrachtet
`git rm`	Entfernt Datei aus Workspace und staged Löschung aus Git für nächsten Commit
`git mv`	Verschiebt Dateien im Workspace, Nutzung auch zum Umbenennen

Git – Grundlagen

`git reset HEAD`	Unstaged alle aktuell gestageten Dateien, Änderungen bleiben erhalten
`git checkout --`	Zurücksetzen aller getrackten Dateien auf den Stand des letzten Commits
`git checkout -- <files>`	Zurücksetzen ausgewählter Dateien auf den Stand des letzten Commits
`git restore --`	Zurücksetzen aller Dateien auf den Stand des letzten Stagings
`git restore --staged` `git restore --staged <files>`	Äquivalent zu `git checkout bzw. git checkout -- <files>`

.gitignore Datei

- Definiert von Git zu ignorierende Verzeichnisse oder Dateien
- Auf verschiedenen Ebenen definiert
  - Globale .gitignore in der Git-Konfiguration
  - Respository-spezifische .gitignore
  - Verzeichnis-spezifische .gitignore
- Verzeichnisse/Dateien können über Pattern angegeben werden
- Beispielinhalt:

```
# class files
*.class

# log files
*.log

# jar files
*.jar

# build and out dir
**/build
**/out
```

Übungsaufgabe 4: Umbenennen von Dateien

1. Erstellen Sie eine Datei new_file.txt mit dem Inhalt „file2 content“, lassen Sie diese durch Git tracken und committen Sie diese.
2. Benennen Sie die Datei new_file.txt über Ihr Betriebssystem in file2.txt um (Beispielsweise `mv new_file.txt file2.txt` unter Linux).
3. Führen Sie den Befehl `git status` aus. Git sollte Ihnen anzeigen, dass new_file.txt gelöscht wurde und file2.txt als untracked file zum Repository hinzugefügt wurde, was nicht direkt unsere Änderungen widerspiegelt.
4. Machen Sie die vorherige Umbenennung rückgängig und führen Sie diese erneut mit dem Befehl `git mv` aus.
   `git status` sollte Ihnen nun anzeigen, dass new_file.txt in file2.txt umbenannt wurde. Committen Sie die Änderungen.

Lösung Übungsaufgabe 4: Umbenennen von Dateien

1.
```
$ echo "file2 content" > new_file.txt

$ git add new_file.txt

$ git commit -m "Add new_file.txt file"
[main d2d6d1a] Add new_file.txt file
 1 file changed, 1 insertion(+)
 create mode 100644 new_file.txt
```

2.
```
$ mv new_file.txt file2.txt
```

3.
```
$ git status
On branch main
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    new_file.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2.txt

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

Lösung Übungsaufgabe 4: Umbenennen von Dateien

4.
```
$ mv file2.txt new_file.txt

$ git mv new_file.txt file2.txt

$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    new_file.txt -> file2.txt

$ git commit -m "Rename new_file.txt to file2.txt"
[main 86561fb] Rename new_file.txt to file2.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename new_file.txt => file2.txt (100%)
```

Übungsaufgabe 5: .gitignore

In vielen Softwareprojekten existieren Ordner wie build/ oder bin/, welche typischerweise beim Bauen bzw. beim Kompilieren von Code erzeugt werden. Diese enthalten meist plattformspezifische Binaries und werden in aller Regel nicht über Git versioniert.

1. Erstellen Sie die beiden Ordner build und bin und legen Sie in beiden Ordnern eine Datei output.class an.
2. Führen Sie den Befehl `git status` aus. Git sollte Ihnen die neuen Ordner und Dateien als untracked anzeigen.
3. Erstellen Sie im Hauptordner eine Datei .gitignore und tragen Sie in diese den folgenden Inhalt ein.

