

Einführung in GIT

jan.schulz@devugees.org

1. Agenda





1. Was ist GIT?
2. History
3. Collaboration
4. Feature Branches
5. Vokabeln
6. GitHub
7. Remote Repository Befehle
8. Lets GIT our hands dirty ...
9. Commits im Detail
10. Merging

1. Was ist GIT?

- Version Control System (VCS)
- **GIT hilft uns, unsere Projektdateien zu verwalten**

1. Was ist GIT?

- Version Control System (VCS)
- **GIT hilft uns, unsere Projektdateien zu verwalten**

 index	15.10.2017 15:38	Chrome HTML Docu...	1 KB
 jquery-3.2.1.min	15.10.2017 15:35	JScript-Skriptdatei	85 KB
 main	15.10.2017 17:43	JScript-Skriptdatei	1 KB
 style	15.10.2017 15:35	Kaskadierendes Styl...	0 KB

1. Was ist GIT?

- Wie hilft uns GIT, unsere Projektdateien zu verwalten?

1. History
2. Collaboration
3. Feature branches

2. History

- GIT beaufsichtigt alle Änderungen, die wir an unseren Projektdateien vornehmen

1. History

Oct. 2017: Wir kreieren die Datei **banners.css**

Dec.2017: Wir ändern **banners.css**

Jan.2018: Unsere Seite sieht kaputt aus?!

Wir gucken mal auf banners.css von Oct. 2017 und stellen fest, dass wir die folgenden Änderungen vorgenommen haben:

*„ float: left;
padding: 2rem;
margin: 2rem; “*

1. History

- GIT erlaubt uns, die history einer Datei zu sehen
- GIT erlaubt uns Dateiänderungen rückgängig zu machen

1. History

- GIT erlaubt uns, die history einer Datei zu sehen
- GIT erlaubt uns Dateiänderungen rückgängig zu machen

-> Nichts geht verloren

-> Nichts ist endgültig

2. Collaboration

- Etwas **alleine** programmieren:
 - Du
 - Deine Dateien

2. Collaboration

- Etwas **alleine** programmieren:
 - Du
 - Deine Dateien
- Etwas **im Team** programmieren:
 - Du
 - Deine Teammitglieder
 - Deine Dateien
 - Die Dateien von Deinen Teammitgliedern

2. Collaboration

- Etwas alleine programmieren:
 - EINFACH
- Etwas im Team programmieren:
 - NICHT SO EINFACH

2. Collaboration

**Du willst ein Buch schreiben „mybook.docx“
über Deine Heimatstadt mit Deinem Freund.**

2. Collaboration

**Du willst ein Buch schreiben „mybook.docx“
über Deine Heimatstadt mit Deinem Freund.**

Du: „Kannst Du bitte Kapitel 4 angucken und
noch Ergänzungen vornehmen?“

Dein Freund: „Okay, Ich brauch 2 Tage dafür.“

2. Collaboration

Gleich nachdem Du die aktuelle Version an Deinen Freund geschickt hast ...

1. Du siehst ein paar Rechtschreibfehler und willst Sie übersetzen.
2. Dir fallen ein paar gute Bilder für Kapitel 1 ein die Du gerne einfügen würdest.

Und schon arbeitet Dein Freund nicht mehr mit der aktuellsten Version!

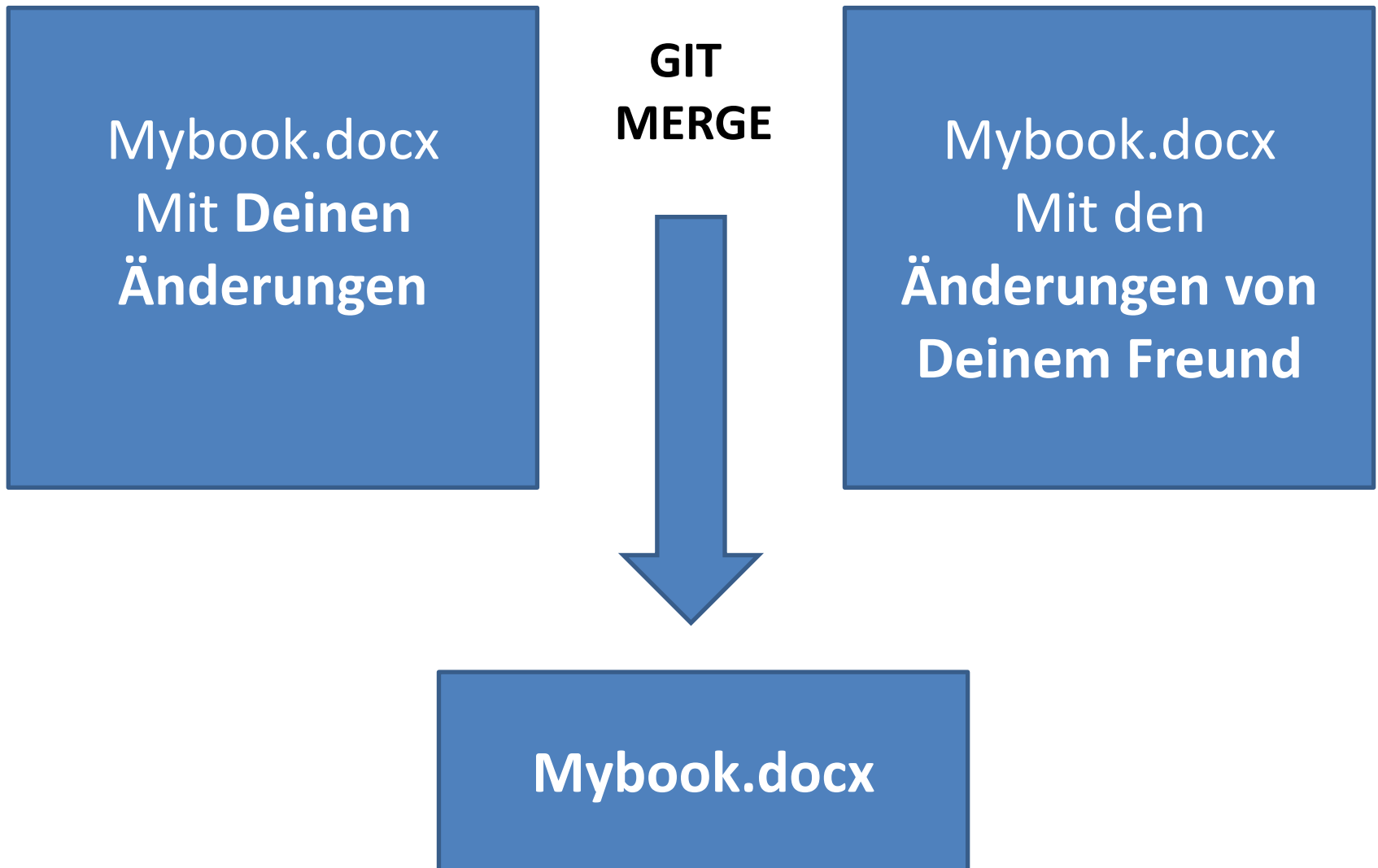
2. Collaboration

**Nach 2 Tagen schickt Dein Freund
„mybook.docx“ zurück.**

Mybook.docx
Mit **Deinen**
Änderungen

Mybook.docx
Mit den
Änderungen von
Deinem Freund

2. Collaboration



3. Feature Branches

Task #1

-> Redesign Header



Some Website

3. Feature Branches

Task #1

-> Redesign Header

Some Website

Task #2

-> Redesign Footer

Day 1

Task #1

-> Redesign Header

UNDER CONSTRUCTION

Task #2

-> Redesign Footer

UNDER CONSTRUCTION

Day 2

Task #1

-> Redesign Header

UNDER CONSTRUCTION

Task #2

-> Redesign Footer

UNDER CONSTRUCTION

End of Day 2

Task #1

-> Redesign Header

UNDER CONSTRUCTION

Task #2

-> Redesign Footer

AMAZING FOOTER

End of Day 2

Task #1

-> Redesign Header

UNDER CONSTRUCTION

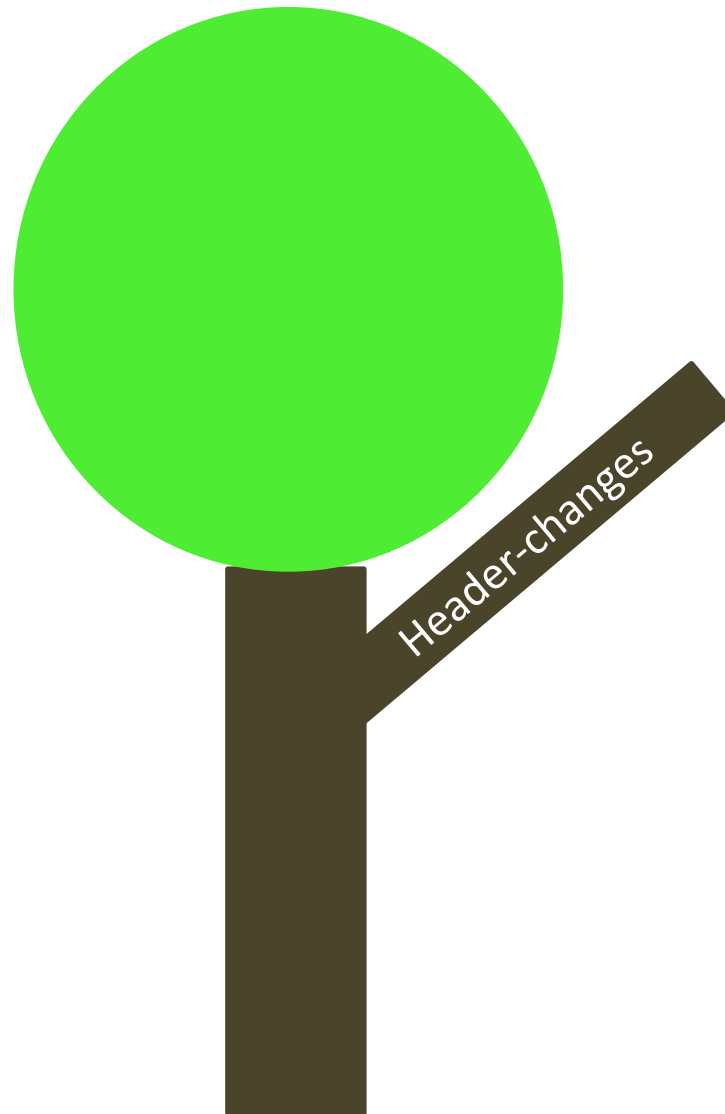
PROBLEM: WEBSITE MIT UNFERTIGEN HEADER CODE

