Advanced Git
Seminar: Effiziente Programmierung

Kevin Kwasny

Arbeitsbereich Wissenschaftliches Rechnen Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg
11.02.2021

Gliederung (Agenda)

- Motivation
- Version Control System
- Datenerwaltung in Git
- Git commands
- Zusammenfassung
- Literatur

Motivation

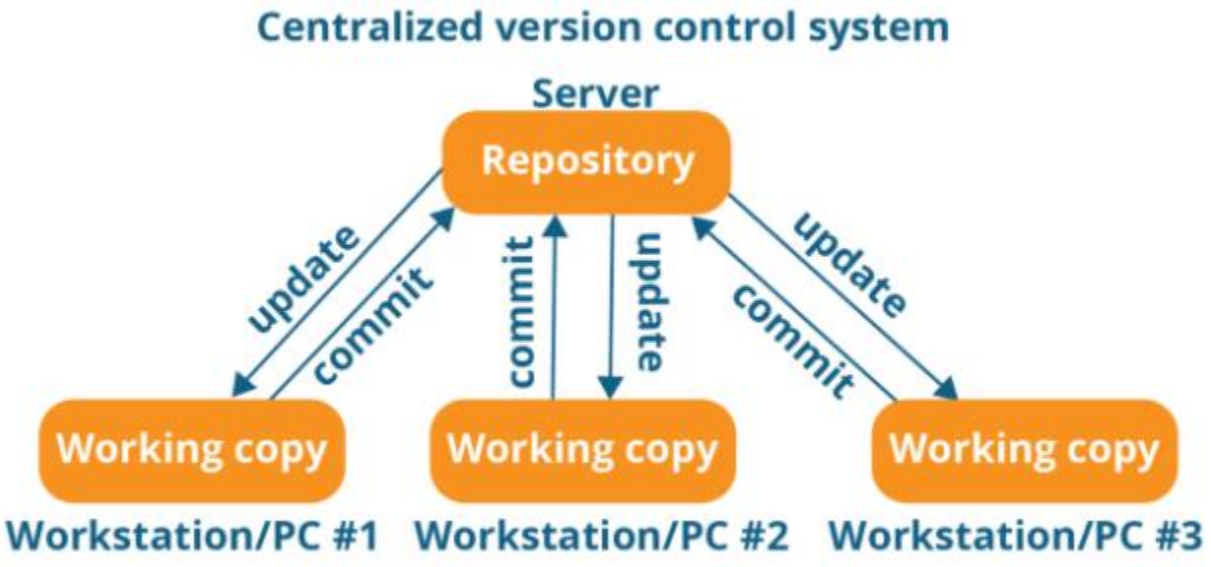
- Versions Kontrolle fundamentales Problem der Software Entwicklung
- Änderungen am Projekt direkt erkennbar
- Änderungen werden Automatisiert zusammengefasst
- Betriebtssystem & Programmiersprach unabhängig (C, Java usw.)

Version **C**ontrol **S**ystem

- Erfassung von Änderungen / Protokollierung
- Archivierung von Versionen
- Wiederherstellung
- Koordinierung auf gemeinsamen Zugriff von mehreren
- Grob Aufteilung in: Centralized und Distributed

Centralized **VCS**

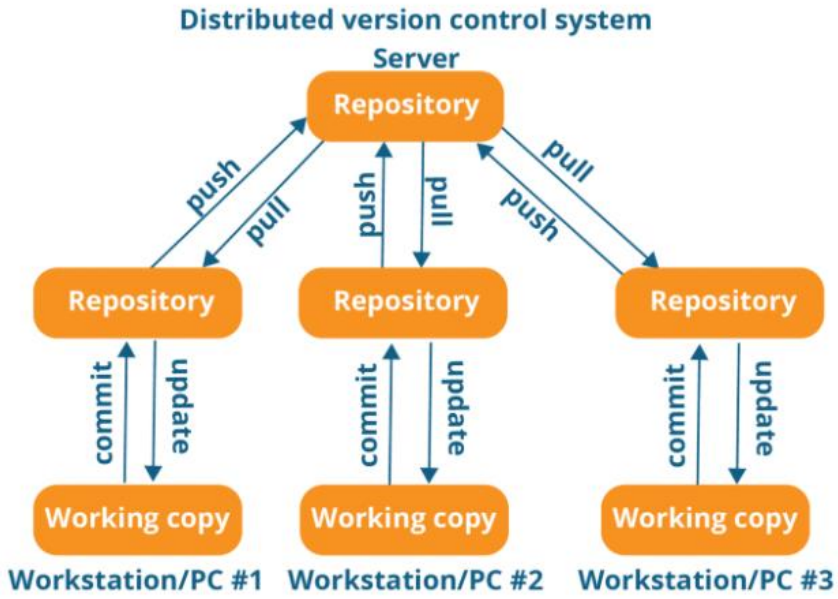
- Client-Server-System
- Archivierung ist Server lastig



Source: https://medium.com/faun/centralized-vs-distributed-version-control-systems-a135091299f0

Distributed **VCS** (DVCS)

- Lokales repository, incl. Historie usw.
- Keine dauer Verbindung mit dem Network
- Mehr Speicher verbrauch
- Branching und Merging ist einfacher



Source: https://medium.com/faun/centralized-vs-distributed-version-control-systems-a135091299f0

Centralized vs. Distributed

Distributed Vorteile:

- Schnellerer zugriff auf alte Daten
- Zusammenfassung von commits
- Connection freies wechseln an Versionen
- Leichteres Austauschen an kleinen Änderungen

Distributed Nachteile:

- Mehr Speicherbedarf
- Große Historie (+50.000) dauert zum kompletten runterladen

Fazit: alles was ein Centralized VCS kann auch ein DVCS!

Datenverwaltung in Git

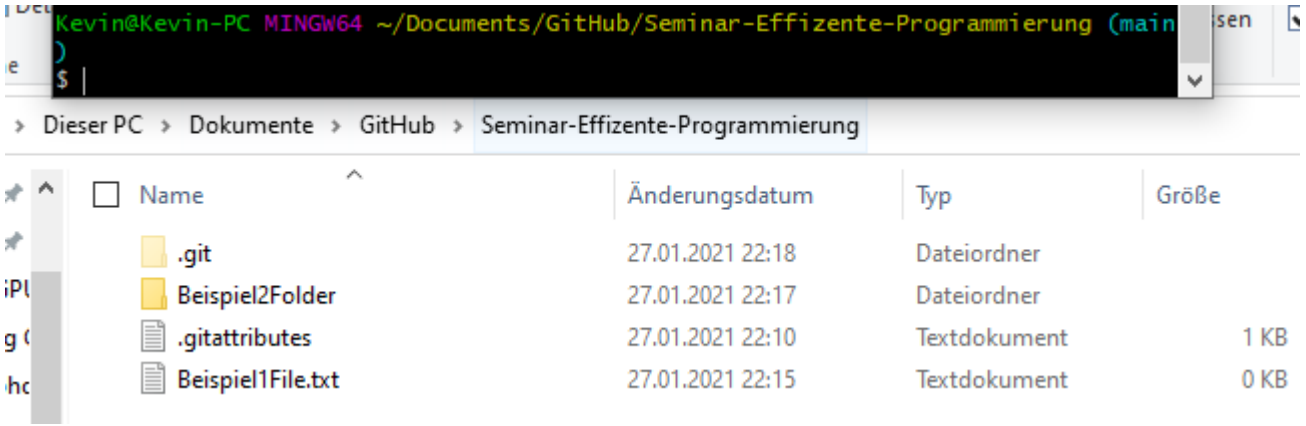
- Objekte
- Pointer (HEAD und Branches)
- The big picture

Objekte

- Eindeutige **O**bjektiv **ID** (**OID**)

  SHA1 hash
- Size
- Bestehen meistens aus Text
- Drei Objekt-Typen: blob, tree und commit
- Commands um OID zu bekommen und Infos der Objekte:
  `$ git rev-parse <ref>` und
  `$ git cat-file <oid>`

Objekte - Beispiel




```
.git/objects/a4/abdc73d873bb7c95ad097d5744ff15e45336ba
.git/objects/b0
.git/objects/b0/75073304c60c025bf4c8d49f1a13799c422eb1
.git/objects/c0
```

blob

- Sind Objekte!
- Inhalte der ‚File‘
- Inhaltlich Identische Files zeigen auf ein blob!

blob - Beispiel

OID?

```
Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git rev-parse HEAD:Beispiel1File.txt
b075073304c60c025bf4c8d49f1a13799c422eb1
```

Objekt Typ?

```
Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git cat-file -t b075
blob
```

Objekt Inhalt?