```
**/build/
**/bin/
*.class
```

Übungsaufgabe 5: .gitignore

Durch die Einträge in der .gitignore werden alle build und bin Ordner, sowie alle .class Dateien im gesamten Projekt ignoriert.

4. Committen Sie die .gitignore Datei. Achten Sie darauf, dass die Dateien im bin bzw. build Ordner nicht von Git getracked werden.
5. Nun sollten in der Ausgabe von `git status` weder der build noch der bin Ordner und deren Dateien als untracked aufgeführt werden.
6. Legen Sie im Ordner build eine neue Datei test.txt an. Auch diese sollte nicht durch Git getracked und damit bei `git status` auch nicht aufgeführt werden.

Lösung Übungsaufgabe 5: .gitignore

1.
```
$ mkdir bin
$ mkdir build
$ touch bin/output.class
$ touch build/output.class
```

2.
```
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        bin/
        build/

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

3.
```
$ nano .gitignore
```

```
**/build/
**/bin/
*.class
```

Lösung Übungsaufgabe 5: .gitignore

4.
```
$ git add .gitignore

$ git commit -m "Add .gitignore"
[main f252eb9] Add .gitignore
 1 file changed, 3 insertions(+)
 create mode 100644 .gitignore
```

5.
```
$ git status
On branch main
nothing to commit, working tree clean
```

6.
```
$ touch build/test.txt

$ git status
On branch main
nothing to commit, working tree clean
```

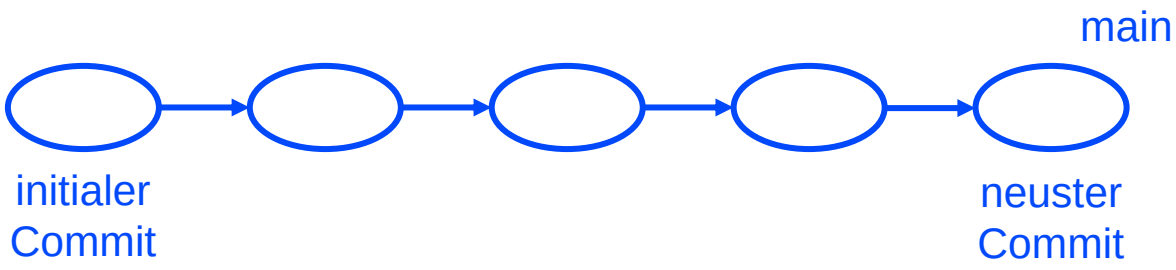
Git

Branches

Git – Branches

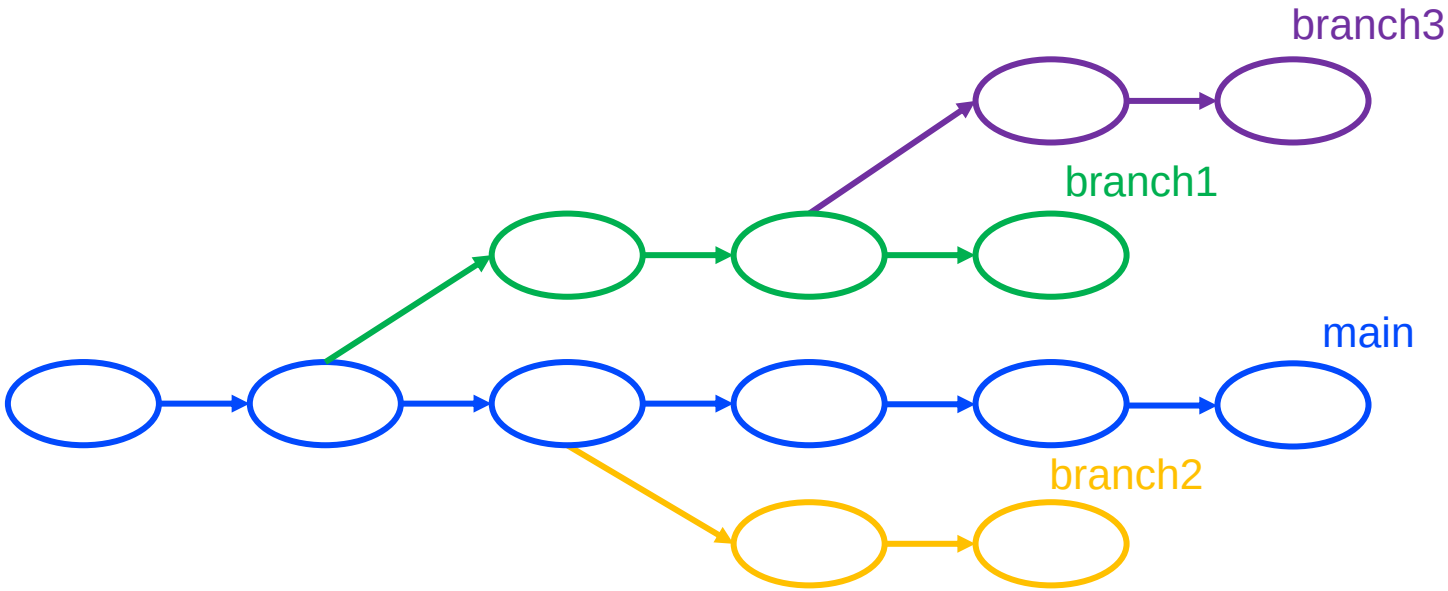
Was sind Branches?