Task #2

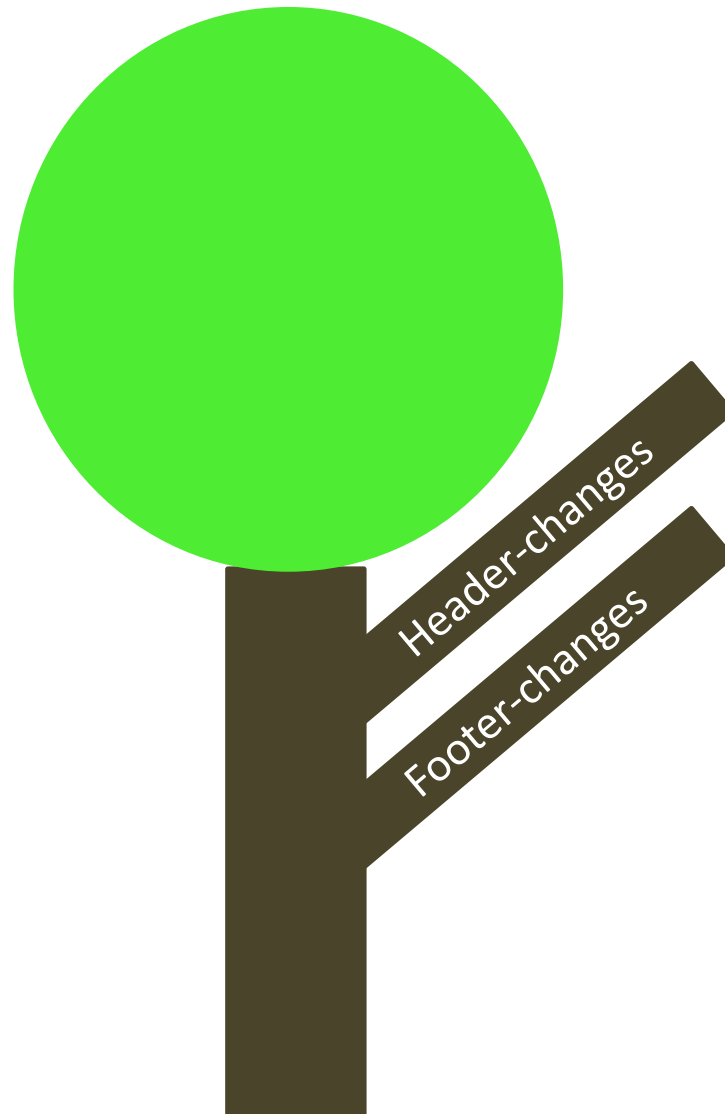
-> Redesign Footer

AMAZING FOOTER

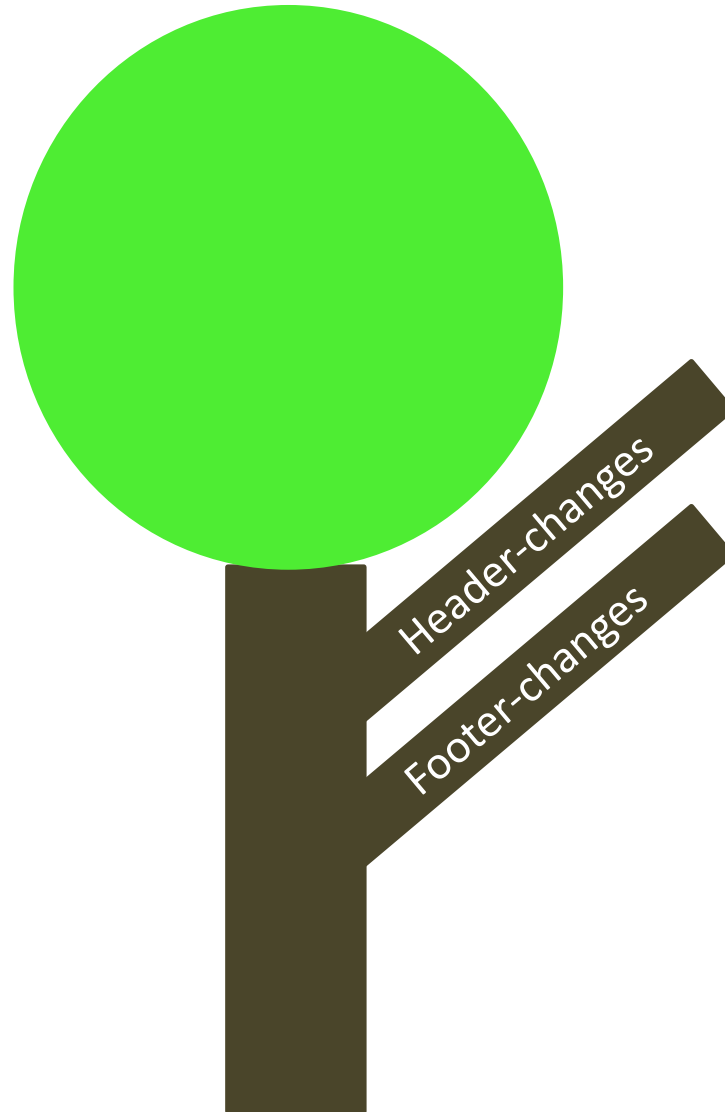
3. Feature Branches



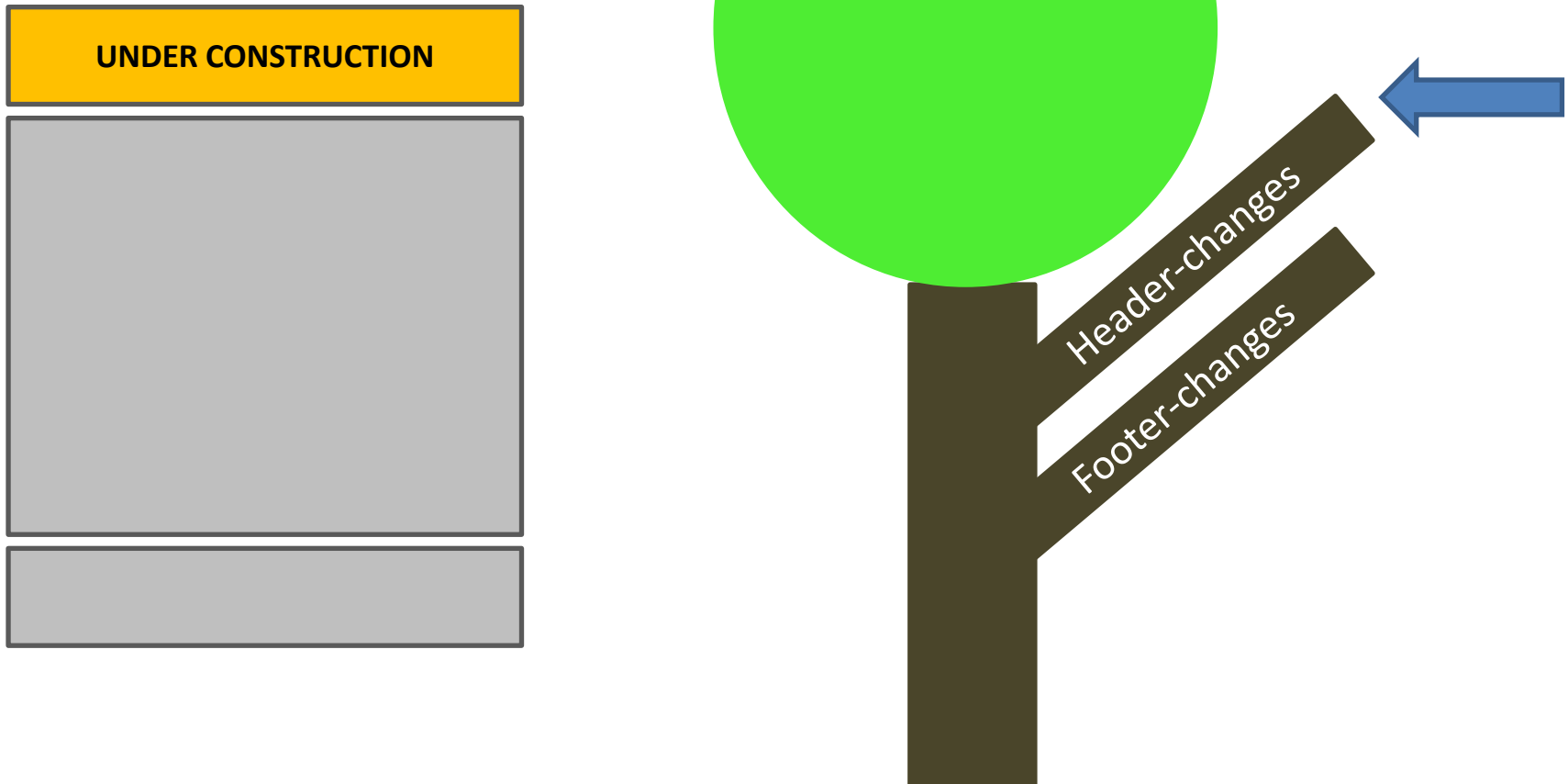
3. Feature Branches



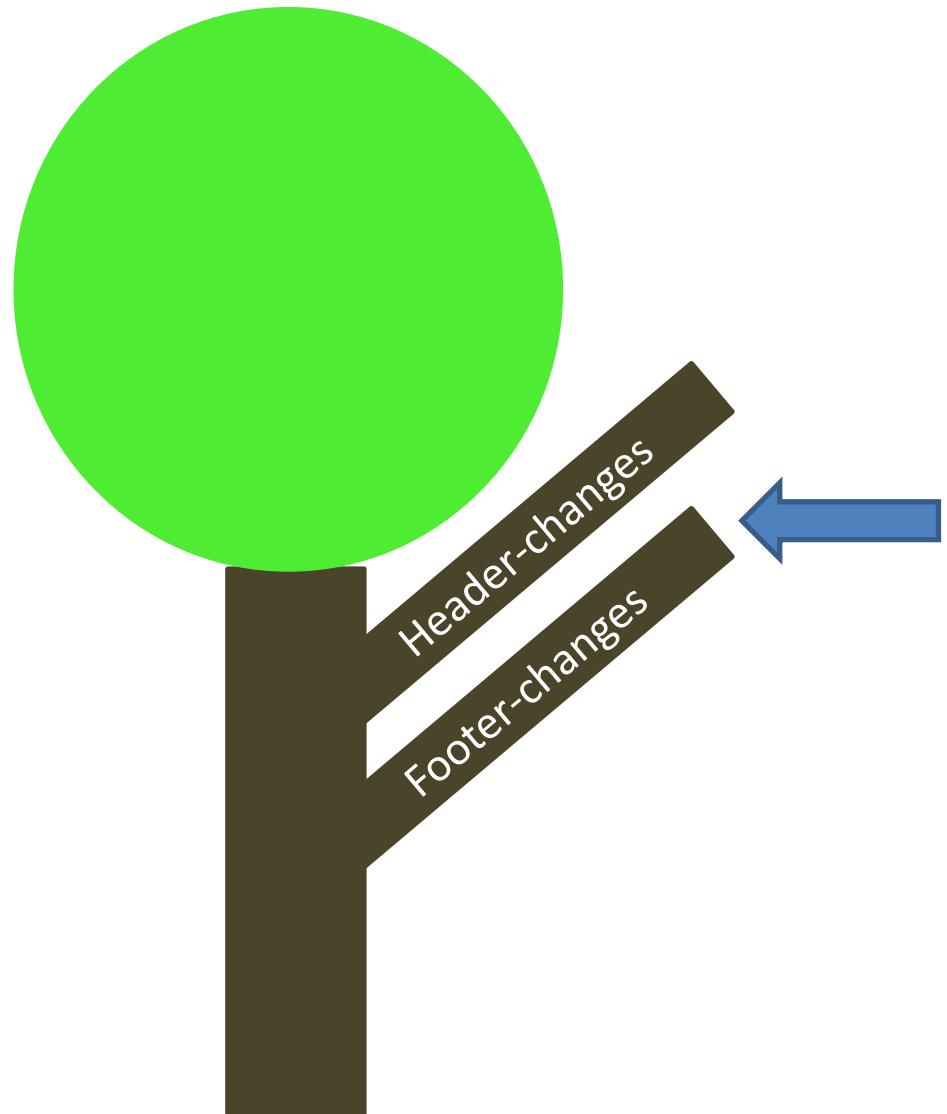
3. Feature Branches



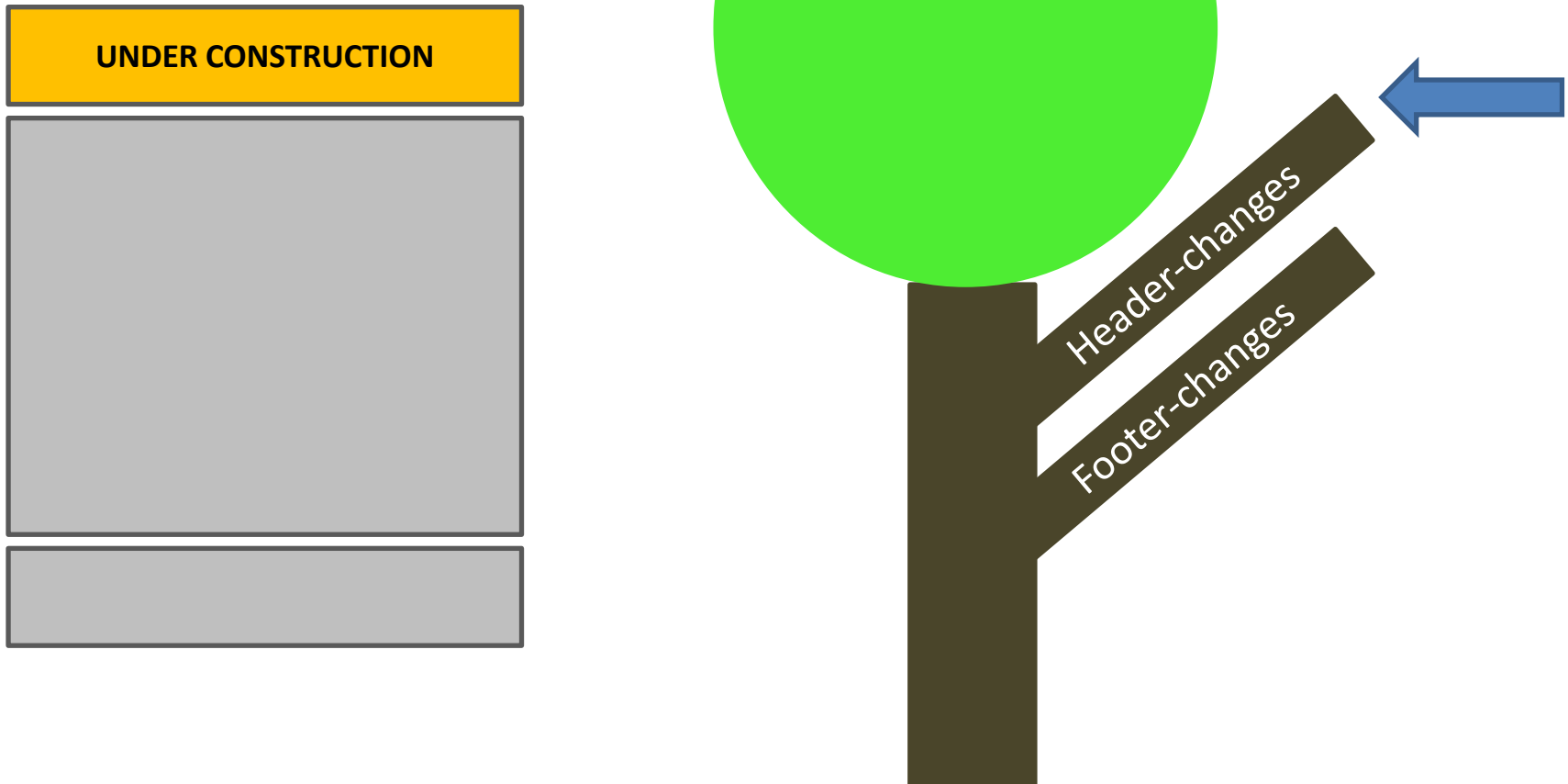
3. Feature Branches



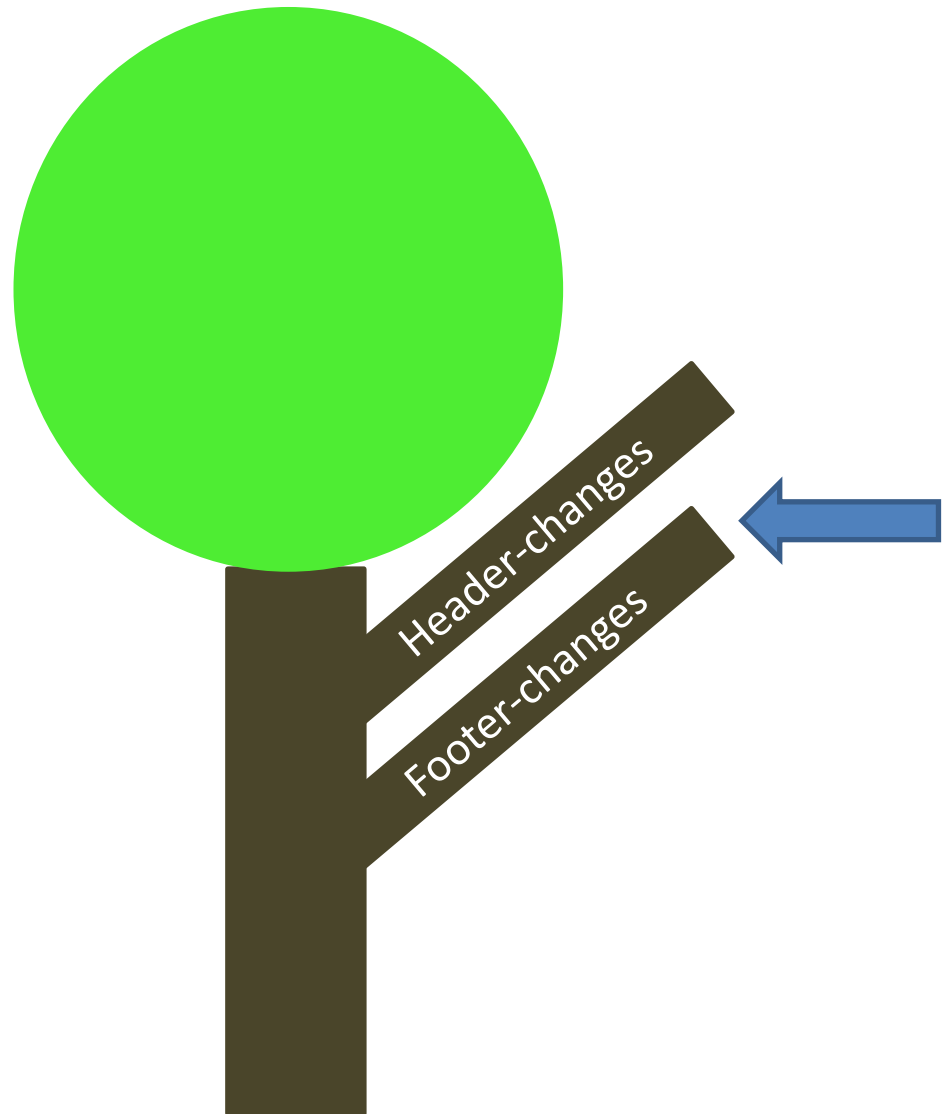
3. Feature Branches



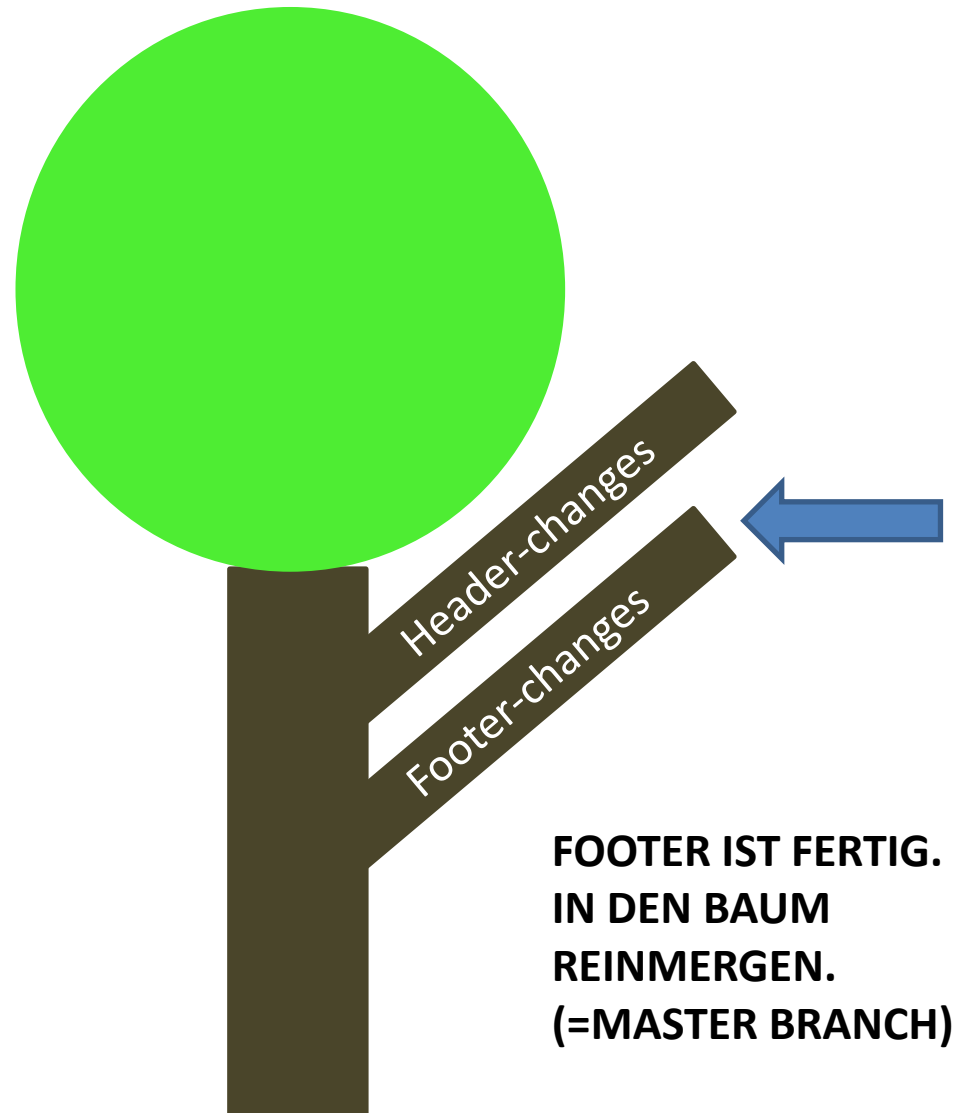
3. Feature Branches



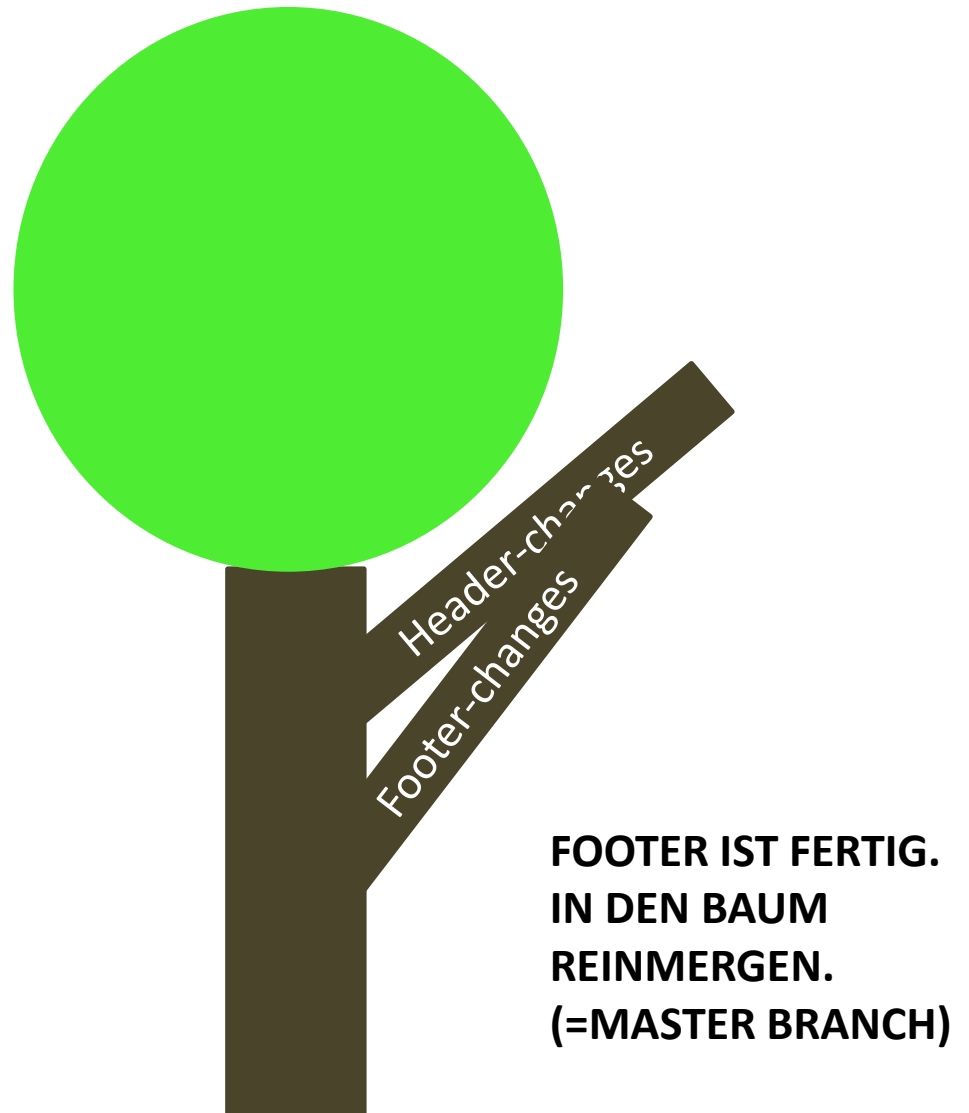
3. Feature Branches



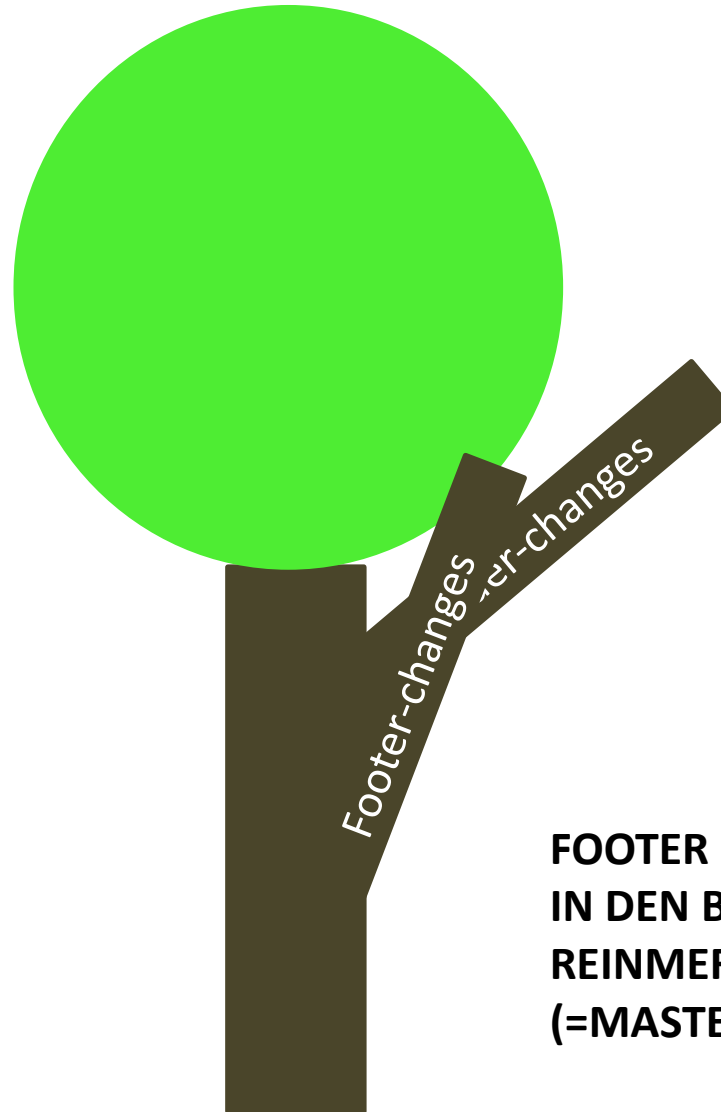