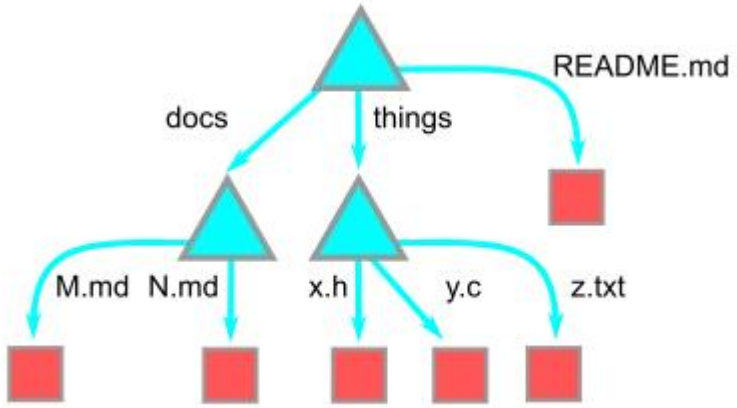
```
Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git cat-file -p b075
Beispiel1File innhalt
```

Git status?

```
Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git status
On branch main
nothing to commit, working tree clean

Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ 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:   Beispiel1File.txt

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

tree

- Sind Objekte!
- Verzeichnisauflistung
- Beinhalten: Objekt Typ, Name des Objektes, OID
- Änderung eines Objektes innerhalb des tree
  => neue OID für das Objekt im tree
  => neue OID für den tree
  => selektiertes erkennen geänderter Objekte



https://github.blog/2020-12-17-commits-are-snapshots-not-diffs/

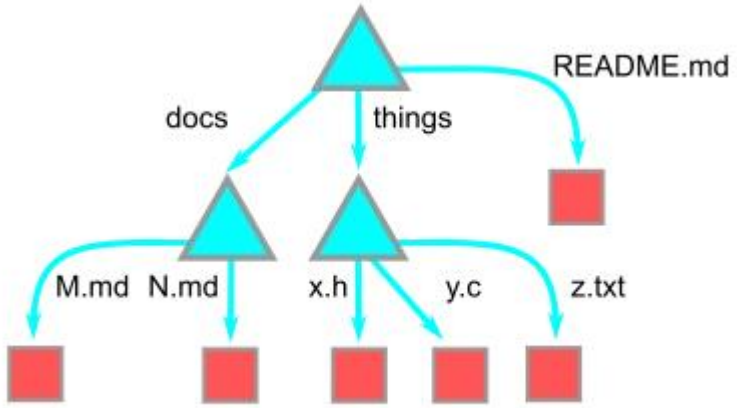
tree - Beispiel

Beispiel2Folder: OID, Typ, Inhalt?

```
Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git rev-parse HEAD:Beispiel2Folder
0f5694a9f6c848826ed12f7e64eddf2e73994198

Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git cat-file -t 0f5694
tree

Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git cat-file -p 0f5694
100644 blob 2998e29a5d7e645932e33d92f3475547d6246f39    Beispiel2File.txt
```

Root directory: OID, Typ, Inhalt?

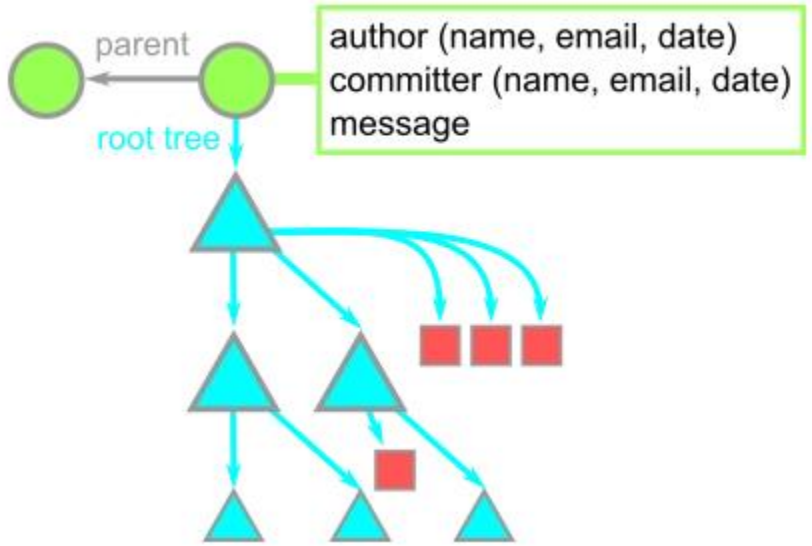
```
Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git rev-parse HEAD:
c0858614bfd6b8a3bfae8b2b73afc7180698d3cf

Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git cat-file -t c08586
tree

Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git cat-file -p c08586
100644 blob dfe0770424b2a19faf507a501ebfc23be8f54e7b    .gitattributes
100644 blob b075073304c60c025bf4c8d49f1a13799c422eb1    Beispiel1File.txt
040000 tree 0f5694a9f6c848826ed12f7e64eddf2e73994198    Beispiel2Folder
```



https://github.blog/2020-12-17-commits-are-snapshots-not-diffs/

commit

- Sind Objekte!
- Schnappschuss (Snapshot)
- Beinhalten: tree, parent, Autor, committer, Kommentar

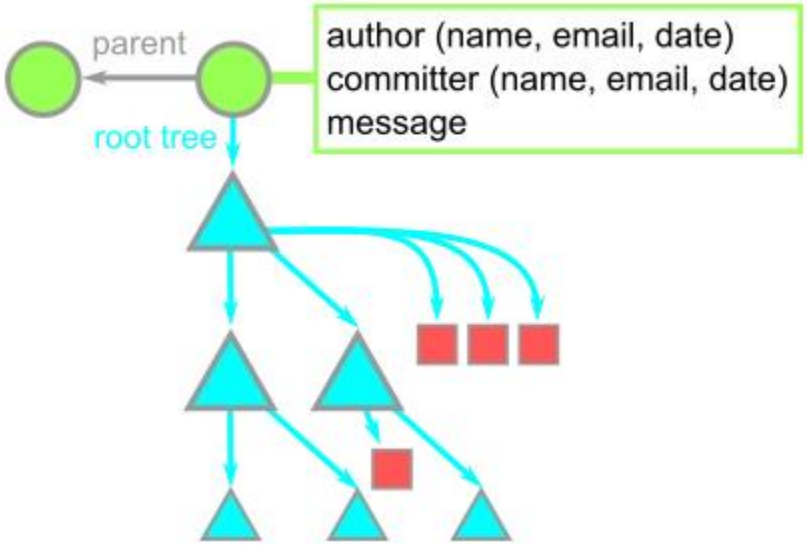


https://github.blog/2020-12-17-commits-are-snapshots-not-diffs/

Commit - Beispiel

- OID, Typ, Inhalt?

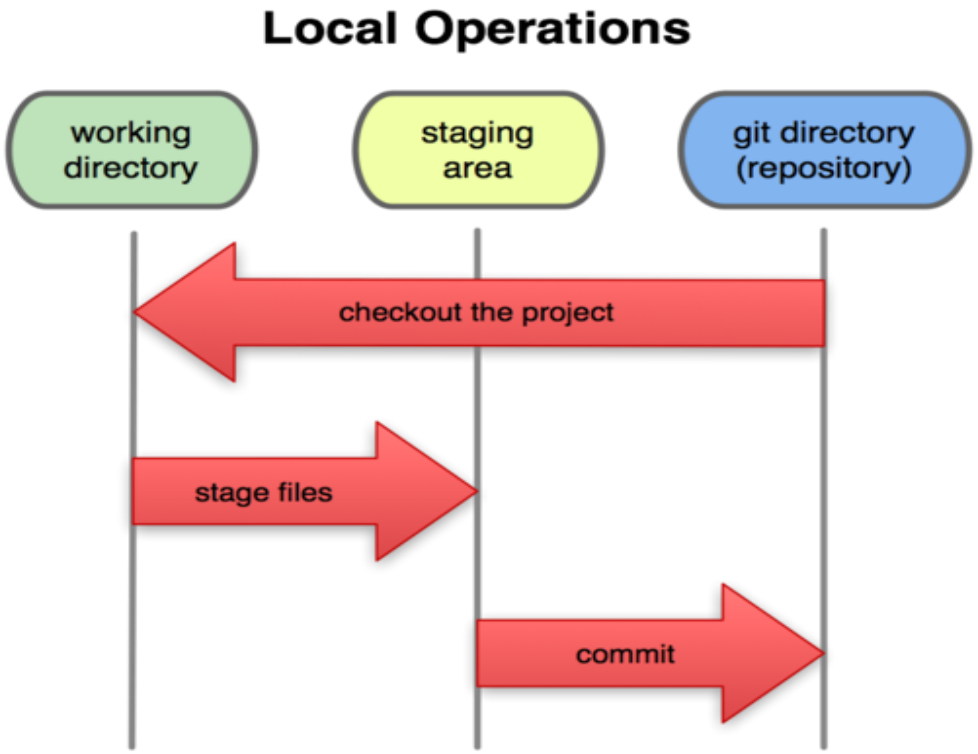
```
Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git rev-parse HEAD
1d8595744a28d96ca804f3e6a23536ed3bd51c74

Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git cat-file -t 1d8595744
commit

Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (main)
$ git cat-file -p 1d8595744
tree c0858614bfd6b8a3bfae8b2b73afc7180698d3cf
parent a26d0ea773ef05002bfcd3b133868c5e624dbde5
author Rumil93 <74466034+Rumil93@users.noreply.github.com> 1611782700 +0100
committer Rumil93 <74466034+Rumil93@users.noreply.github.com> 1611782700 +0100

File innhalt erstellt

Beiden Beispiel Files einen innhalt gegeben (name der file + 'innhalt')
```



https://github.blog/2020-12-17-commits-are-snapshots-not-diffs/

Local Operations

- Working directory ist lokaler Rechner
- Staging area ist puffer
- Git repository ist der letzte commit
- Eine File kann in allen drei Locations unterschiedlich sein!



https://www.chromium.org/developers/fast-intro-to-git-internals

Branches

- Sind keine Objekte
- Branches sind Pointer/Referenzen auf commits!
- Branches sind für uns, um Änderungen zu verfolgen und zu teilen

Spezieller HEAD Pointer

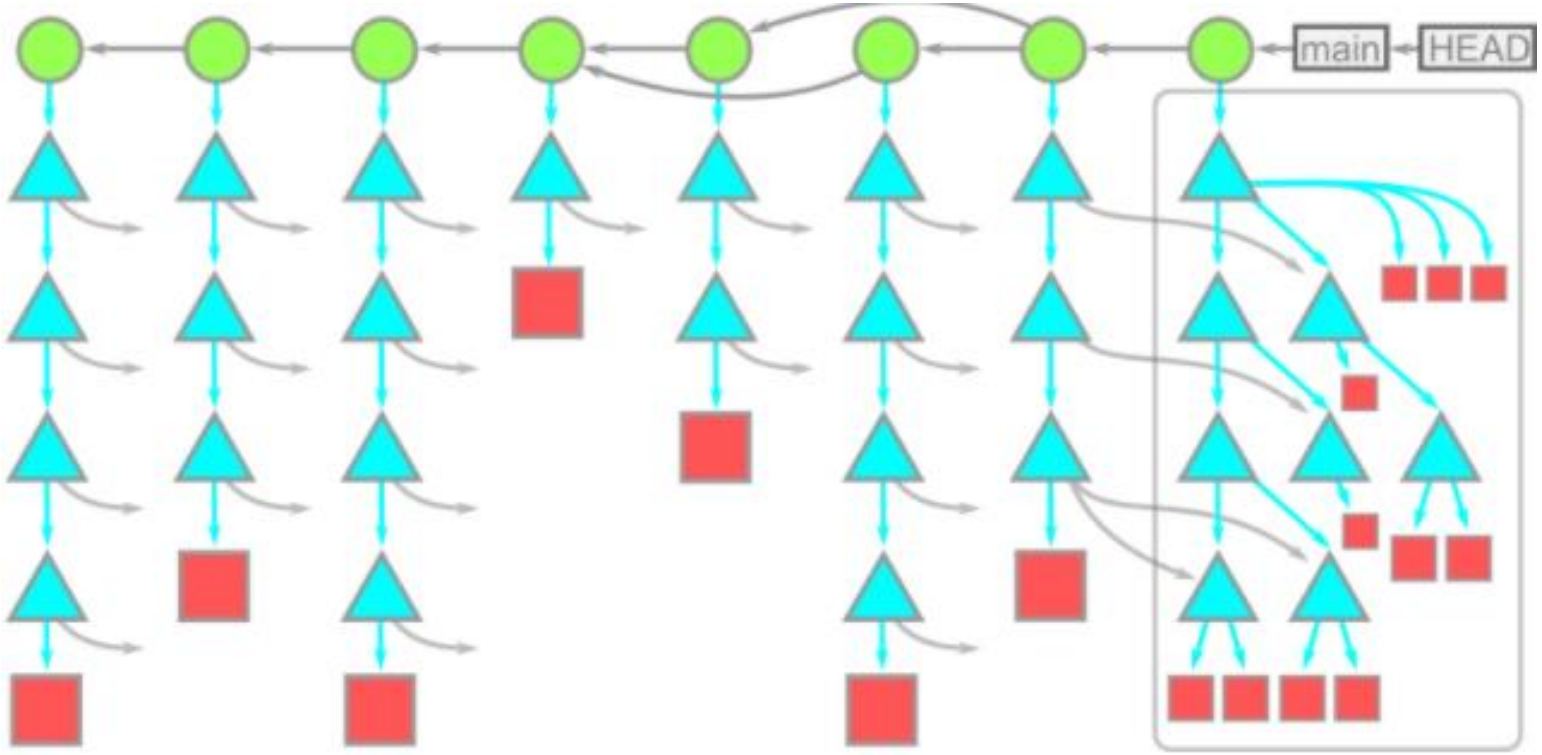
- Zeigt auf gerade ausgewählten branch
- Ändert unsere Directory zum entsprechenden Sichtfeld

Branches - Beispiel

```
Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (branch_commit_1)
$ git branch
* branch_commit_1
  branch_innhalt
  branch_innhalt_beginning
  main

Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (branch_commit_1)
$ git branch erstelle_neues_branch_für_branch_beispiel

Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (branch_commit_1)
$ git branch
* branch_commit_1
  branch_innhalt
  branch_innhalt_beginning
  erstelle_neues_branch_für_branch_beispiel
  main
```

The big picture

- Main commit ist vollständig
- Snapshot View



https://github.blog/2020-12-17-commits-are-snapshots-not-diffs/

Git Commands

- Navigieren durch Git
  ->Erstellen von branches
  ->Wechseln von commits
  ->Zusammenfügen von commits
- Vieles mehr!

Checkout

- Wechsel zwischen branches
- Updated files im working tree

- Git command:

```
$ git checkout [<branch>]
```

Checkout - Beispiel

```
Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (branch_innhalt)
$ git branch
  branch_commit_1
* branch_innhalt
  branch_innhalt_beginning
  main

Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (branch_innhalt)
$ git checkout branch_commit_1
Switched to branch 'branch_commit_1'