- Entwicklungszweige bestehend aus einer Abfolge einzelner Commits
- Git arbeitet immer in Branches
- Initialisierung eines Repositorys legt automatisch *main* Branch an
  - Bis 2020 *master,* aufgrund von offensive Language umbenannt
  - Standardname konfigurierbar mittels
    `git config --global init.defaultBranch <branch-name>`
- Visuelle Darstellung als gerichteter, azyklischer Graph



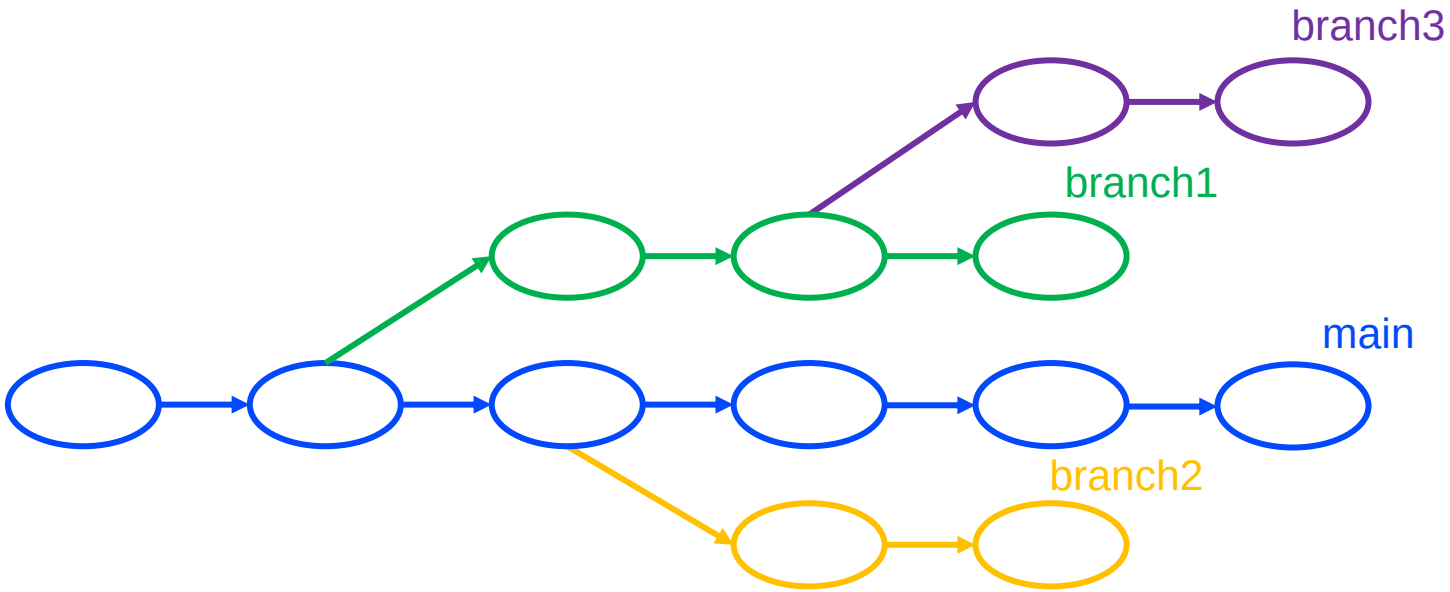
Git – Branches

- Mehrere Branches ermöglichen paralleles Arbeiten
  - Funktionale Komponenten
  - Bugfixes
  - Hotfixes für spezielle Versionen
  - Versionierung (bspw. Separater Branch für jedes Release/Version)
- Abzweigung von jedem Commit aus möglich
- Beispielgraph:



Git – Branches

- Viele Möglichkeiten und Ansätze, um Branches zu verwenden
- Aufbau und Nutzung stark von individuellen Bedürfnissen abhängig
- Werden im Rahmen von Workflows genauer besprochen