3. Feature Branches



3. Feature Branches

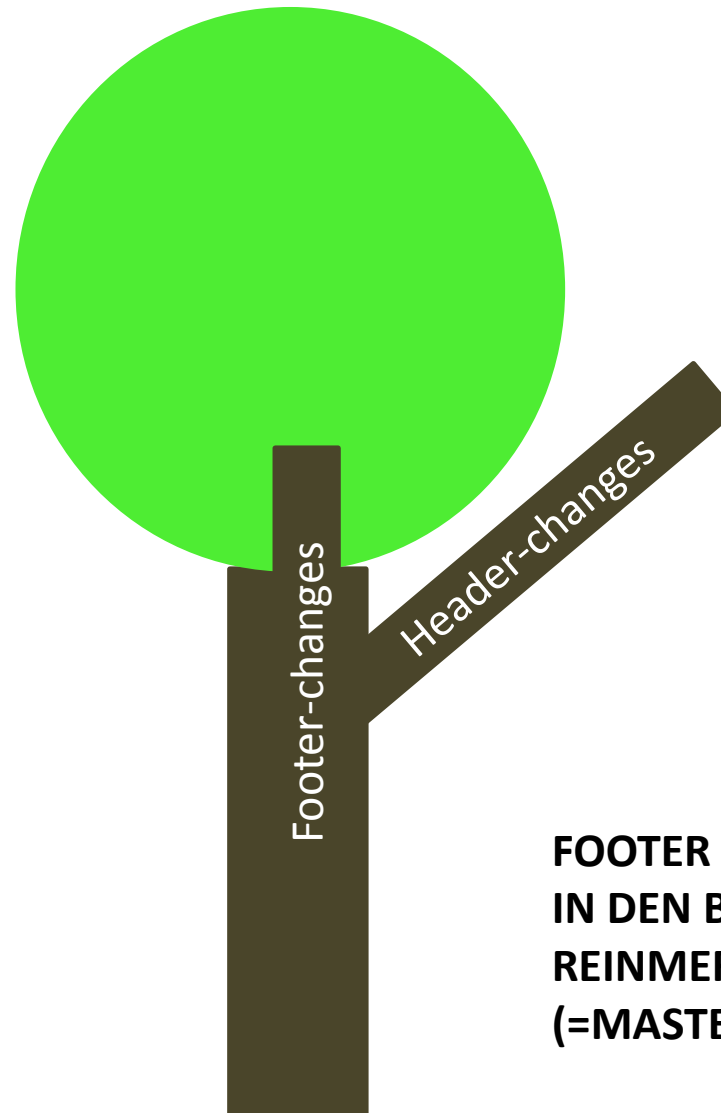


3. Feature Branches



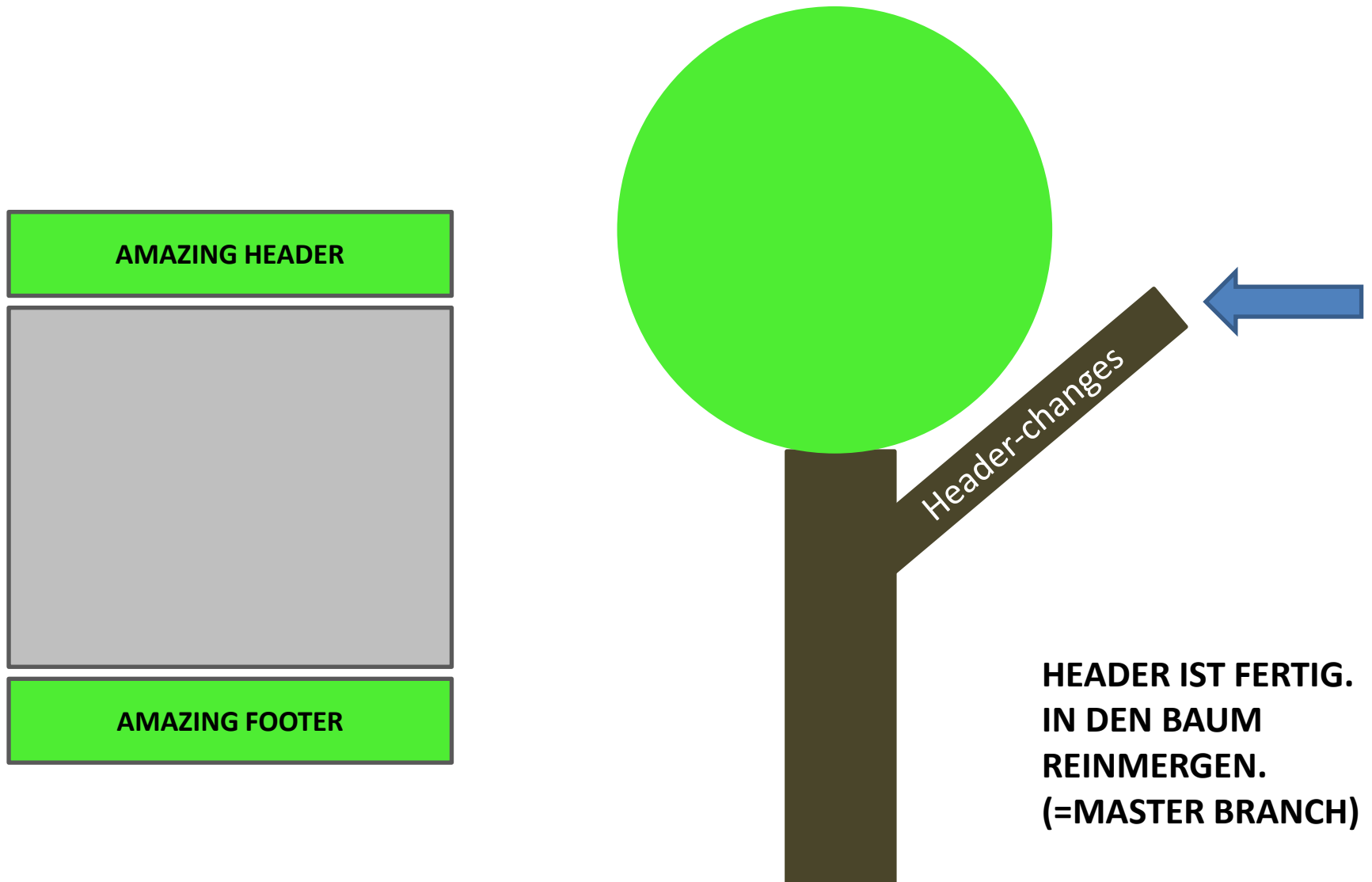
**FOOTER IST FERTIG.
IN DEN BAUM
REINMERGEN.
(=MASTER BRANCH)**

3. Feature Branches

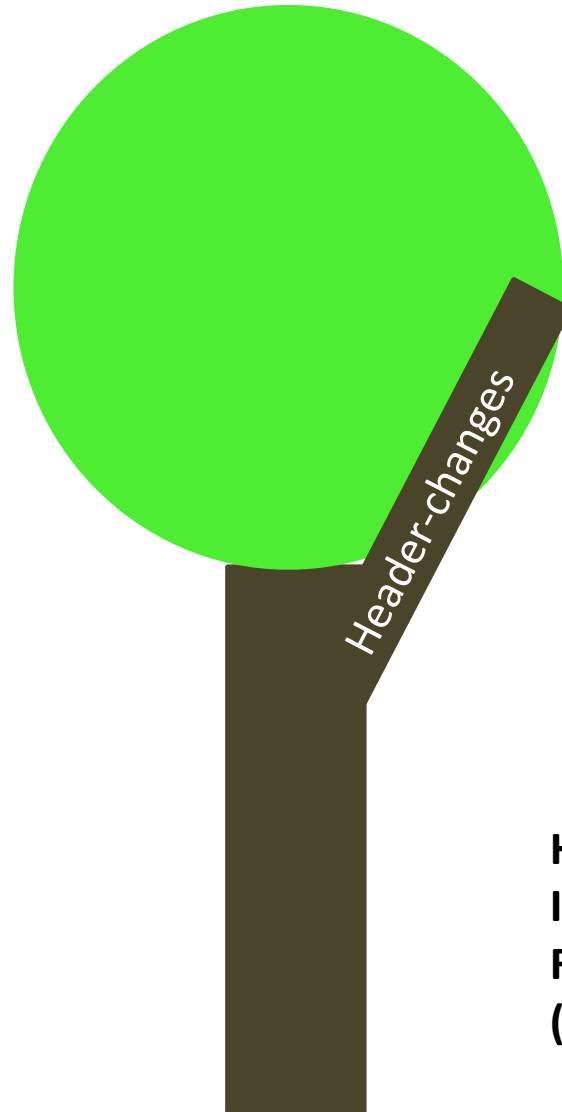


**FOOTER IST FERTIG.
IN DEN BAUM
REINMERGEN.
(=MASTER BRANCH)**

3. Feature Branches

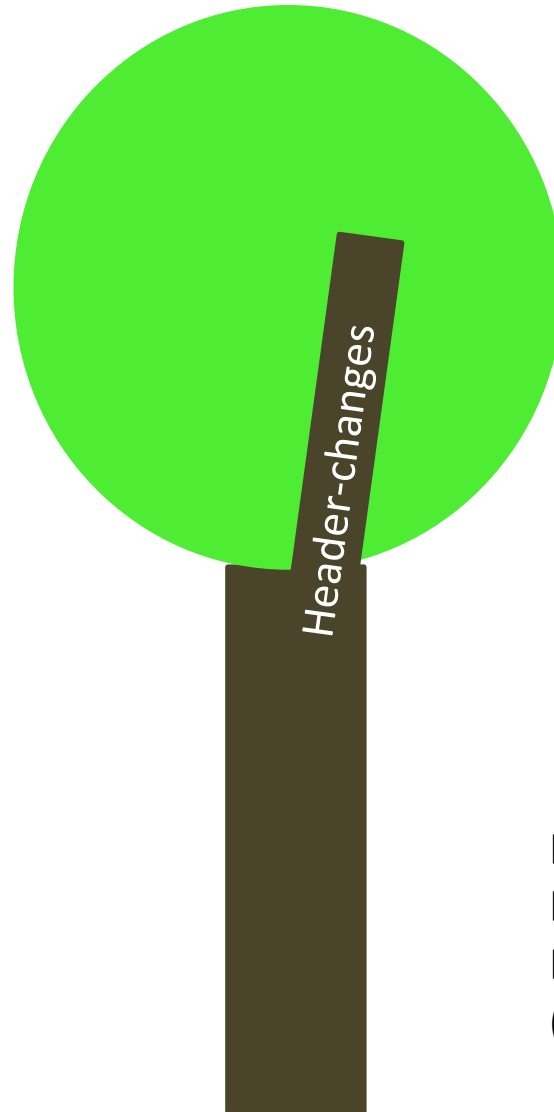


3. Feature Branches



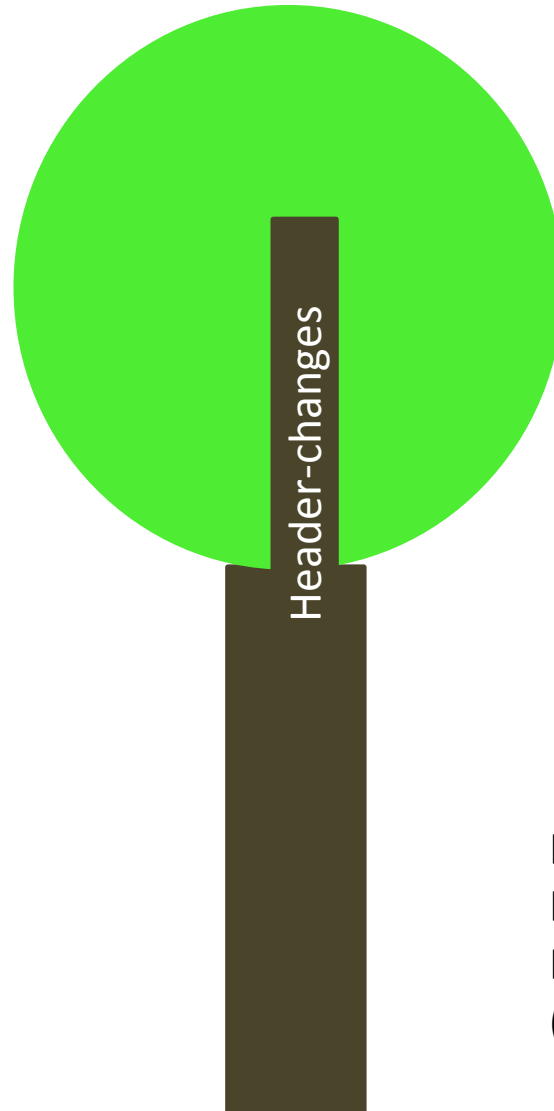
**HEADER IST FERTIG.
IN DEN BAUM
REINMERGEN.
(=MASTER BRANCH)**