Kevin@Kevin-PC MINGW64 ~/Documents/GitHub/Seminar-Effizente-Programmierung (branch_commit_1)
$
```

Cherry-Picking

- Wähle einen (besonderen) commit.

->Untersuche unterschiede zwischen partent und HEAD

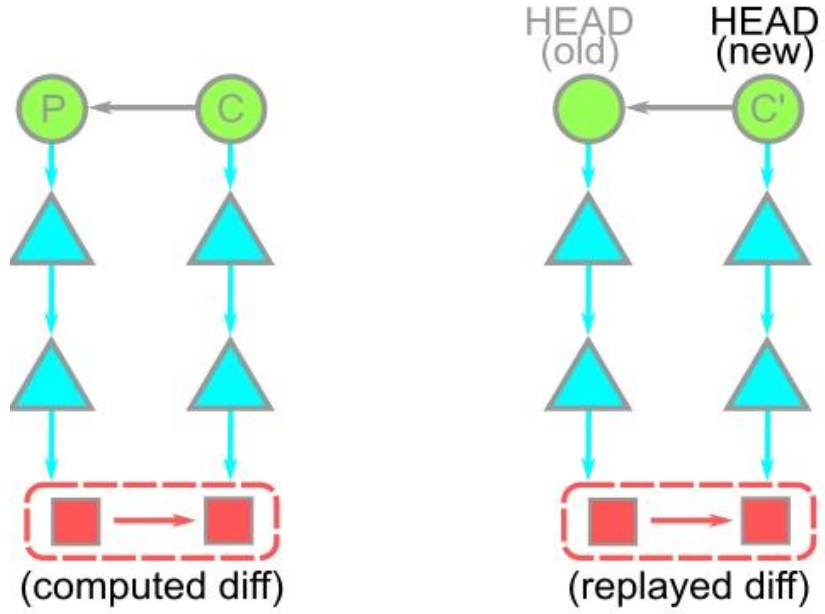
->Wende unterschiede auf HEAD an

->Erstelle neuen Commit

->Bewege HEAD auf den neuen commit

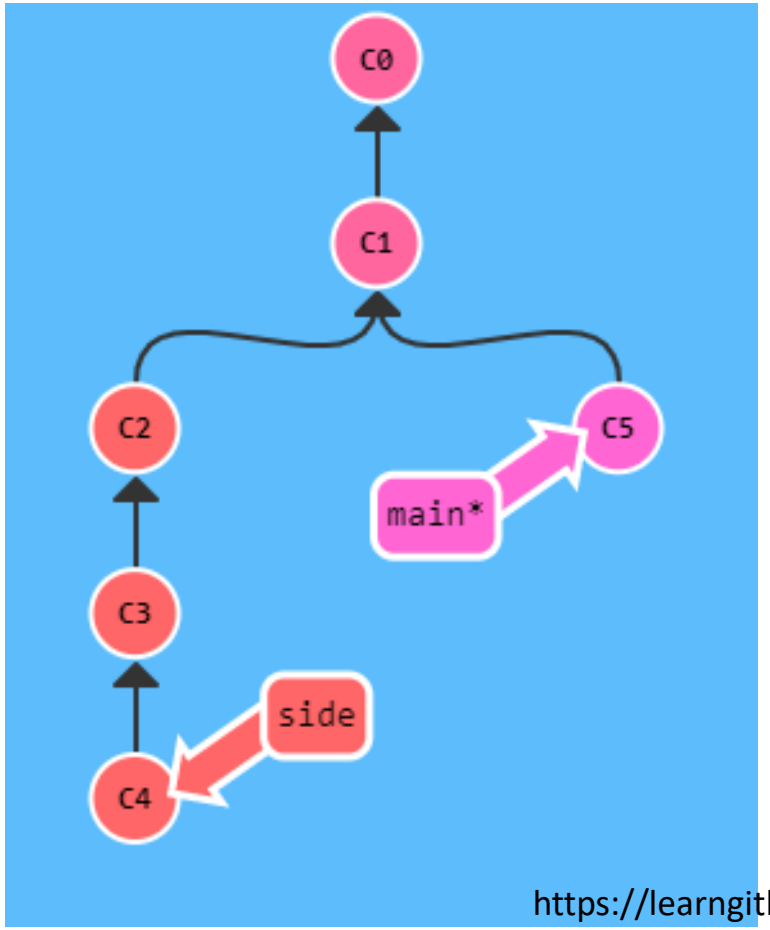
- Git command:

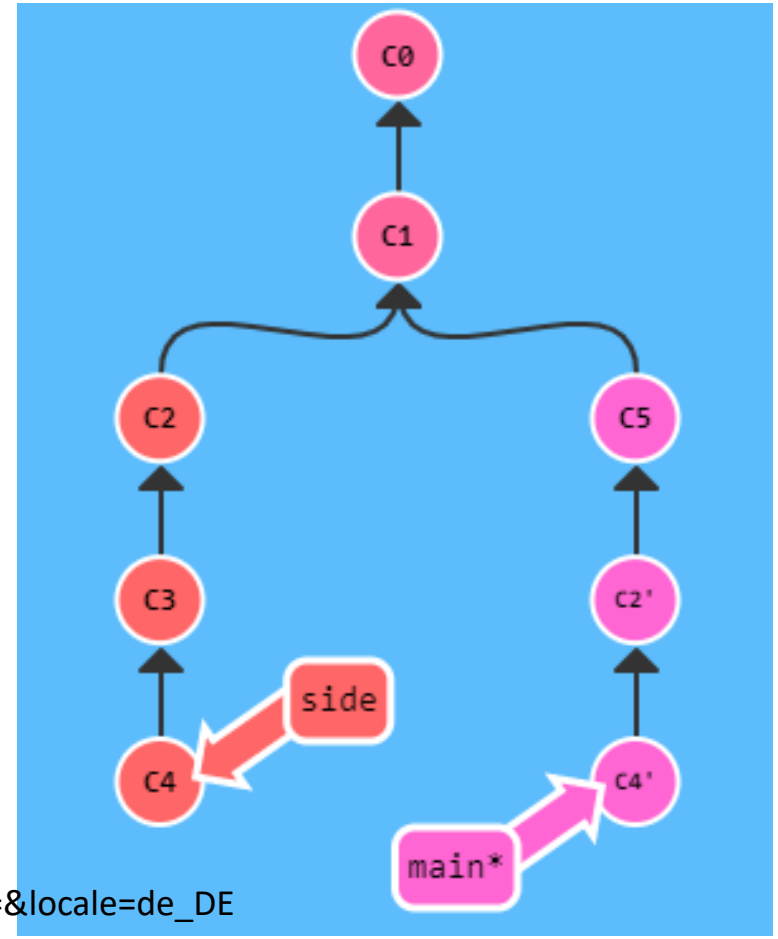
```
$ git cherry-pick <commit>
```



https://github.blog/2020-12-17-commits-are-snapshots-not-diffs/

Cherry-Picking - Beispiel





https://learngitbranching.js.org/?demo=&locale=de_DE

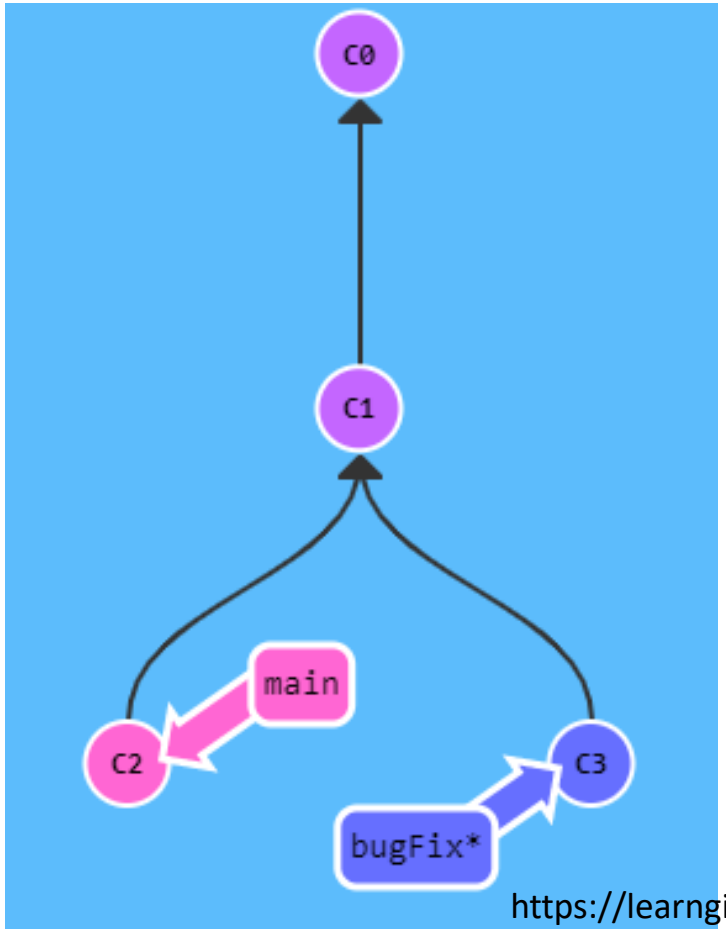
Rebase

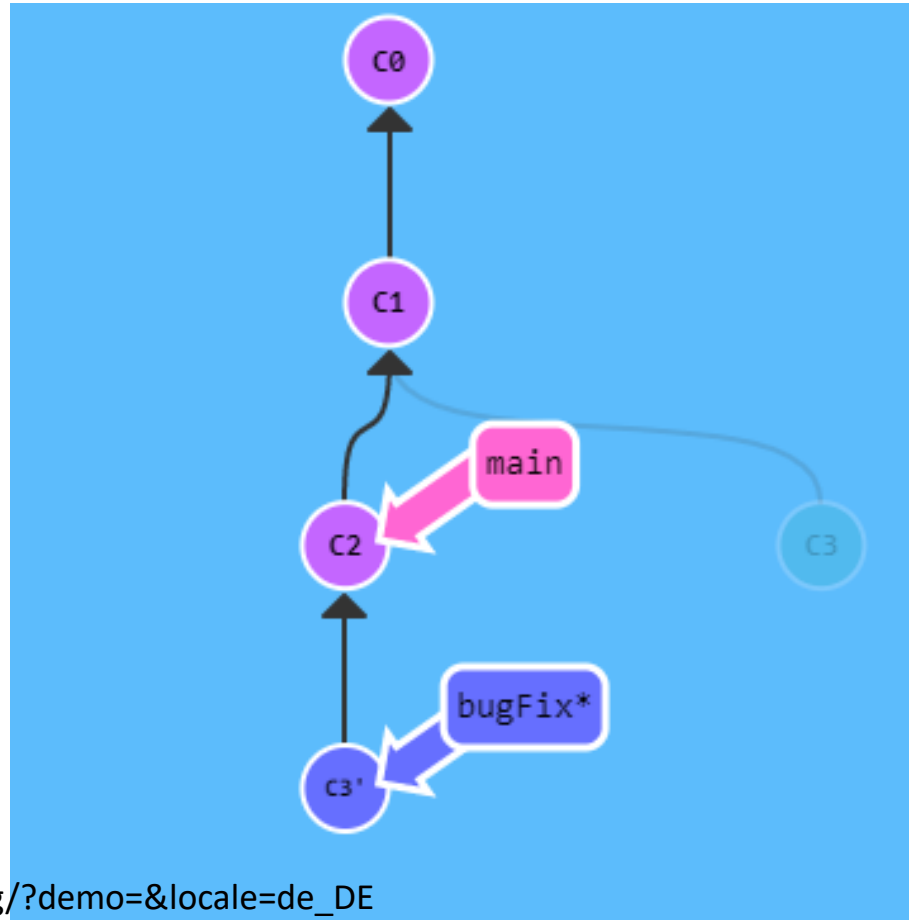
- Im Grunde eine menge `git cherry-pick` commands
- Erstellt neue commits
- Branch wird ebenfalls mit verschoben
- Alte commits existieren noch
  -> sind löschbar da diese nicht mehr benutzt werden

- Git command:

```
$ git rebase <commit>
```

Rebase - Beispiel





https://learngitbranching.js.org/?demo=&locale=de_DE

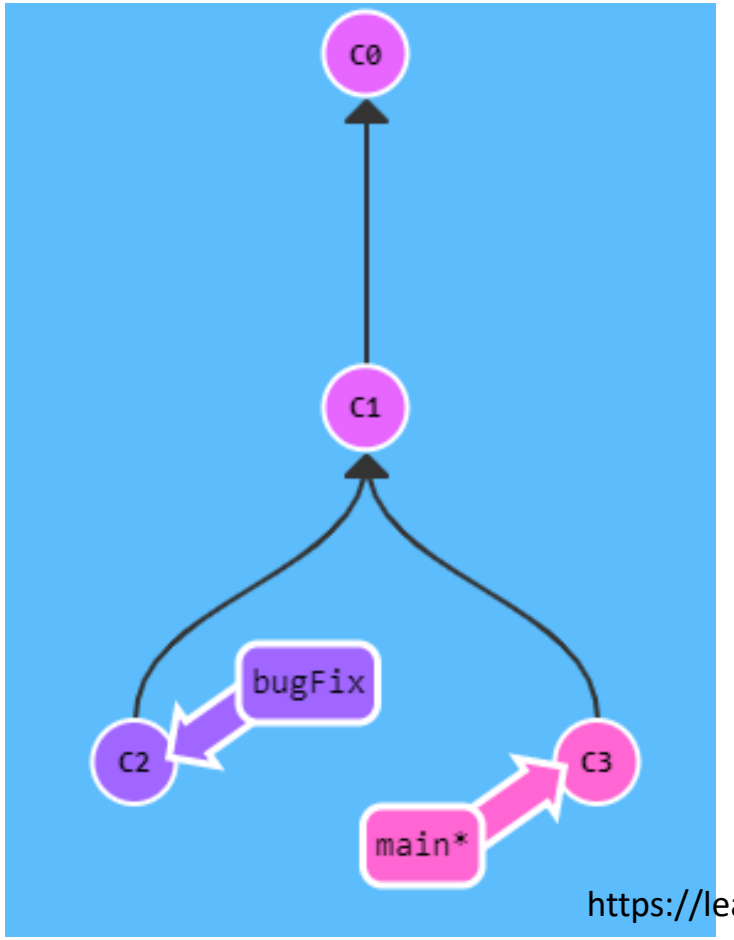
Merge

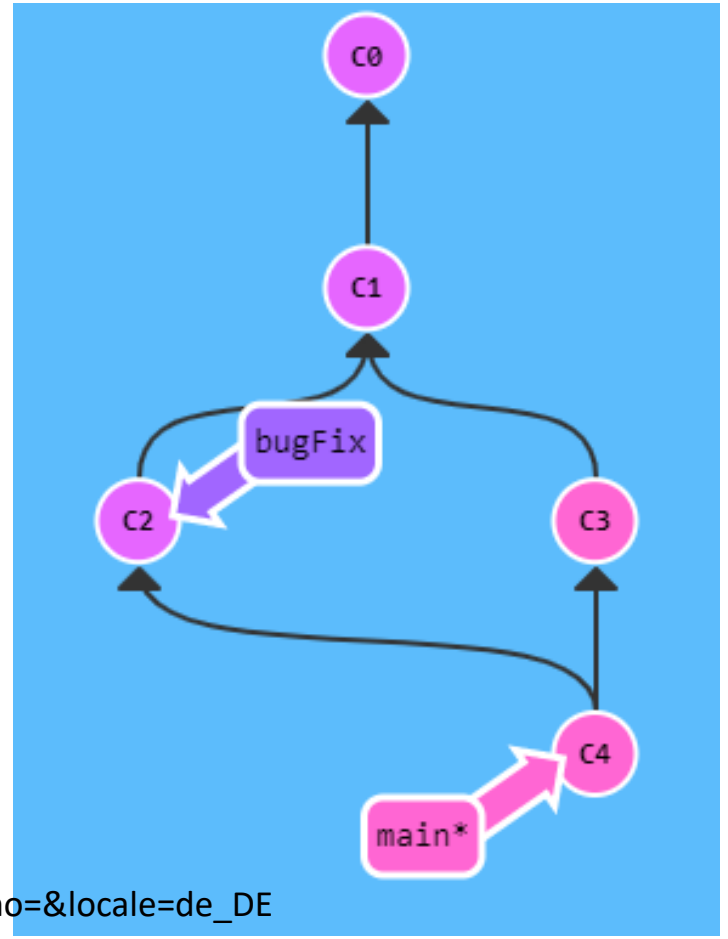
- Fügt mehrere commits zu einem zusammen
- Referenz auf Parent beinhaltet mehrere Einträge
- Merged auf aktuellen branch

- Git command:

```
$ git merge <commit>
```

Merge - Beispiel





https://learngitbranching.js.org/?demo=&locale=de_DE

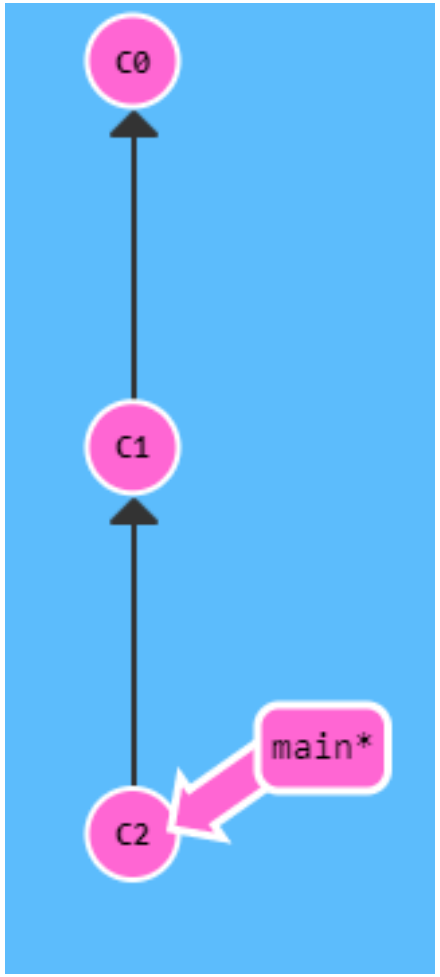
Reset

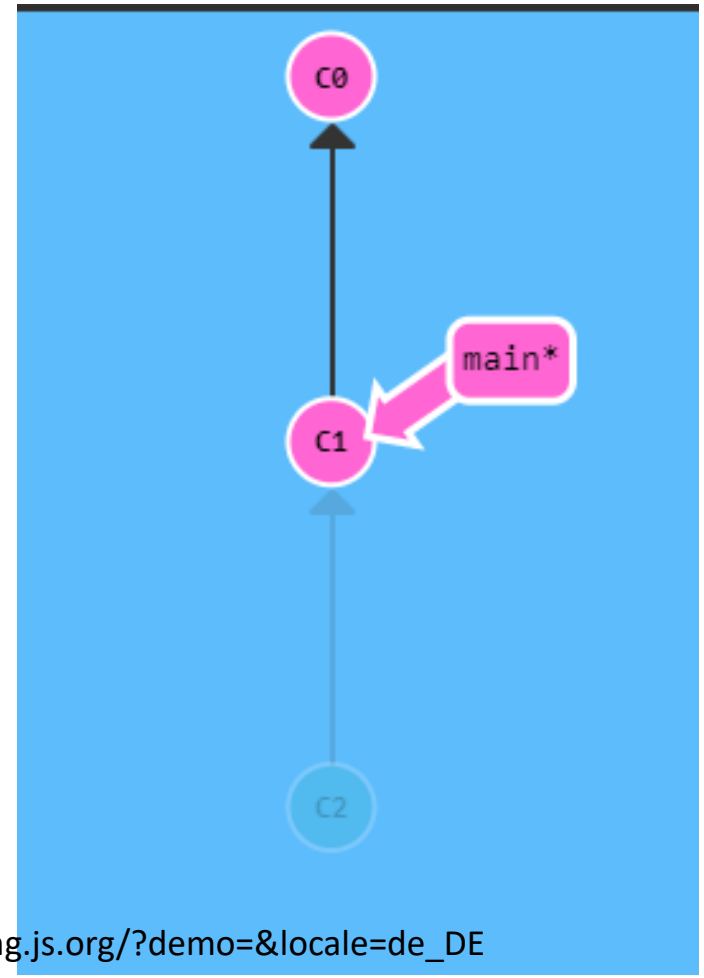
- Änderung rückgängig machen
- Branch-Referenz wird auf ein anderen commit gesetzt
- Meistens Parent

- Git command:

```
$ git reset <commit> (meistens HEAD^)
```

Reset - Beispiel





https://learngitbranching.js.org/?demo=&locale=de_DE

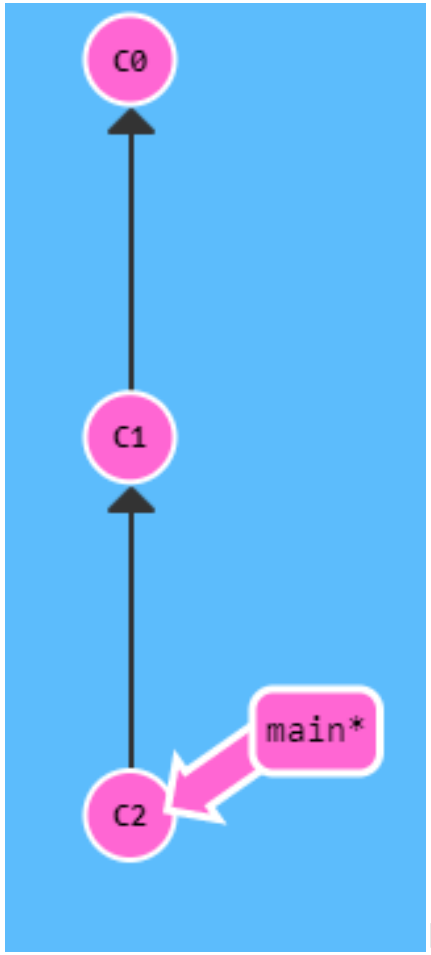
Revert

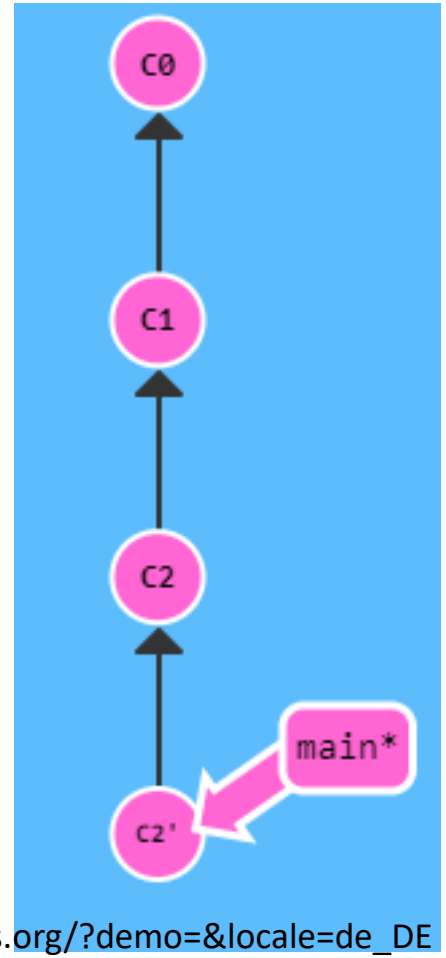
- Änderung rückgängig machen
- Erstellt neuen commit
- Änderungen enthält die Änderungen, die den letzten commit aufheben

- Git command:

```
$ git revert <commit> (meistens HEAD)
```

Revert - Beispiel





https://learngitbranching.js.org/?demo=&locale=de_DE

Fetch

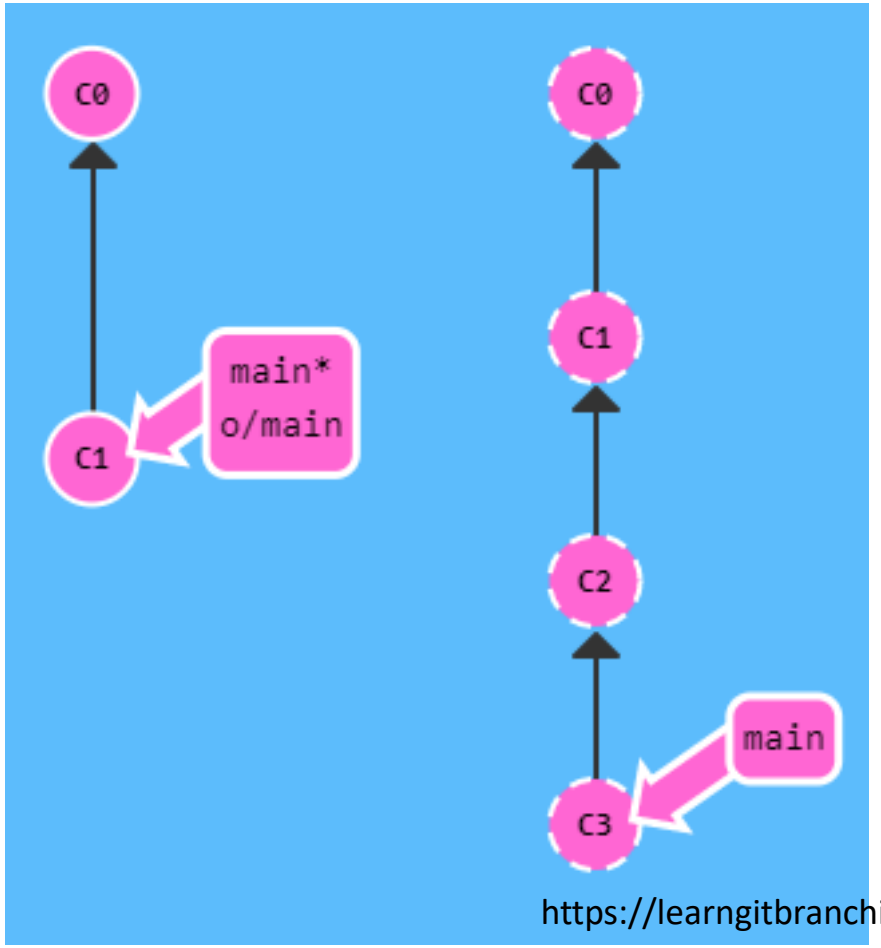
- Synchronisiert die lokale Abbildung mit dem Server (Download)
- Lokale branches sind unberührt
  (o steht für origin)

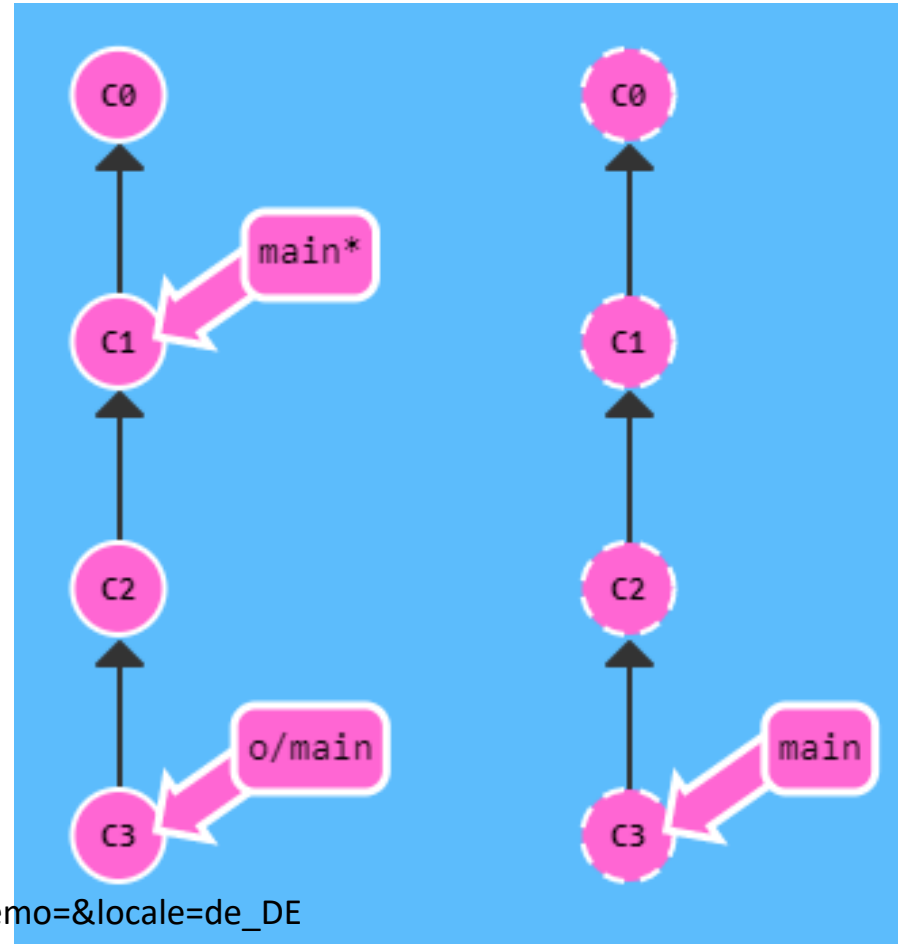
- Git command:

```
$ git fetch
```

https://learngitbranching.js.org/?demo=&locale=de_DE

Fetch - Beispiel





https://learngitbranching.js.org/?demo=&locale=de_DE

Pull

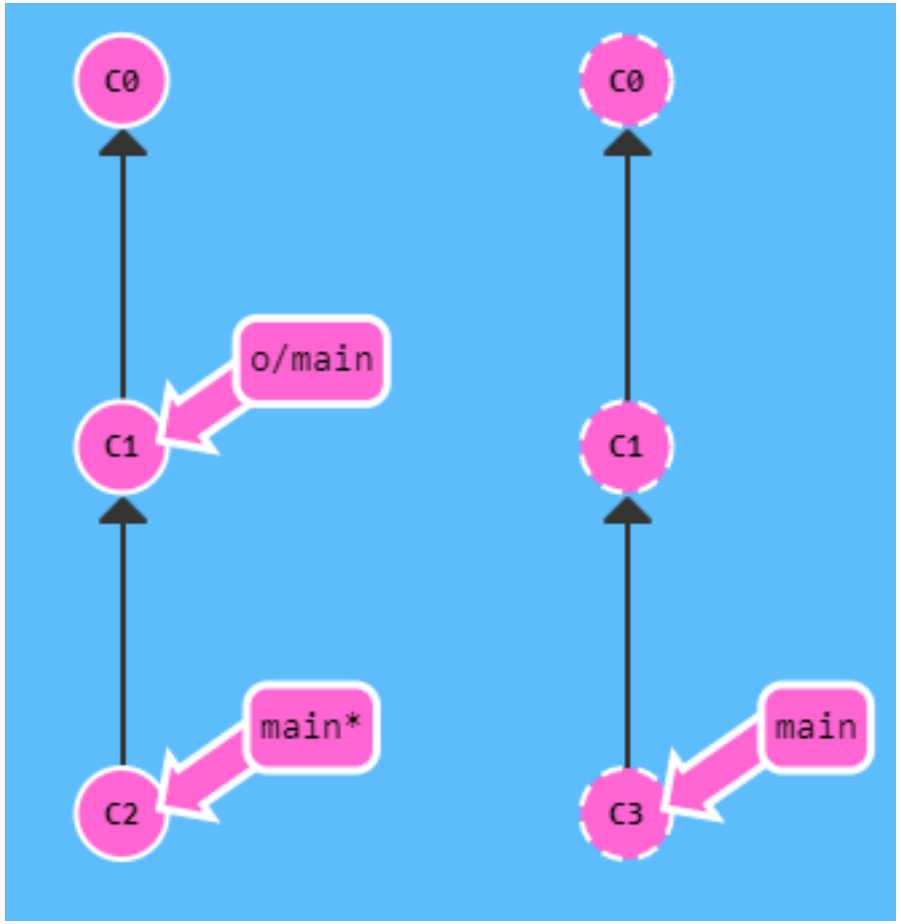
- Führt ein `git fetch` durch
- Ändert unsere localen branches und daten
- Bekannte Wege nach dem fetch:

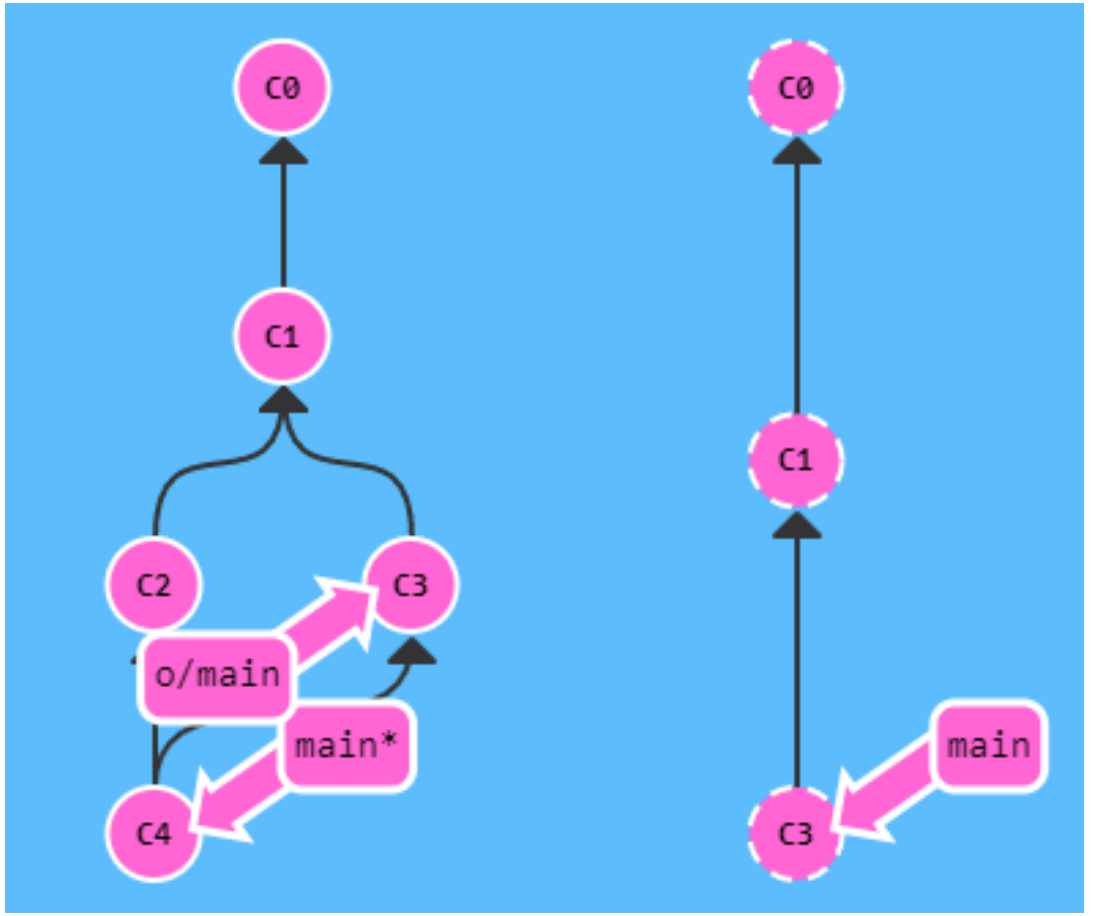
```
$ git cherry-pick oirigin/main
$ git rebase o/main
$ git merge o/main (standartmäßiges pull)
```

Git Command:

```
$ git pull
```

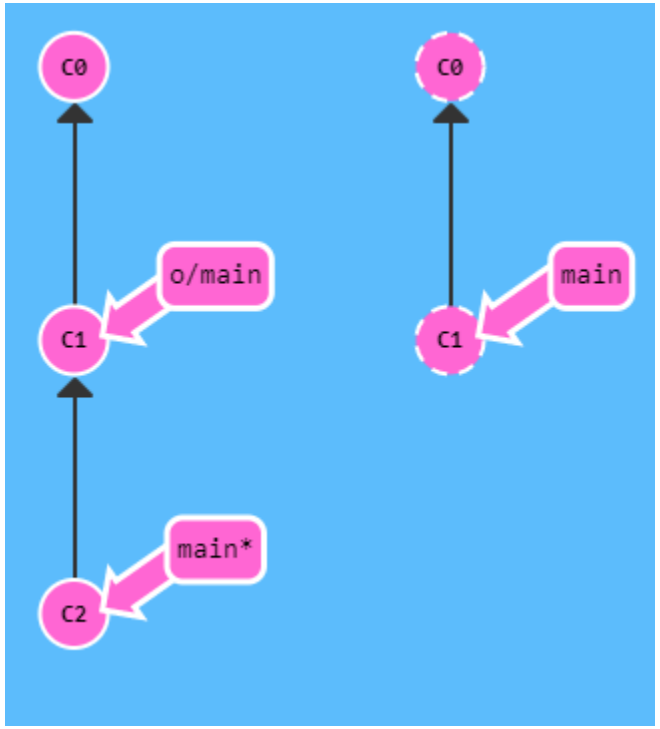
Pull - Beispiel





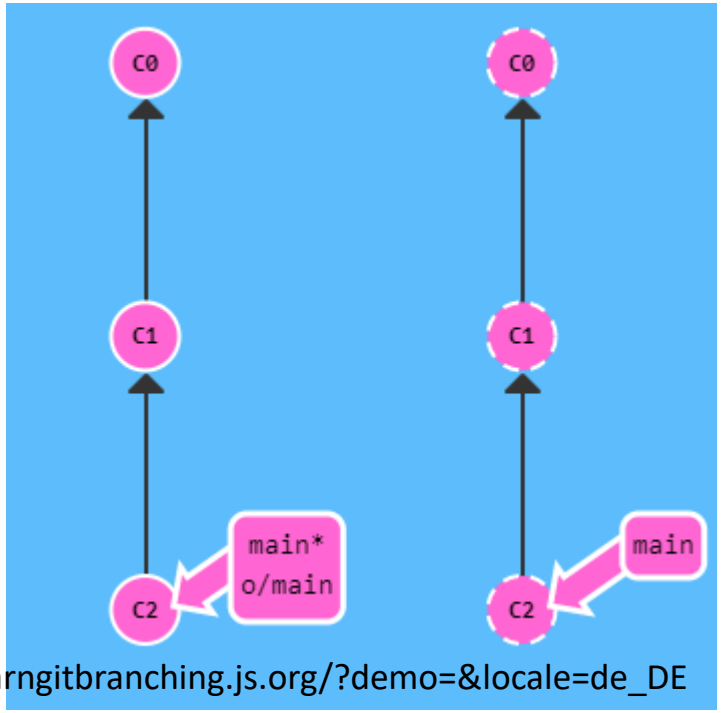
Push + Beispiel

- Synchronisiert den Server mit der lokalen Abbildung (Upload)



- Git Command:

```
$ git push
```



https://learngitbranching.js.org/?demo=&locale=de_DE

Zusammenfassung

- Git ist ein DVCS
  -> Lokale Kopie
  -> Einfaches navigieren durch Versionen
- Git besitzt 3 Objekt-Typen
  ->blobs
  ->tree
  ->commits
- Commit sind Snapshots!
- Navigation mit git

Literatur I

[1] Commits are snapshots, not diffs. https://github.blog/2020-12-17-commits-are-snapshots-not-diffs/ Letzter Zugriff: 11.02.2021

[2] Learn Git Branching https://learngitbranching.js.org/ Letzter Zugriff: 11.02.2021

[3] A Fast Intro to Git Internals. https://www.chromium.org/developers/fast-intro-to-git-internals Letzter Zugriff: 11.02.2021

[4] Scott Chacon and Ben Straub, Pro Git (Second Edition), http://git-scm.com/book/en/v2/ Letzter Zugriff: 11.02.2021

Literatur II

[5] Für alle Dokumentation über die Git Commands https://git-scm.com/ Letzter Zugriff: 11.02.2021

[6] Version 2.30.1 git mit 61.562 commits https://github.com/git/git/tree/v2.30.1/ Letzter Zugriff: 11.02.2021

[7] What is version control: centralized vs. DVCS, https://www.atlassian.com/blog/software-teams/version-control-centralized-dvcs#:~:text=Centralized%20version%20control%20systems%20are,change%20in%20the%20central%20system. Letzter Zugriff: 11.02.2021

Literatur III

[8] Git Internals – How Git works https://www.linkedin.com/pulse/git-internals-how-works-kaushik-rangadurai#:~:text=When%20you%20commit%2C%20git%20stores,pack%20file. Letzter Zugriff: 21.02.2021

[9] Anzahl der Mitarbeiter von Facebook weltweit in den Jahren 2004 bis 2019, https://de.statista.com/statistik/daten/studie/193372/umfrage/anzahl-der-mitarbeiter-von-facebook-weltweit/#:~:text=Die%20Anzahl%20der%20Mitarbeiter%20von,Media%2DGigant%20noch%201.218%20Mitarbeiter.
Letzter Zurgiff: 21.02.2021