Git – Branches

Grundlegende Befehle für Branches

- Branches des aktuellen Projektes anzeigen

```
git branch
```

- Beispiel

```
$ git branch
  feature1
  feature2
  hotfix1
* Main
```

- Aktiver Branch wird mit Stern und ggf. farblich markiert

Git – Branches

Weitere `git branch` Befehle

`git branch <branch-name>`	Erzeugt neuen Branch ausgehend vom letzten Commit des aktiven Branches
`git branch <branch-name> <source-branch>`	Erzeugt neuen Branch ausgehend vom letzten Commit des angegebenen Source Branches
`git branch <branch-name> <commit-id>`	Erzeugt neuen Branch ausgehend vom spezifizierten Commit

Git – Branches

`git checkout <branch-name>`	Wechselt den aktiven Branch zum angegeben Branch
`git checkout -b <branch-name>`	Erstellt angegebenen Branch, falls dieser nicht existiert und setzt ihn als aktiven Branch
`git switch <branch-name>`	Äquivalent zu `git checkout <branch-name>`
`git switch -c <branch-name>`	Äquivalent zu `git checkout -b <branch-name>`
`git log --graph --all`	Ausgabe einer grafischen Repräsentation der Branches auf der Konsole

Git – Branches

switch / restore

- `git switch` und `git restore` sind aus Befehl `git checkout` entstanden
- Ziel: Mehrdeutigkeit von `git checkout` auflösen
  - `git checkout <branch>` ➔ Wechsel des Branches
  - `git checkout --` ➔ Zurücksetzen von Änderungen
- `git checkout` trotzdem immer noch weit verbreitet

Git – Branches

Beispiel: neuen Branch anlegen und direkt wechseln

```
$ git checkout -b feature3
Switched to a new branch 'feature3'

$ git branch
  feature1
  feature2
* feature3
  hotfix
  main

$ git status
On branch feature3
nothing to commit, working tree clean
```

Git – Branches

Aktiver Branch

- Immer ein aktiver Branch im lokalen Workspace
- Alle Commits werden auf dem aktiven Branch ausgeführt
- Wechsel des Branches ändert Inhalt sämtlicher Dateien im Workspace auf den Stand des Zielbranches
- Anzeige des aktiven Branches

  `git status` oder über `git branch`
- Beispiel

```
$ git status
On branch main
nothing to commit, working tree clean
```

Git – Branches

Wechseln zwischen Branches

- Wechseln von Branches mit gestageten Änderungen oder veränderten getrackten Dateien mitunter problematisch
- Änderungen auf Zielbranch würden ggf. verloren gehen
- Git warnt Nutzer, sollten Änderungen verloren gehen
- Ausnahme: Existiert Änderung nur im Ausgangsbranch, kann der Branch gewechselt werden

Git – Branches

Beispiel:

```
$ git checkout feature1
error: Your local changes to the following files would be overwritten by checkout:
        shared_file.txt
Please commit your changes or stash them before you switch branches.
Aborting
```

- Mögliche Optionen
  - Änderungen committen
  - Änderungen verwerfen und Checkout mittels `git checkout -f <branch>` (`-f` oder `--force`) erzwingen
  - Änderungen in *Stashing Area* zwischenspeichern und später wiederherstellen mittels `git stash`

Git – Branches

HEAD

- Spezielle Referenz, die auf letzten Commit des aktiven Branches verweist
- Beispiel:

```
$ git log
commit e0b3cf300585c6230d1abffafe37bb4b9d540247 (HEAD -> main)
Author: Git User <git.user@example.com>
Date:   Fri May 10 00:04:39 2024 +0200

    Update file1.txt
...
```

- Bei neuen Commits wandert die HEAD Referenz weiter

Detached HEAD

- HEAD verweist nicht auf letzten Commit des aktiven Branches
- Auschecken einzelner Commits

```
$ git checkout c61ef14
Note: switching to 'c61ef14'.

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 c61ef14 Initial commit
```

Git – Branches

Übungsaufgabe 6: Branches

1. Erstellen Sie im Projekt einen Ordner features.
2. Erstellen Sie einen neuen Branch feature1 und wechseln Sie in diesen.
3. Legen Sie im Ordner features eine Datei feature1_file1.txt mit beliebigem Inhalt an und committen Sie Ihre Änderungen.
4. Erstellen Sie eine zweite Datei feature1_file2.txt im Ordner an und committen Sie auch diese.
5. Wechseln Sie zurück auf den main Branch. Da Git den Workspace an den aktiven Branch anpasst, sollte der Ordner features hier leer bzw. nicht vorhanden sein.