3. Feature Branches



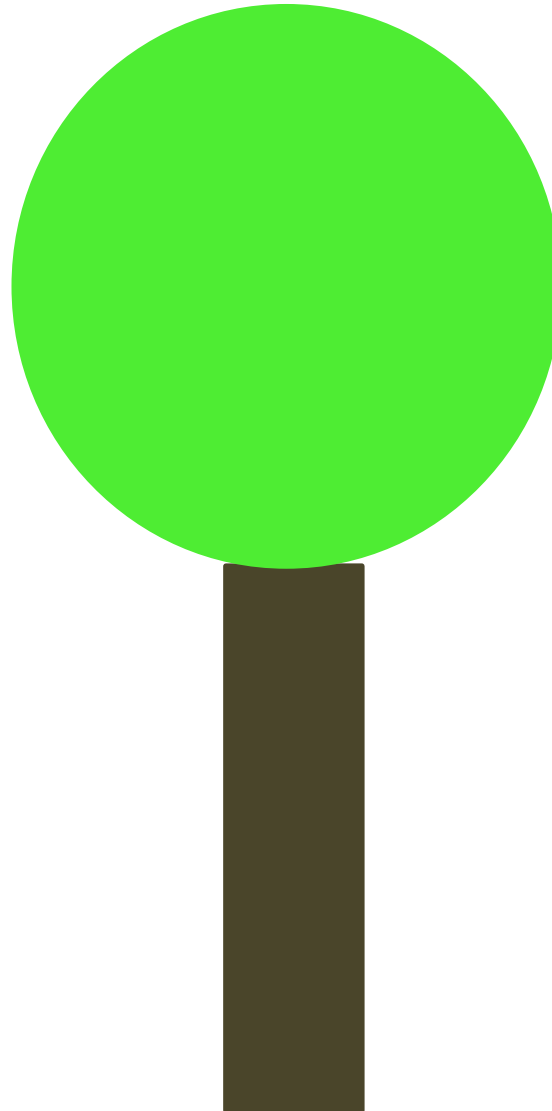
**HEADER IST FERTIG.
IN DEN BAUM
REINMERGEN.
(=MASTER BRANCH)**

3. Feature Branches



**HEADER IST FERTIG.
IN DEN BAUM
REINMERGEN.
(=MASTER BRANCH)**

3. Feature Branches



Review

- Was sind die 3 Core-Funktionen von GIT?

Review

- Was sind die 3 Core-Funktionen von GIT?
 - History
 - Collaboration
 - Feature Branches

4. Vocabulary Time-Out

- **Repository?**

4. Vocabulary Time-Out

- **Repository**
 - Arbeitsverzeichnis, Euer Projektordner
 - GIT beaufsichtigt die Änderungen im Repository

4. Vocabulary Time-Out

- **Repository**
 - Arbeitsverzeichnis, Euer Projektordner
 - GIT beaufsichtigt die Änderungen im Repository
- **Commit?**

4. Vocabulary Time-Out

- **Repository**
 - Working directory, your project files folder
 - GIT's job is to keep track of any changes here
- **Commit**
 - GIT speichert die Änderungen in der History nur ab, wenn wir GIT proaktiv dazu auffordern
= "**GIT's way of saving**"
 - In einem Texteditor sagen wir CTRL+S und dieser speichert die Datei ab
 - In GIT passiert dies nur, wenn wir einen **COMMIT** machen

4. Vocabulary Time-Out

- **Bevor wir COMMIT machen, machen wir STAGE**
- **STAGING** = wir bereiten etwas vor,
 - Wenn Ihr einen Haus verkaufen wollt,
 - wird es erst für den Verkauf vorbereitet,
 - Es wird schick und gemütlich gemacht

Index.html – vor dem Staging



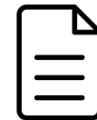
Index.html – Staged and bereit für den Commit



Git States

**Working
Directory**

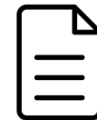
**Modified
Files**



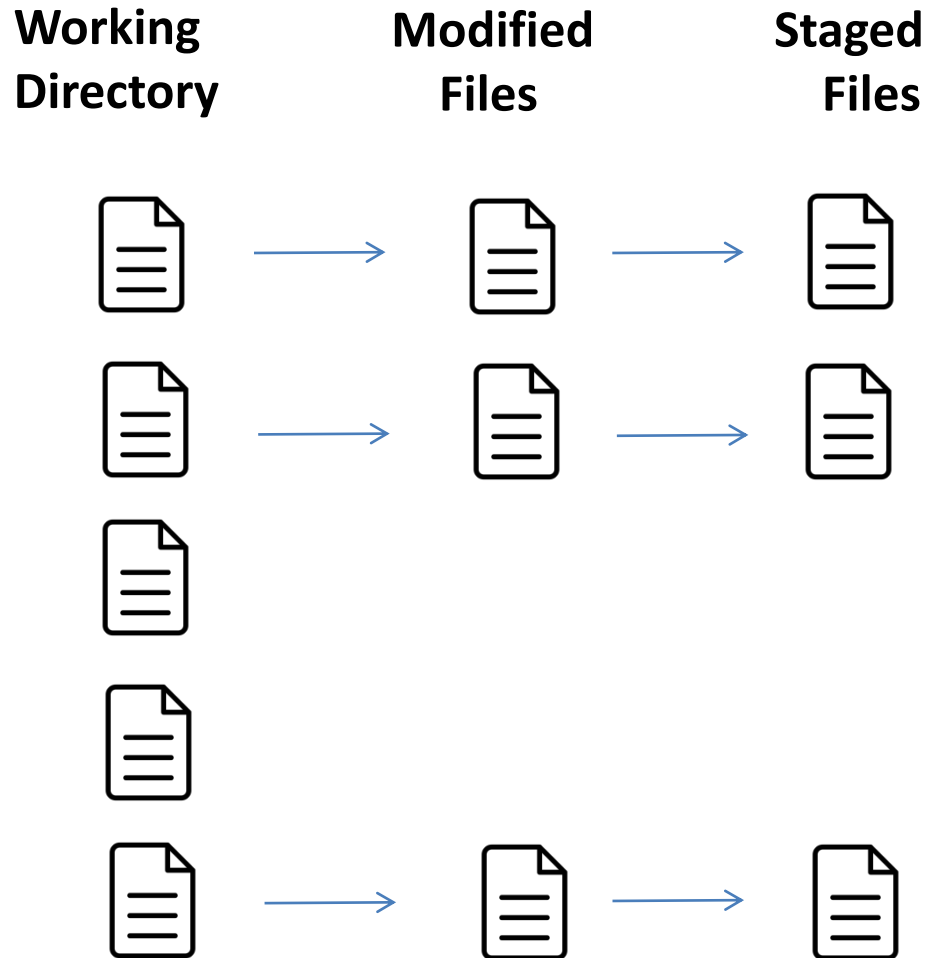
Git States

**Working
Directory**

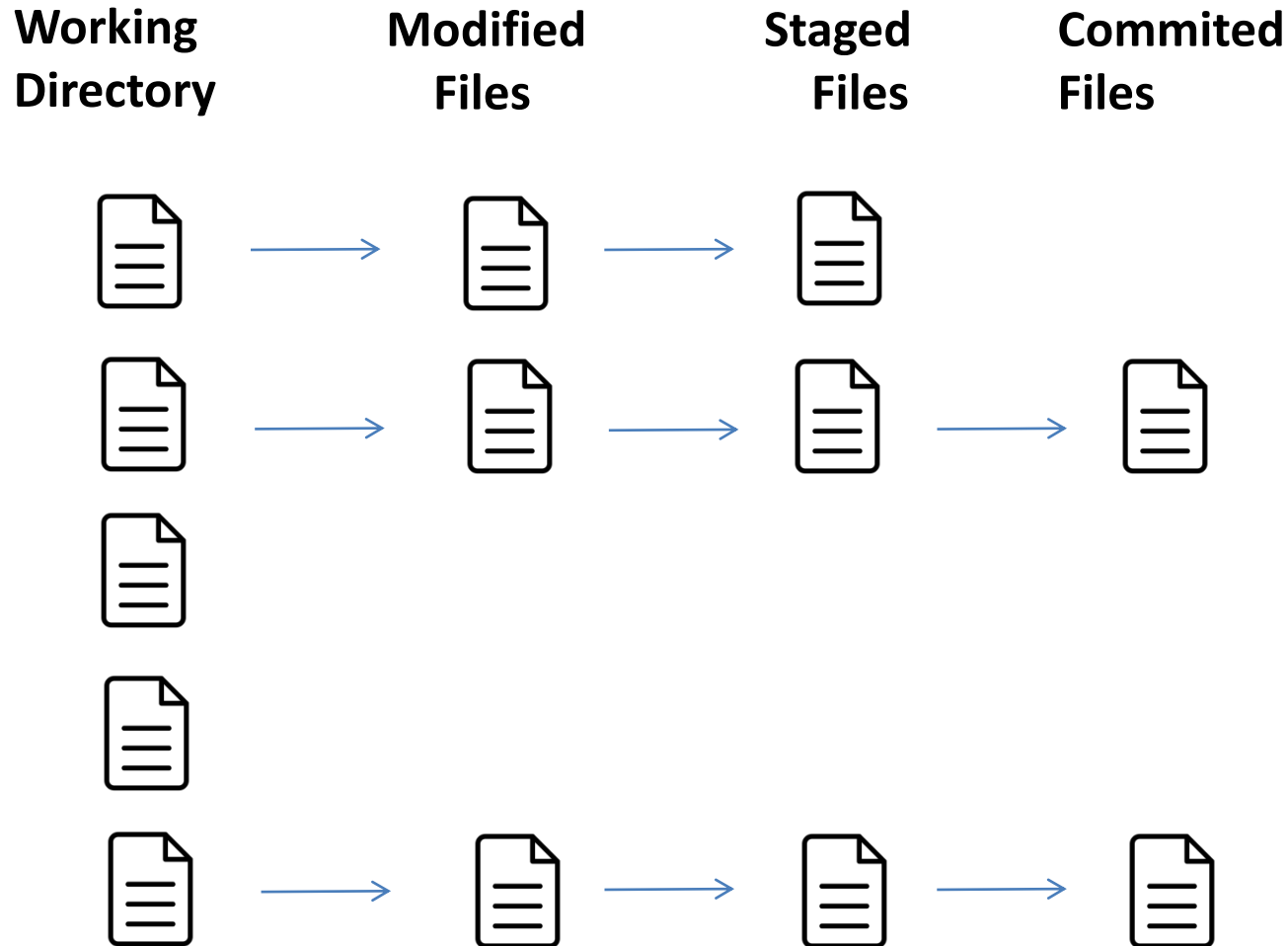
**Modified
Files**



Git States



Git States



Git States



**Working
Directory**

**Modified
Files**

**Staged
Files**

**Committed
Files**

**Remote
Repository**



Git States

LOCAL

REMOTE

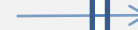
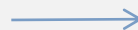
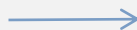
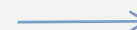
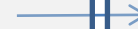
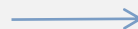
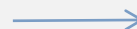
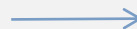
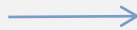
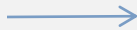
Working
Directory

Modified
Files

Staged
Files

Committed
Files

Remote
Repository



5. GitHub

- = Dein Remote Repository im Internet
- Frei: Public Repositories
- Premium: Public + Private Repositories
- Public: Jeder kann Deinen Code sehen
- Private: Du entscheidest wer Deinen Code sehen kann
- BitBucket.com: Freie Private Repositories

6. Remote Repository Befehle

- Clone
 - Lade komplettes Remote-Verzeichnis auf Deinen lokalen Computer
- Fork
 - Lade komplettes Remote-Verzeichnis in Dein Remote-Repository
- Push
 - Lade/Pushe gesamtes Lokales Verzeichnis in das Remote-Repository
- Pull
 - Downloade/Pulle die letzten Änderungen des Remote-Verzeichnisses auf Deinen lokalen Rechner

Closer look at commits

- **Ein commit** beinhaltet:
 - Einen Zustand des Verzeichnisses
 - Eine Referenz/Zeiger auf seinen Vorgänger
- **Ein commit** ist:
 - Identifiziert durch einen **ref**
- **HEAD** eine Referenz/Zeiger die immer auf das Arbeitsverzeichnis zeigt

Ok ...


8. Lets **GIT** our hands dirty

Preparations

[http://blogs.pdmlab.com/alexander.zeitler/
articles/installing-and-configuring-p4merge-for-
git-on-ubuntu](http://blogs.pdmlab.com/alexander.zeitler/articles/installing-and-configuring-p4merge-for-git-on-ubuntu)

Let us install P4Merge together ...

Commits im Detail



Main.js
Navbar.js
Index.html

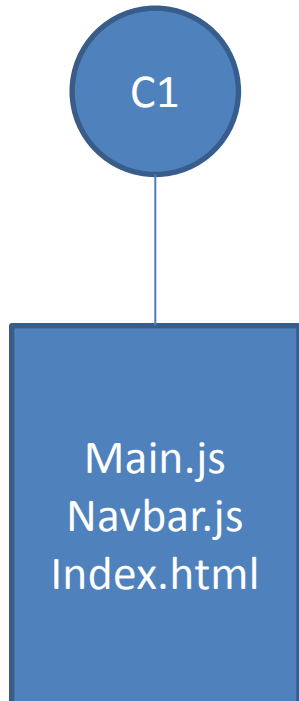
Commits im Detail

HEAD

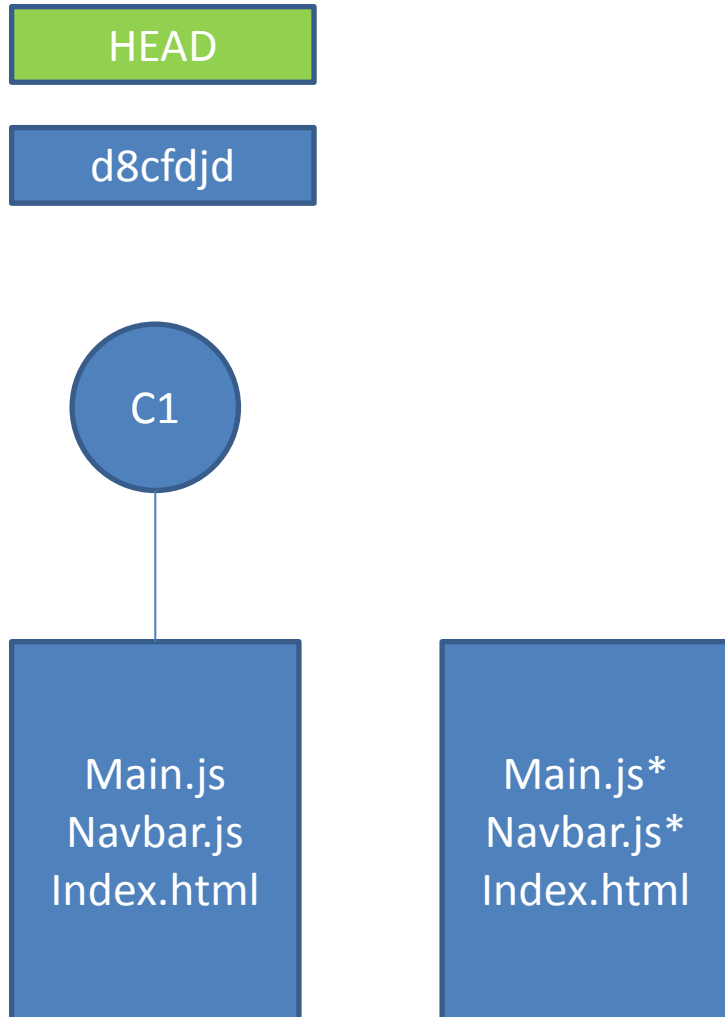
d8cfdjd

C1

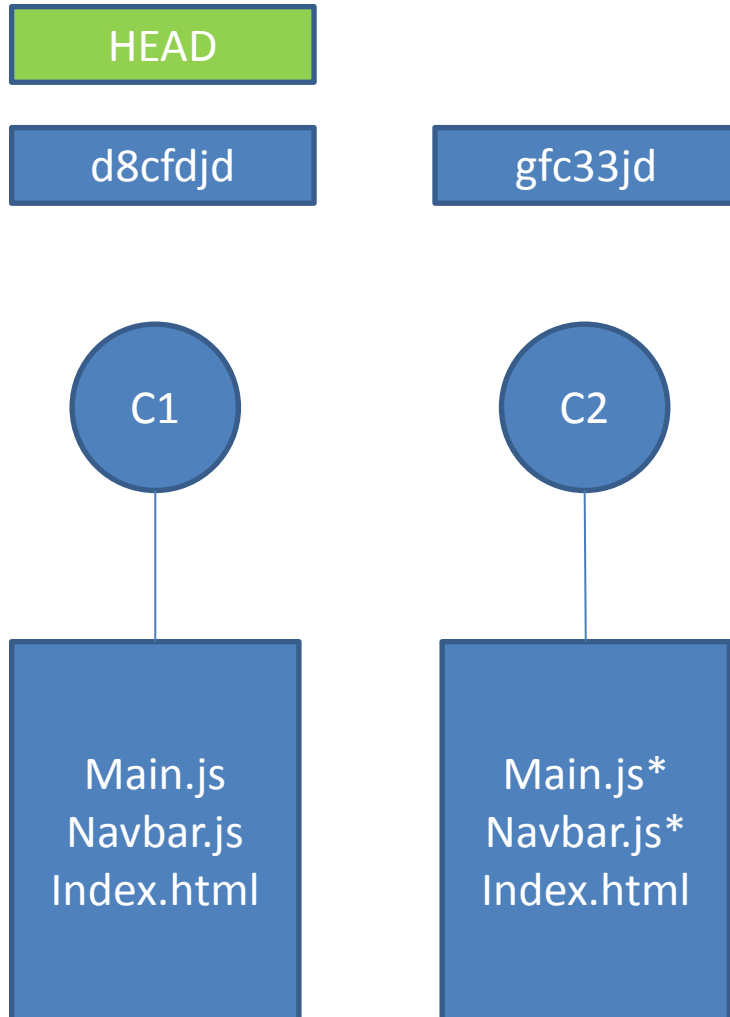
Main.js
Navbar.js
Index.html



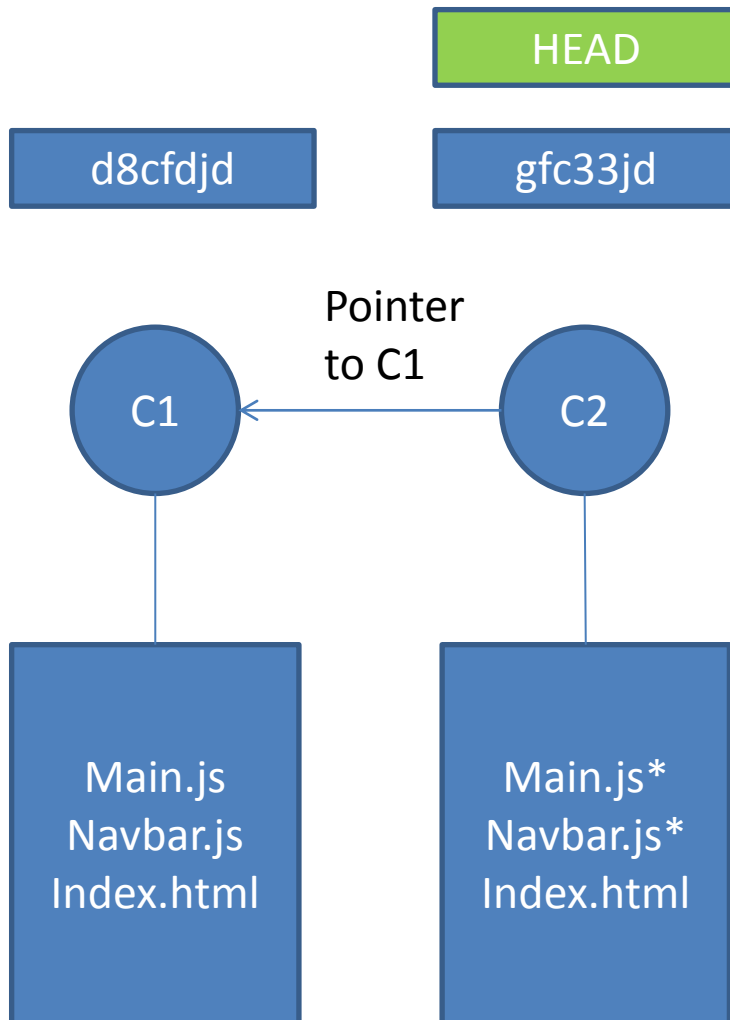
Commits im Detail



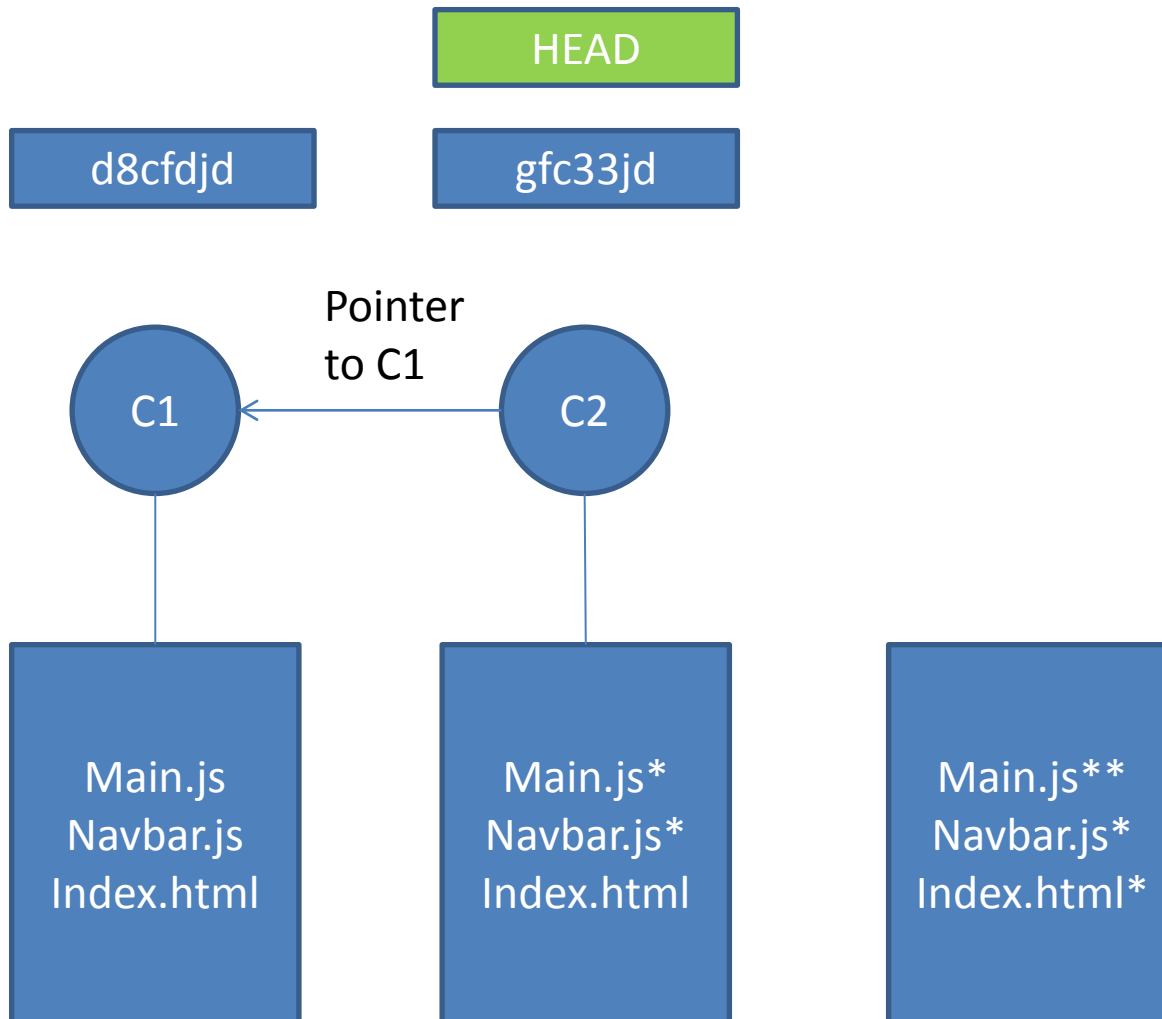
Commits im Detail



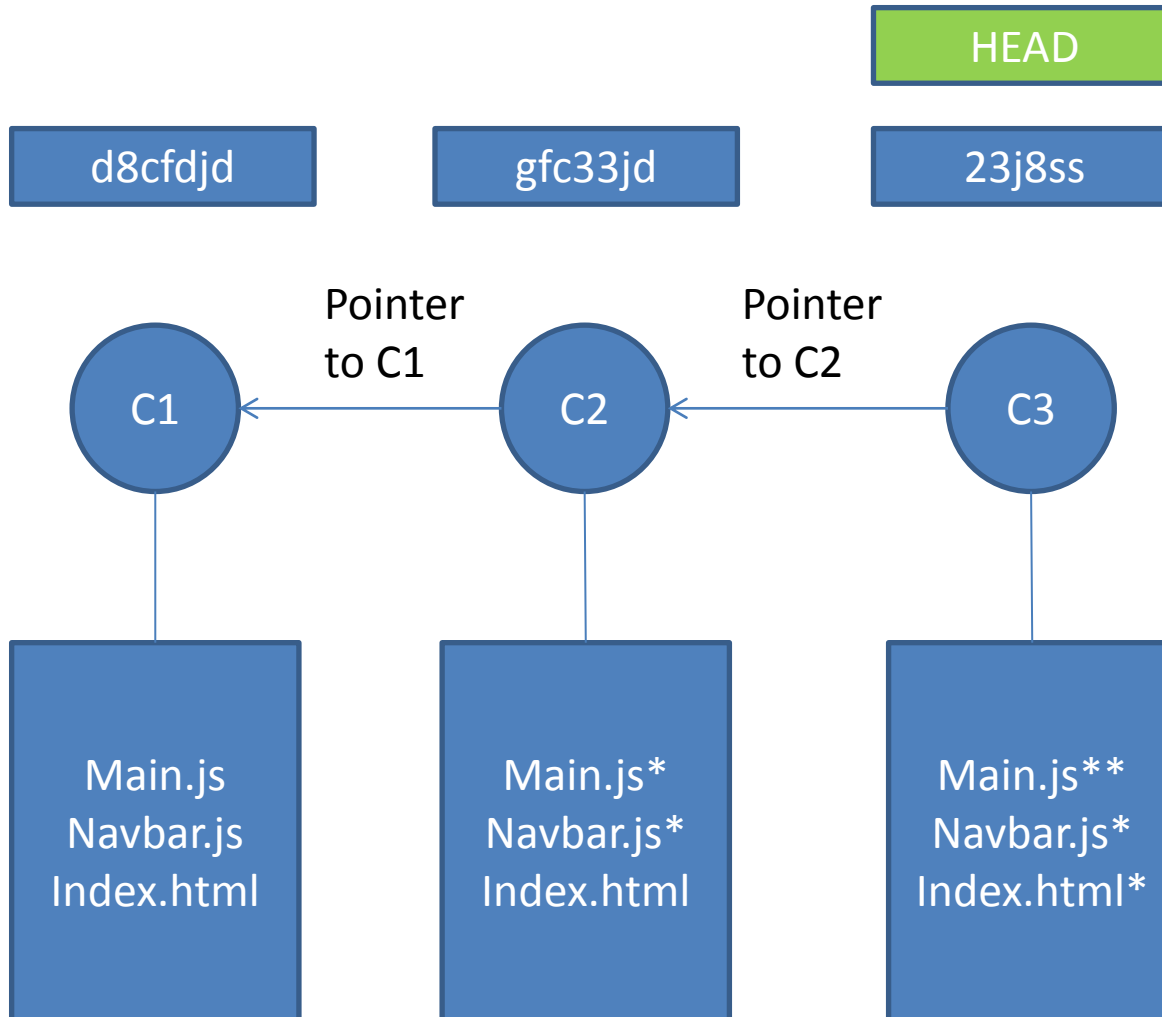
Commits im Detail



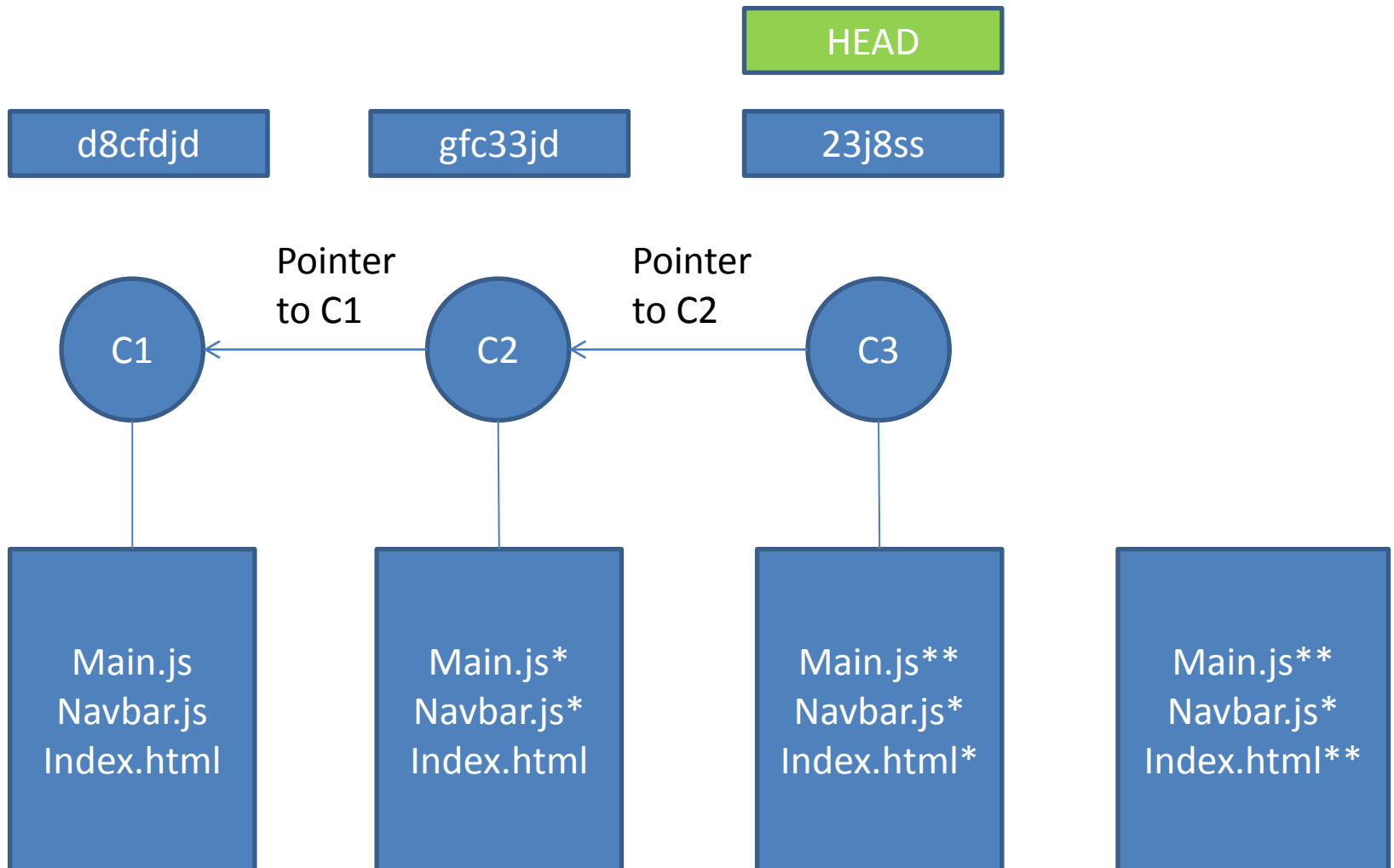
Commits im Detail



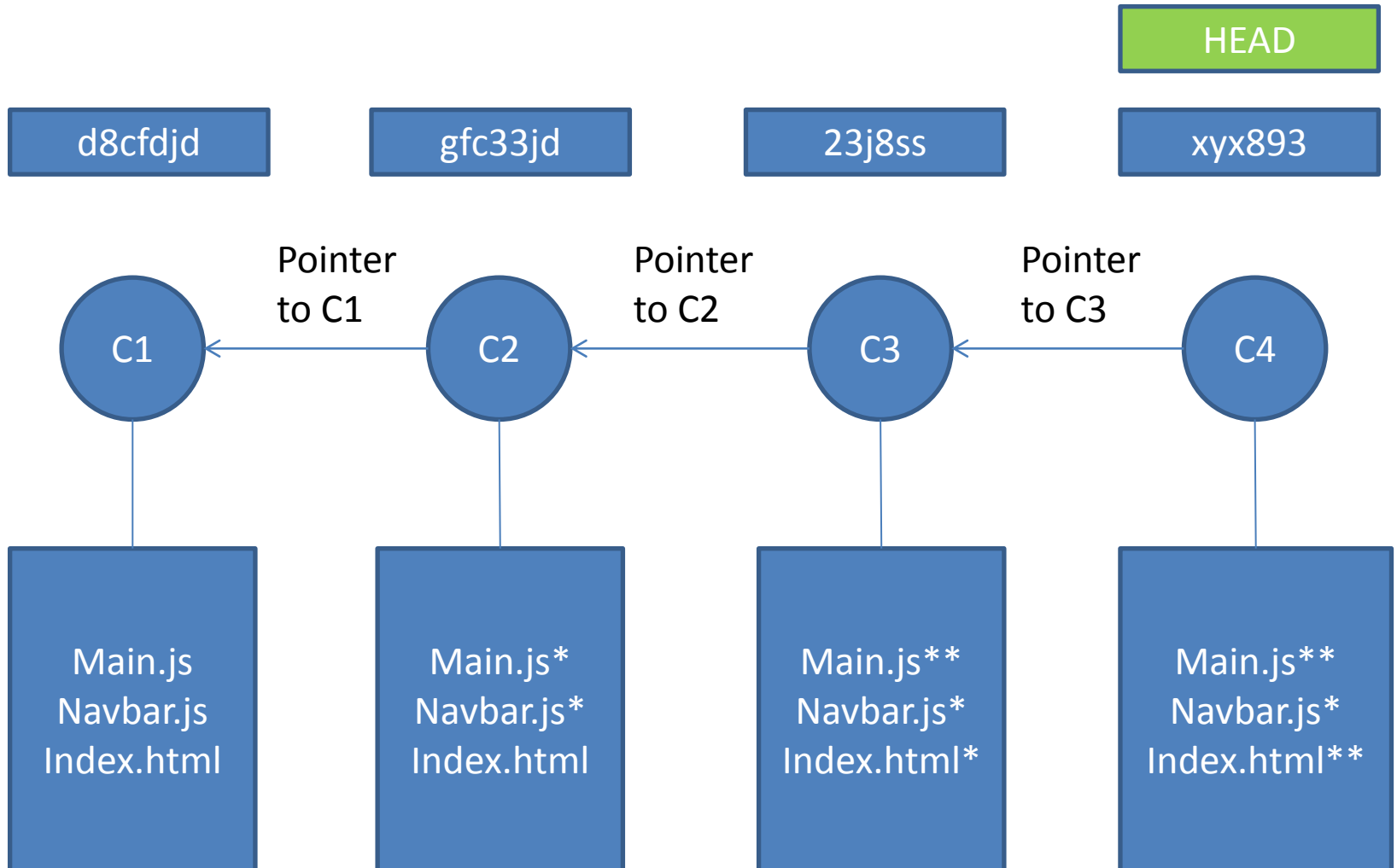
Commits im Detail



Commits im Detail



Commits im Detail



git add

	New Files	Modified Files	Deleted Files
git add -u		X	X
git add .	X	X	
git add -A	X	X	X