Lösung Übungsaufgabe 6: Branches

1.
```
$ mkdir features
```

2.
```
$ git checkout -b feature1
Switched to a new branch 'feature1'
```

3.
```
$ echo "feature1 file1 content" > features/feature1_file1.txt

$ git add features/feature1_file1.txt

$ git commit -m "Add feature1_file1.txt"
[feature1 0fd3371] Add feature1_file1.txt
 1 file changed, 1 insertion(+)
 create mode 100644 features/feature1_file1.txt
```

4.
```
$ echo "feature1 file2 content" > features/feature1_file2.txt

$ git add features/feature1_file2.txt

$ git commit -m "Add feature1_file2.txt"
[feature1 6c2b85a] Add feature1_file2.txt
 1 file changed, 1 insertion(+)
 create mode 100644 features/feature1_file2.txt
```

Lösung Übungsaufgabe 6: Branches

5.
```
$ git checkout main
Switched to branch 'main'

$ ls
bin  build  file1.txt  file2.txt
```

Übungsaufgabe 7: Branches

Branches können nicht nur vom letzten Commit des aktuellen Branch abzweigen, sondern von jedem beliebigen Commit.

1. Lassen Sie sich über `git log` die Commit-IDs ausgeben und erstellen Sie einen Branch feature2 ausgehend vom zweiten Commit des main Branches.
2. Legen Sie erneut den Ordner features, sowie eine Datei feature2_file1.txt an und committen Sie Ihre Änderungen.
3. Wechseln Sie zurück auf den main Branch. Der features Ordner ist hier immer noch leer bzw. nicht vorhanden.

Lösung Übungsaufgabe 7: Branches

1.
```
$ git log --oneline
f252eb9 (HEAD -> main) Add .gitignore
86561fb Rename new_file.txt to file2.txt
d2d6d1a Add new_file.txt file
1092ab2 Add content to file1
7aa4fee Add file1.txt file

$ git checkout -b feature2 1092ab2
Switched to a new branch 'feature2'
```

2.
```
$ mkdir features

$ echo "feature2 file1 content" > features/feature2_file1.txt

$ git add features/feature2_file1.txt

$ git commit -m "Add feature2_file1.txt"
[feature2 a8c94d1] Add feature2_file1.txt
 1 file changed, 1 insertion(+)
 create mode 100644 features/feature2_file1.txt
```

Lösung Übungsaufgabe 7: Branches

3.
```
$ git checkout main
Switched to branch 'main'

$ ls
bin  build  file1.txt  file2.txt
```

Git

Reset & Revert

Git – Reset

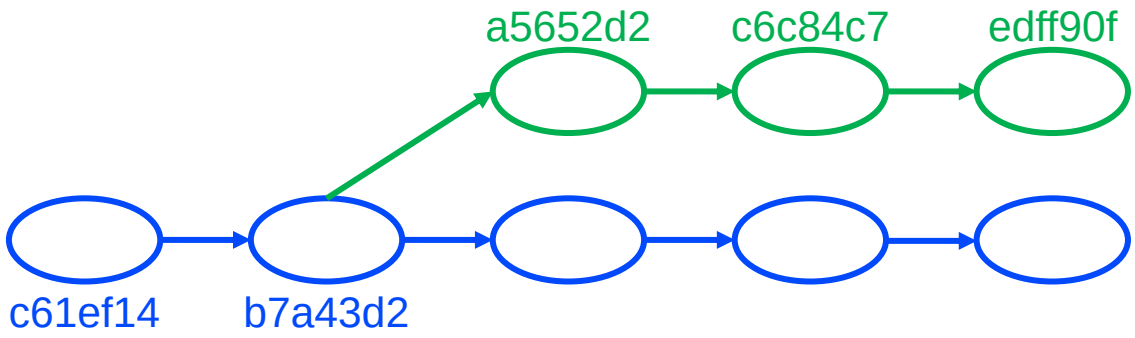
Reset

```
git reset [<mode>] [<commit>]
```

- Setzt den HEAD auf angegebenen Commit
- Default ist letzter Commit
- Veränderung der Commit Historie möglich
- Commits werden nicht gelöscht