Git States



**Working
Directory**

**Staged
Files**

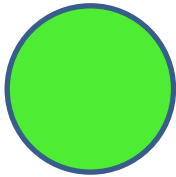
**Committed
Files**

**Remote
Repository**

git diff

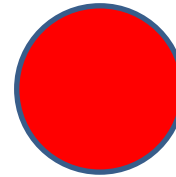


**Working
Directory**



**Staged
Files**

**Committed
Files**

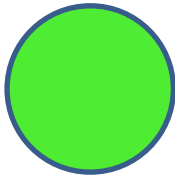


**Remote
Repository**

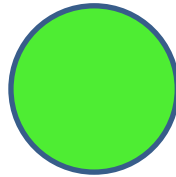
git diff HEAD~1



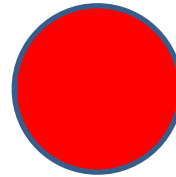
**Working
Directory**



**Staged
Files**



**Committed
Files**



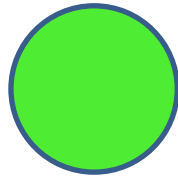
**Remote
Repository**

git diff --staged

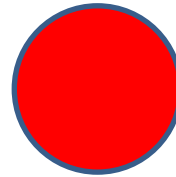


**Working
Directory**

**Staged
Files**



**Committed
Files**



**Remote
Repository**

git diff master origin/master

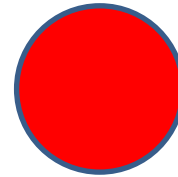
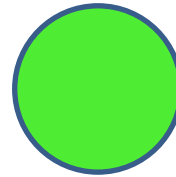


**Working
Directory**

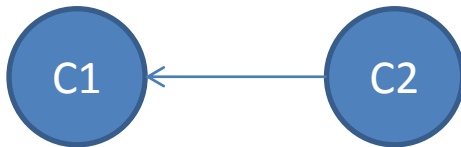
**Staged
Files**

**Committed
Files**

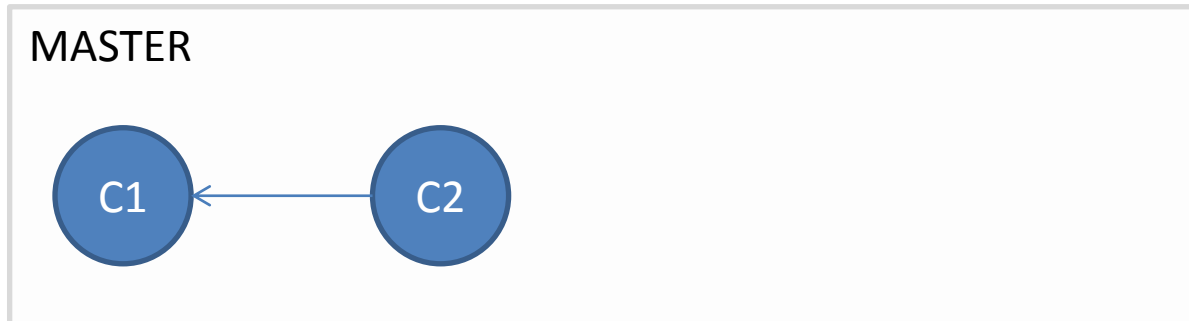
**Remote
Repository**



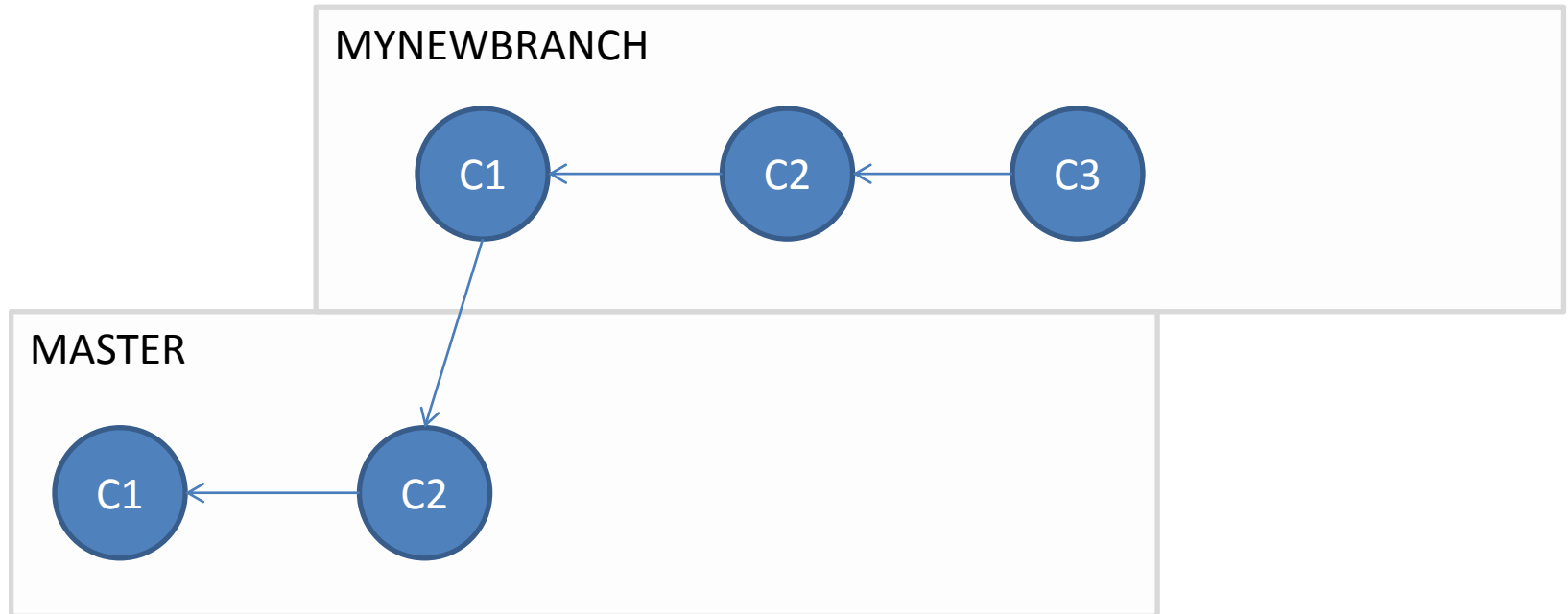
Unsere Branches bis jetzt ...



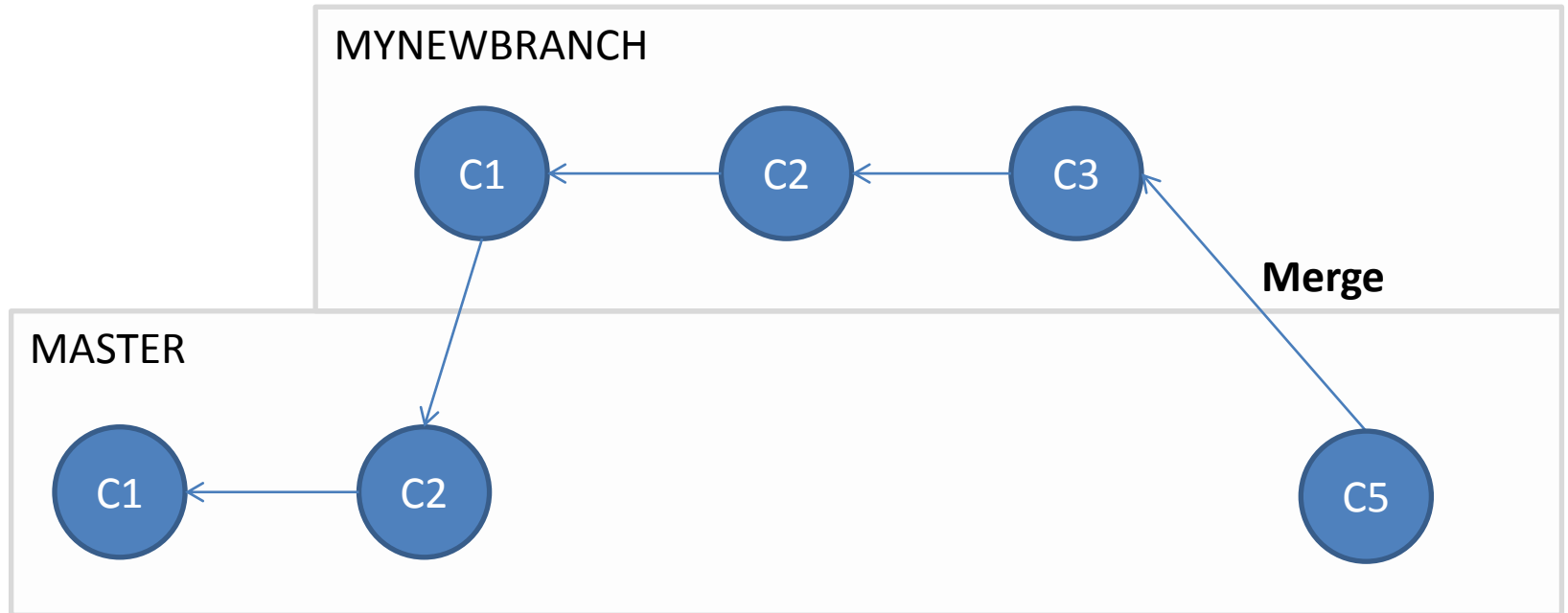
Unsere Branches bis jetzt ...



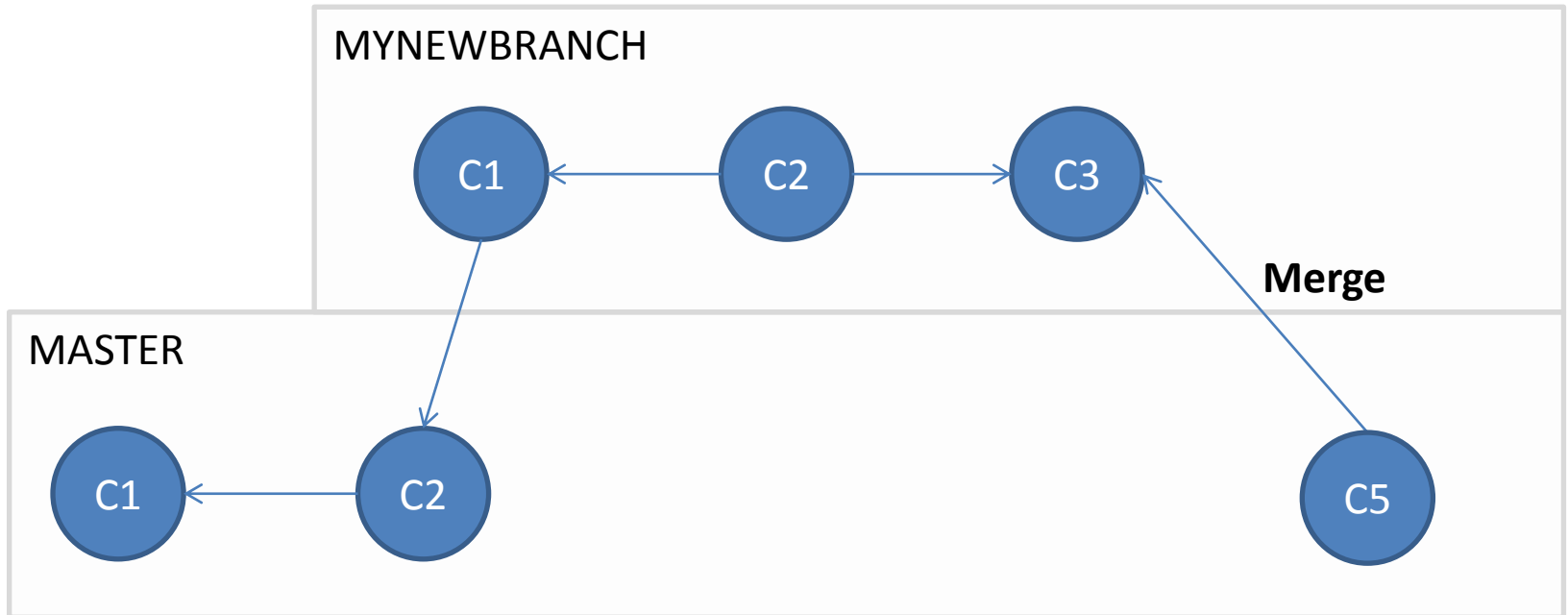
Wir brauchen einen neuen Branch



We need a new branch



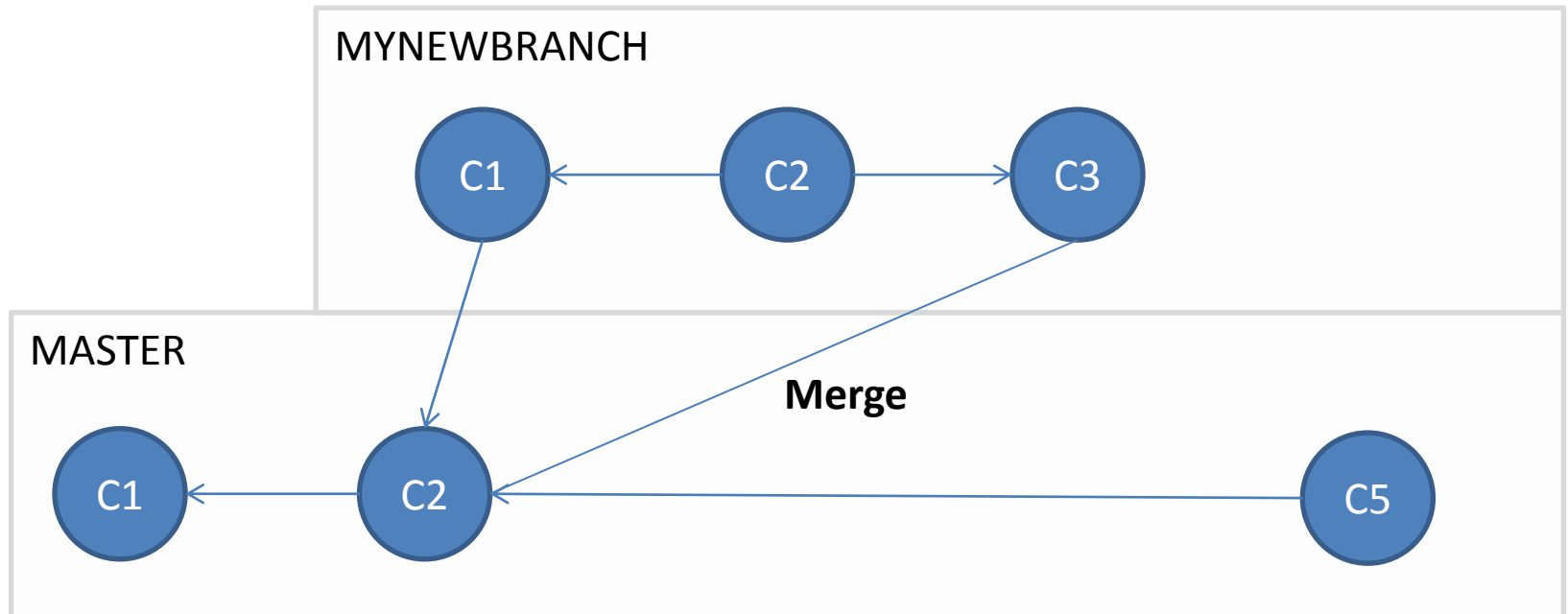
Fast Forward Merges



FAST FORWARD :