Git – Reset

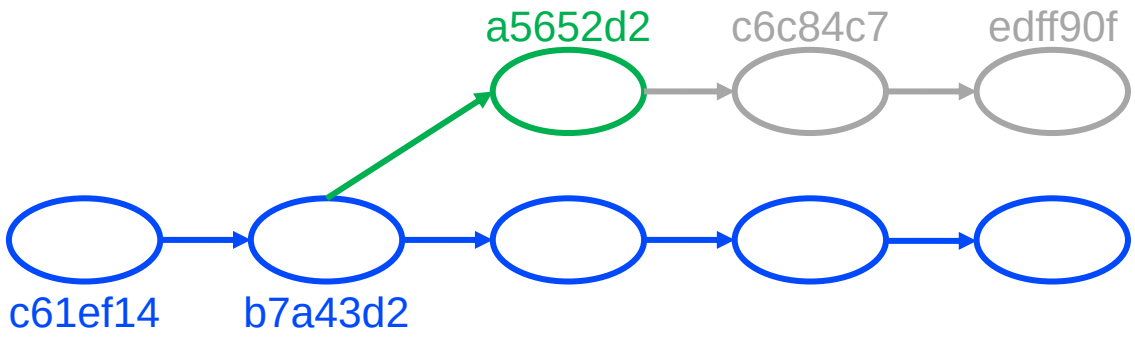
Beispiel:




```
$ git log --oneline
edff90f (HEAD -> feature1) Add feature_file3
c6c84c7 Add feature_file2
a5652d2 Add feature_file1
b7a43d2 Add main_file1
c61ef14 Initial commit
```

Git – Reset

Beispiel:




```
$ git reset a5652d2

$ git log --oneline
a5652d2 (HEAD -> feature1) Add feature_file1
c61ef14 Initial commit
```

Git – Reset

Reset Modi

```
git reset [<mode>] [<commit>]
```

- `--soft`
  - Resettet nur die Commit Historie
  - Staging Bereich und Workspace bleiben unverändert
- `--mixed` (Default)
  - Resettet die Commit Historie und den Staging Bereich
- `--hard`
  - Resettet Commit Historie, Staging Area und Workspace

Git – Revert

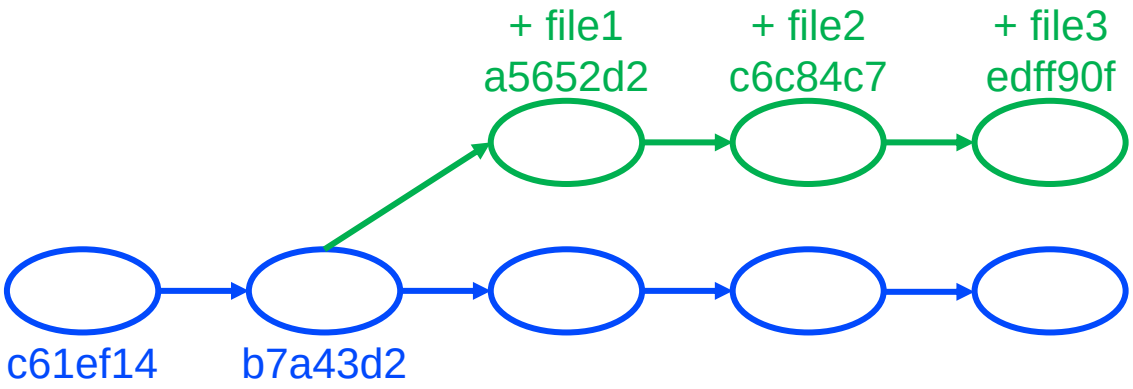
Revert

```
git revert <commit>…
```

- Macht Änderungen der angegebenen Commits rückgängig
- Nutzt für jeden zurückgesetzten Commit neuen Commit
- Verändert zurückliegende Commit Historie nicht
- Workspace darf keine uncommitteten Änderungen beinhalten