- GIT sieht die zwei Branches als einen Branch
- Macht nur Sinn, wenn währenddessen keine Commits auf dem Master-Branch gemacht wurden

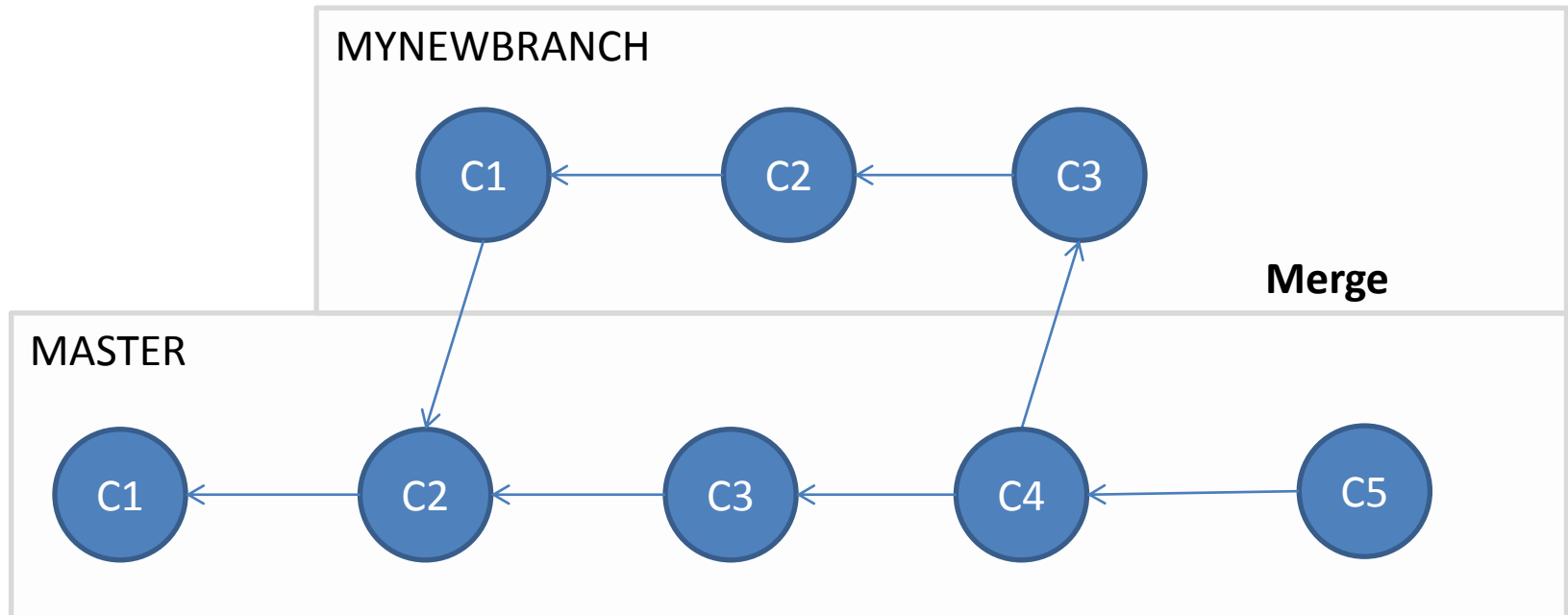
Disabled Fast Forward Merges



Disabled FAST FORWARD :

- C3 merged nach C2

Disable Fast Forward Merges / Automatic Merges



Disabled FAST FORWARD :

- C3 merged nach C4

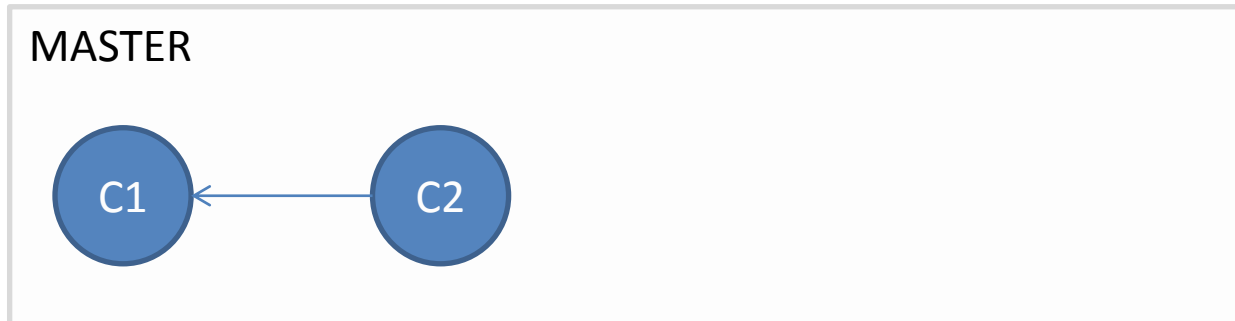
Merge Conflicts

MASTER



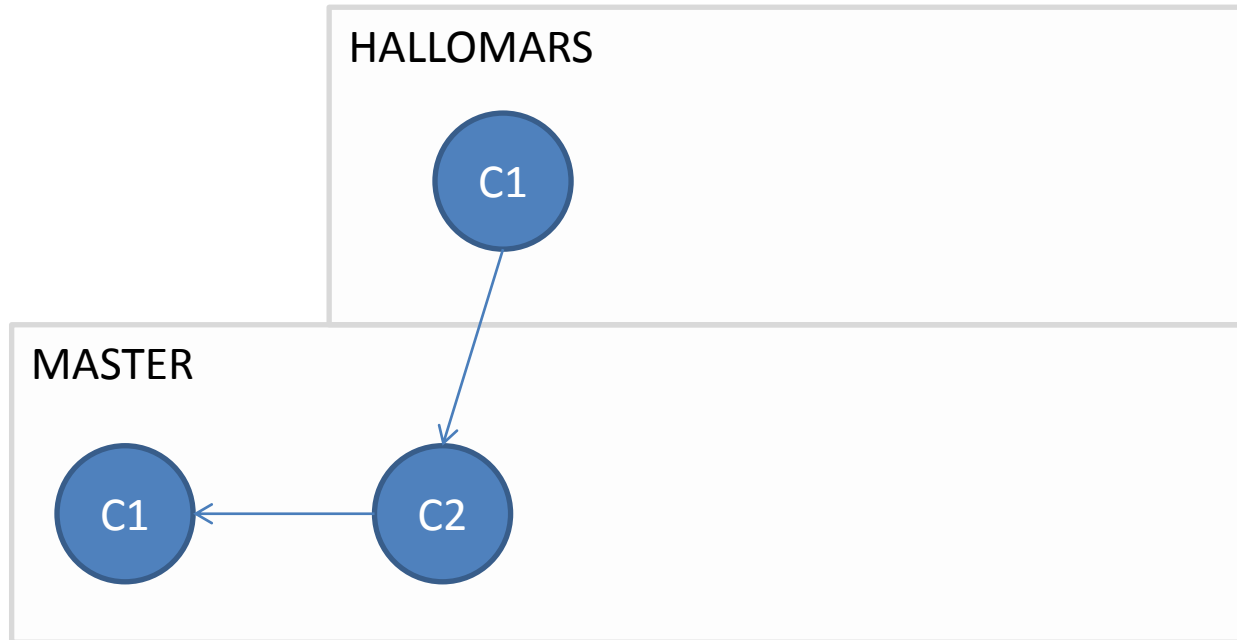
Master C1 -> Index.html wurde erstellt

Merge Conflicts



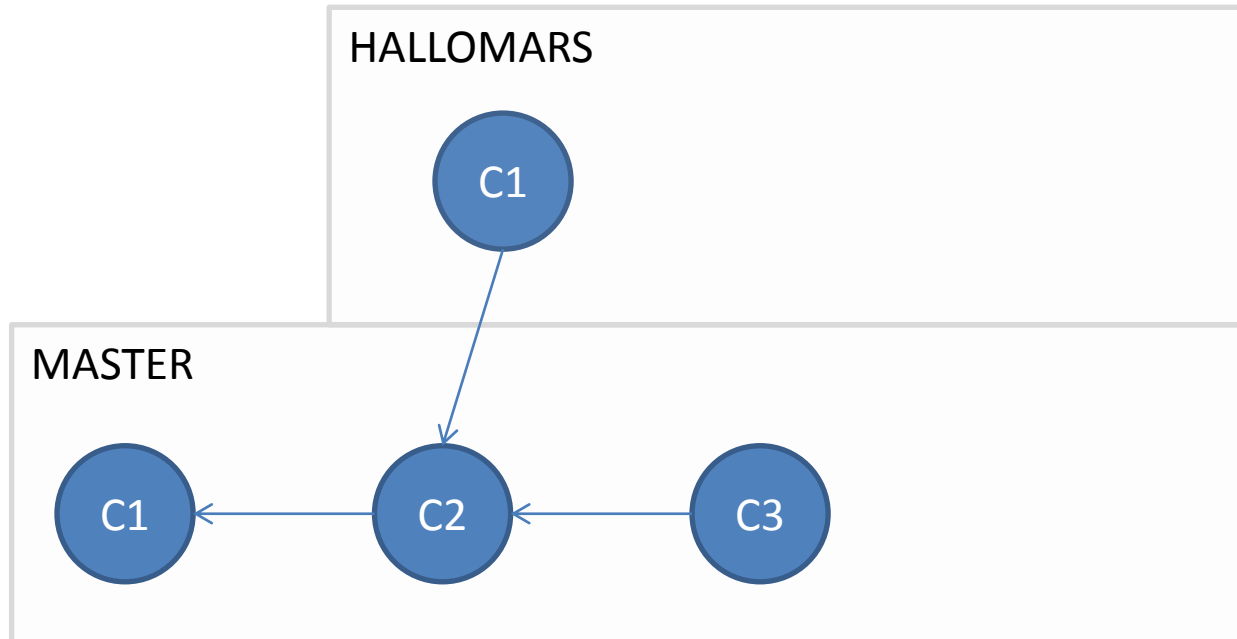
Master C2 -> Wir haben **<h1>Hallo </h1>** nach index.html auf Zeile 1 hinzugefügt

Merge Conflicts



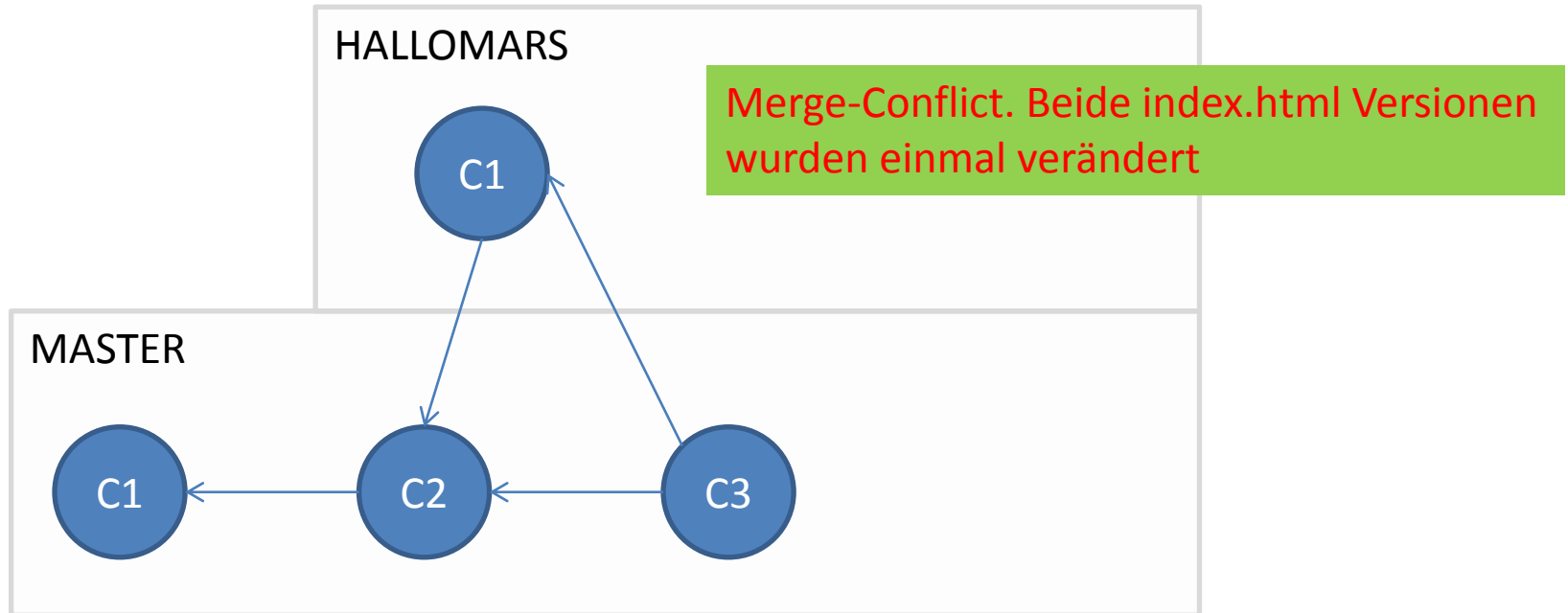
HalloMars C1 -> Wir haben die Zeile verändert zu **<h1>Hallo Mars</h1>**

Merge Conflicts



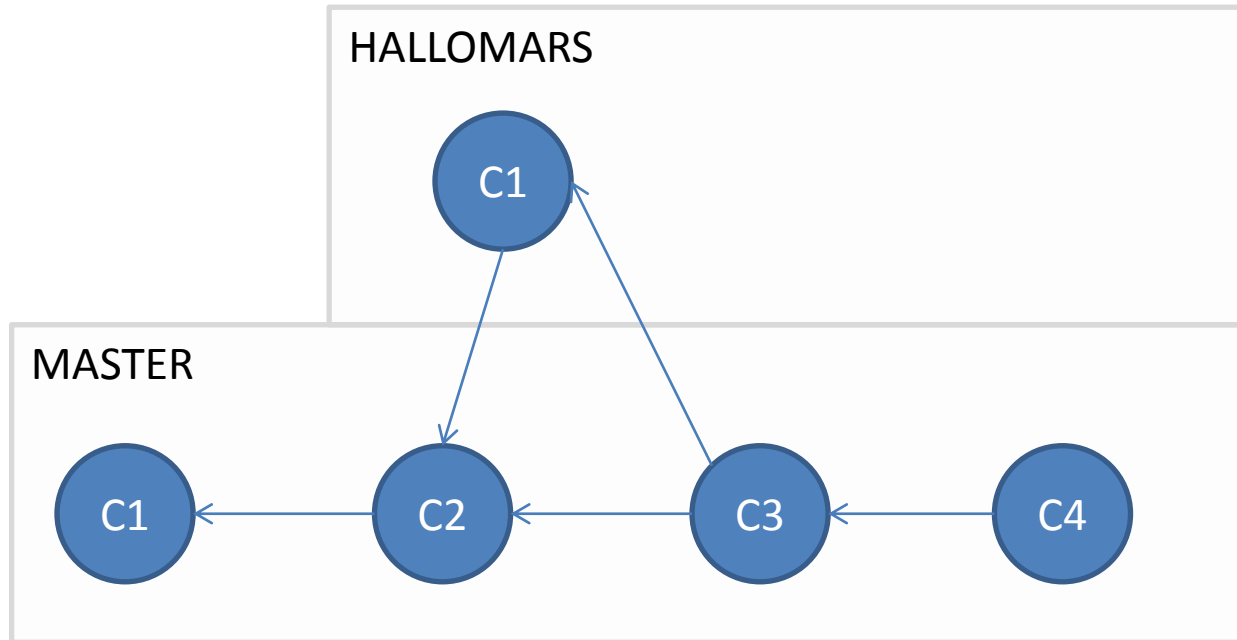
Master C3 -> Wir haben die Zeile verändert zu **<h1>Hallo World</h1>**

Merge Conflicts



Master C4 -> Merge HalloMars C1 nach Master C3

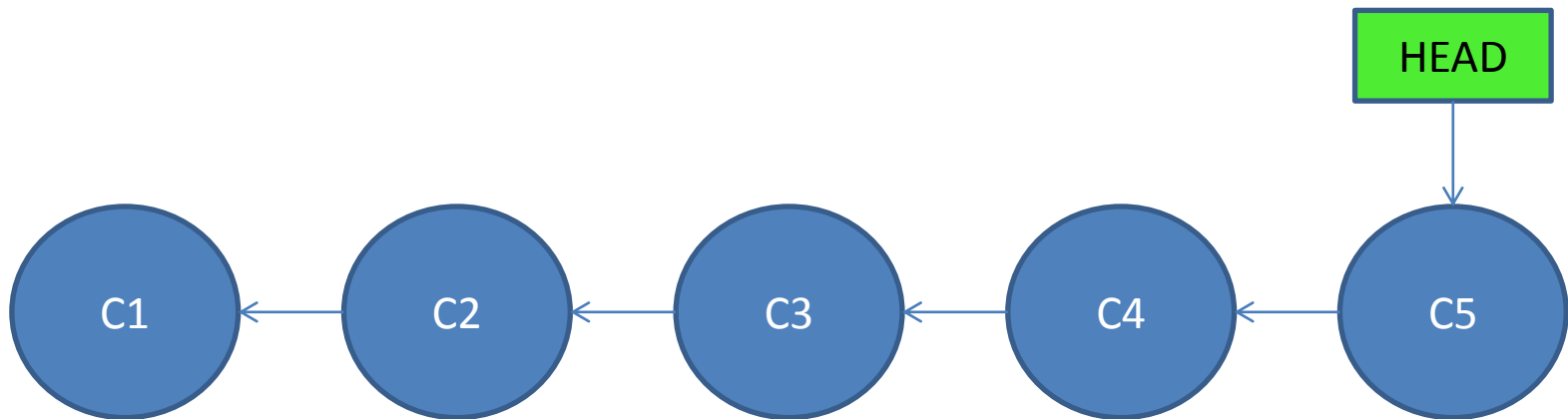
Merge Conflicts



Master C4 -> Entschieden für Master C3 version.