Git – Revert

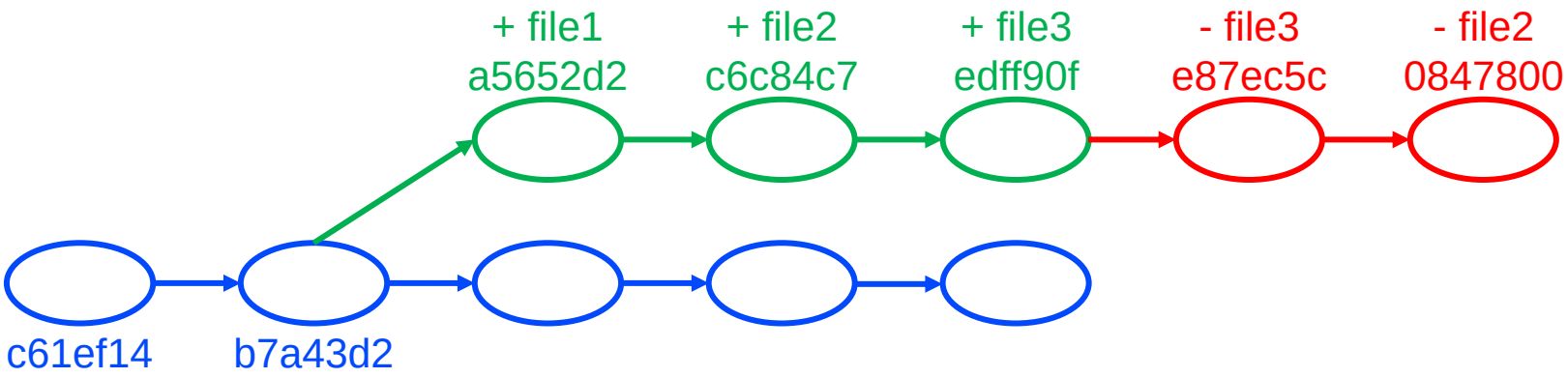
Beispiel:




```
$ git log --oneline
edff90f (HEAD -> feature1) Add feature_file3
c6c84c7 Add feature_file2
a5652d2 Add feature_file1
b7a43d2 Add main_file1
c61ef14 Initial commit
```

Git – Revert

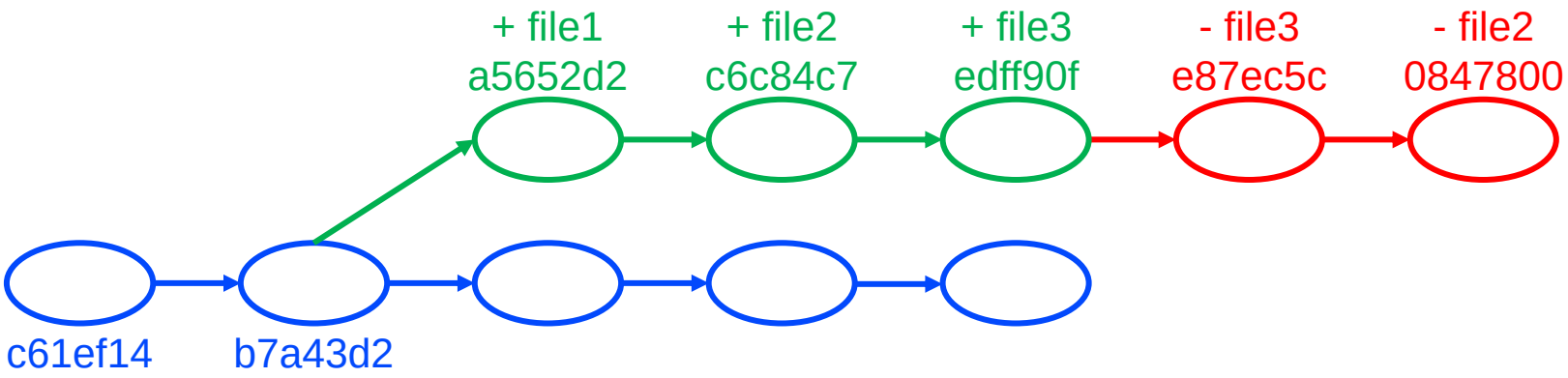
Beispiel:




```
$ git revert edff90f c6c84c7
[feature1 e87ec5c] Revert "Add feature_file3"
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 feature_file3.txt
[feature1 0847800] Revert "Add feature_file2"
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 feature_file2.txt
```

Git – Revert

Beispiel:




```
$ git log --oneline
0847800 (HEAD -> feature1) Revert "Add feature_file2"
e87ec5c Revert "Add feature_file3"
edff90f Add feature_file3
c6c84c7 Add feature_file2
a5652d2 Add feature_file1
c61ef14 Initial commit
```