Reverting Changes

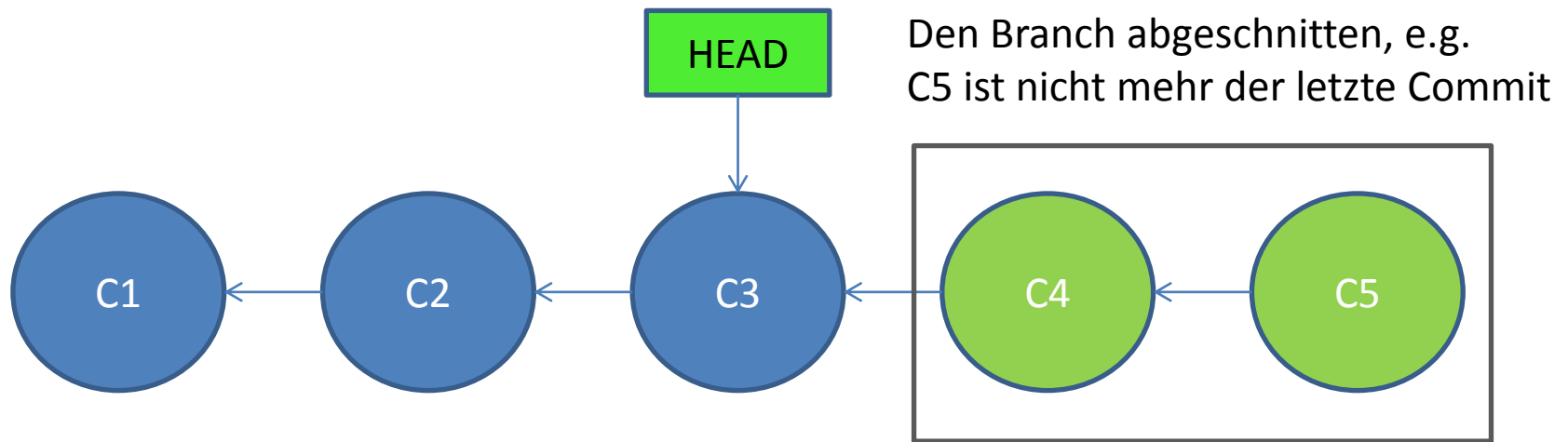
- **Our goal:** Wir wollen zurück zu C3



Reverting Changes

- **Option 1: Hard Reset**

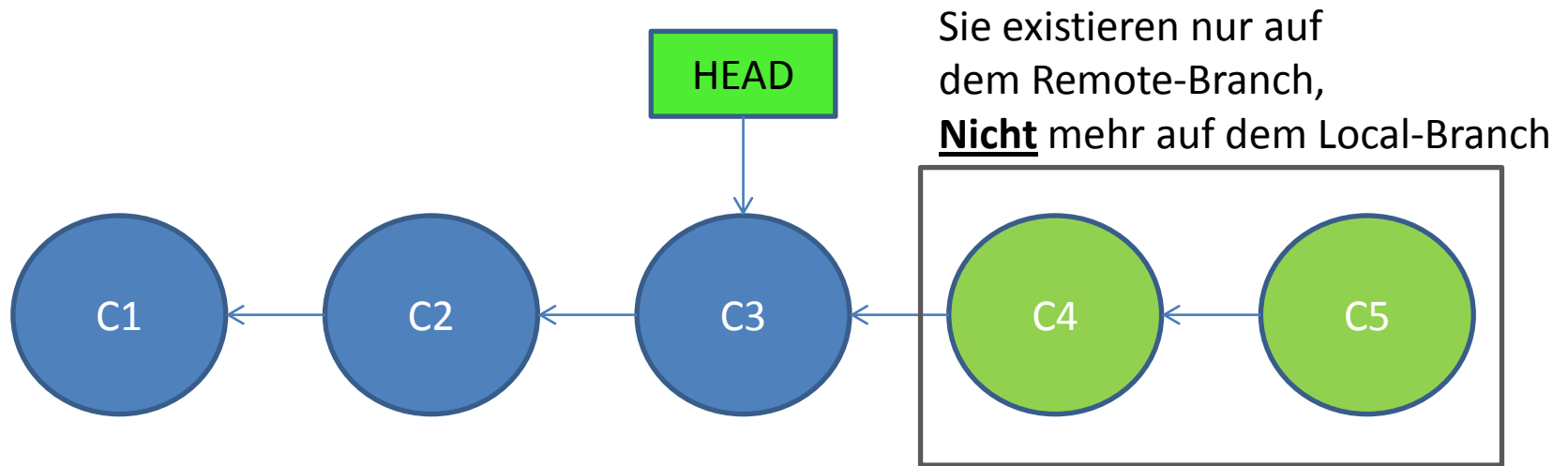
- Einfach den HEAD zurück auf C3 setzen und alle folgenden Commits ignorieren



Reverting Changes

- **Option 1: Hard Reset**

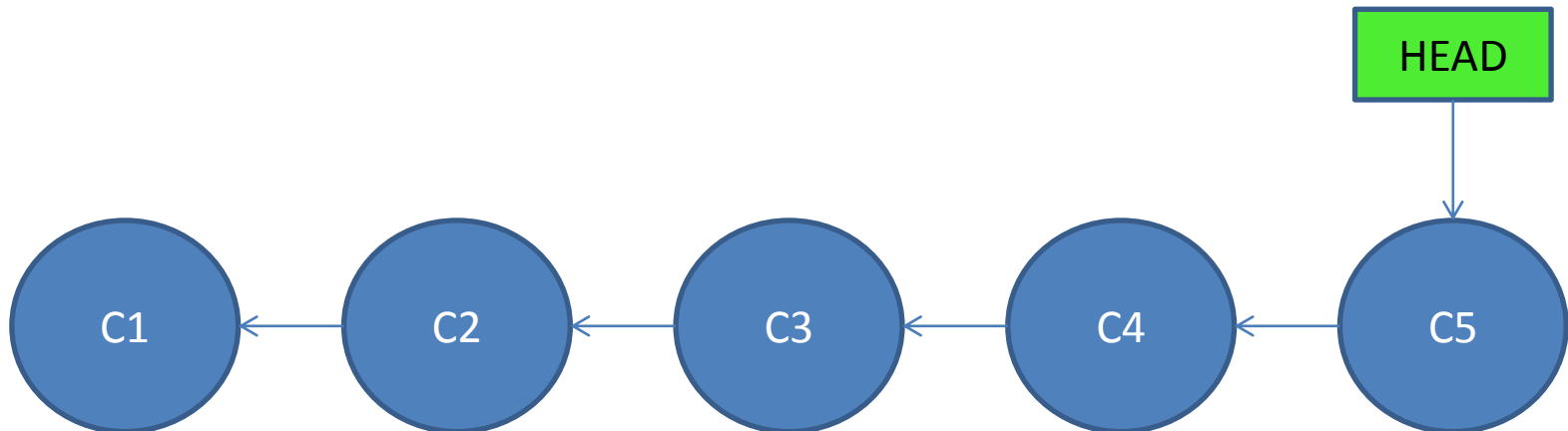
- Einfach den HEAD zurück auf C3 setzen und alle folgenden Commits ignorieren



Problem: Wenn C4 und C5 schon auf dem Remote-Repository existieren, funktioniert Pushen nicht mehr!
Einzige Lösung: das Remote-Repo neu zu erstellen!

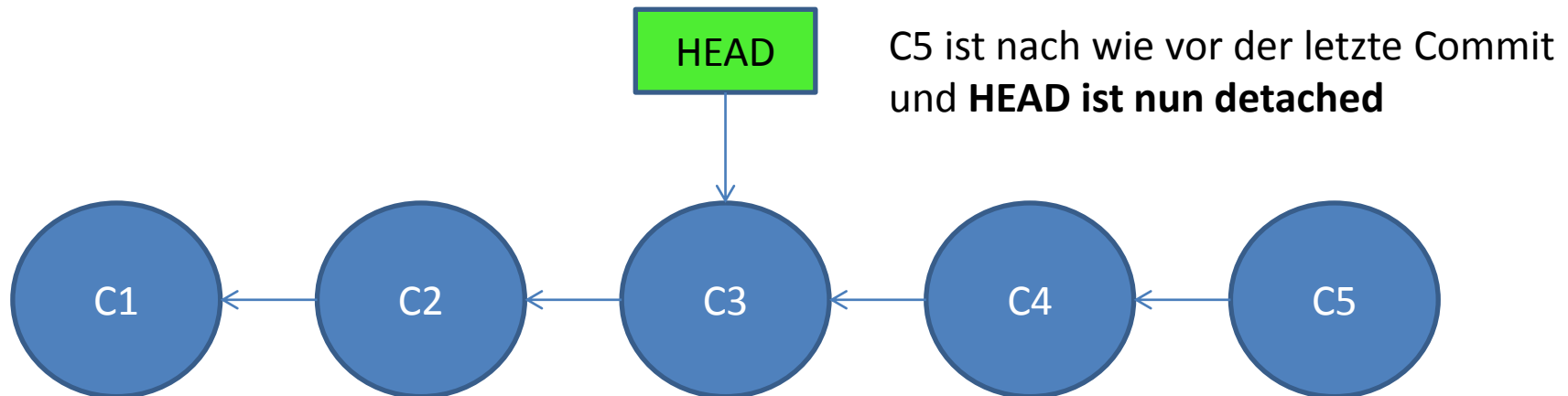
Reverting Changes

- **Option 2: Branching mit Checkout**
 - We checken C3 aus



Reverting Changes

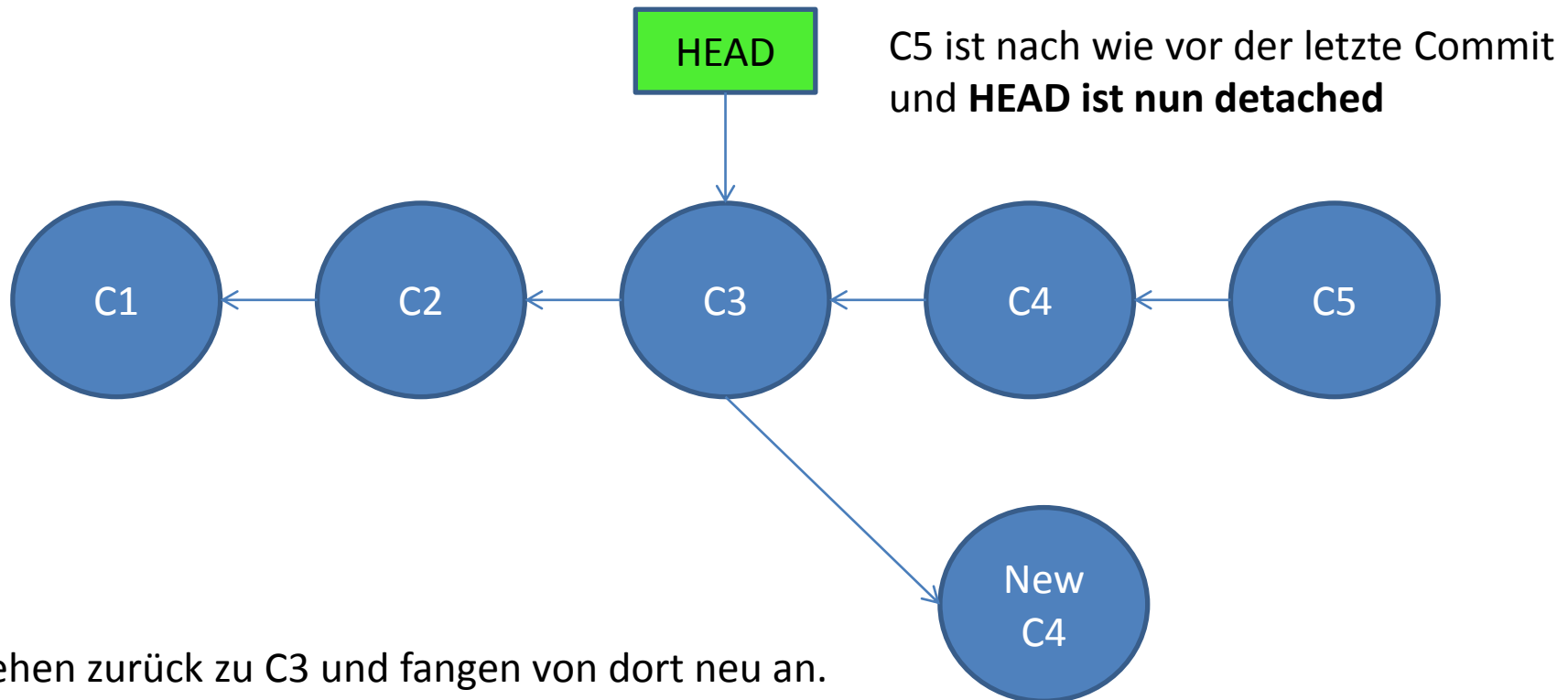
- **Option 2: Branching mit Checkout**
 - We checken C3 aus



Wir gehen zurück zu C3 und fangen von dort neu an.

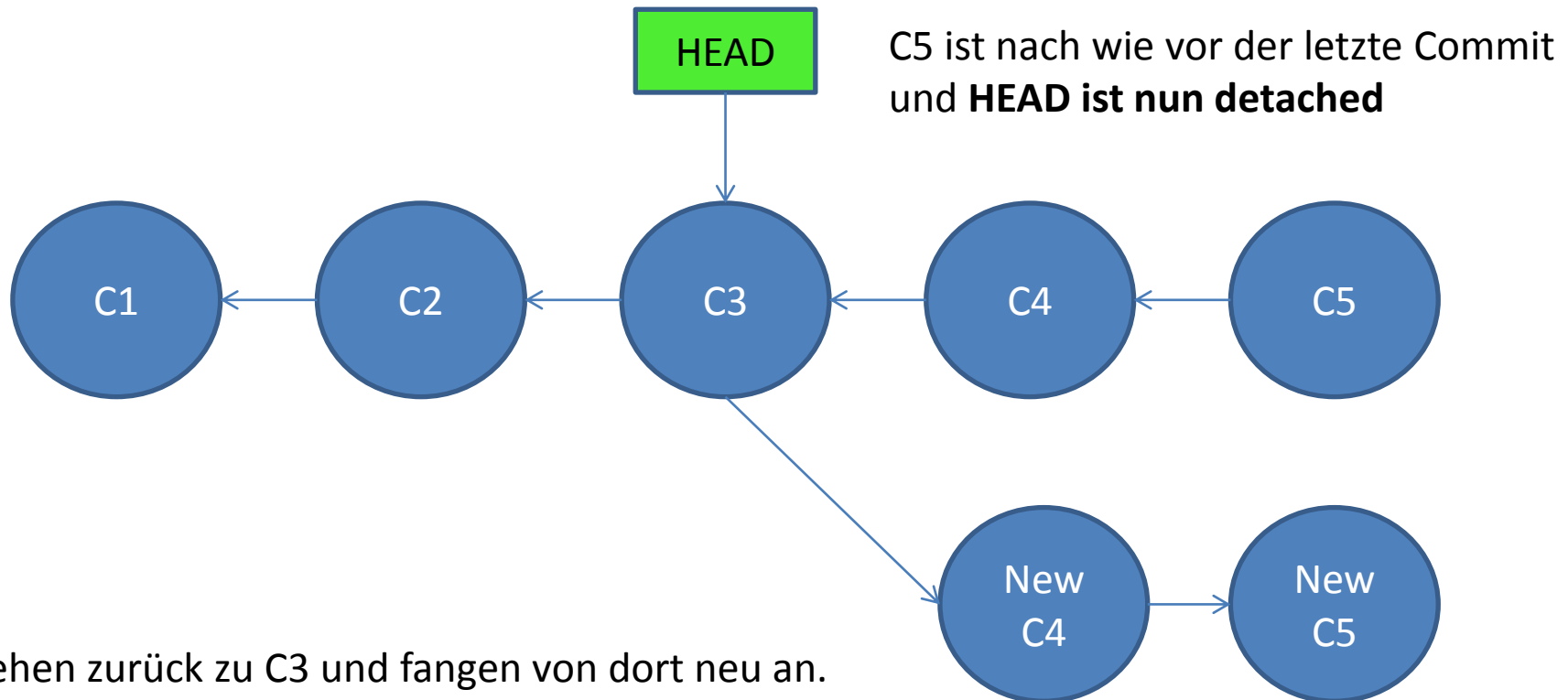
Reverting Changes

- **Option 2: Branching mit Checkout**
 - We checken C3 aus



Reverting Changes

- **Option 2: Branching with Checkout**
 - We checken C3 aus

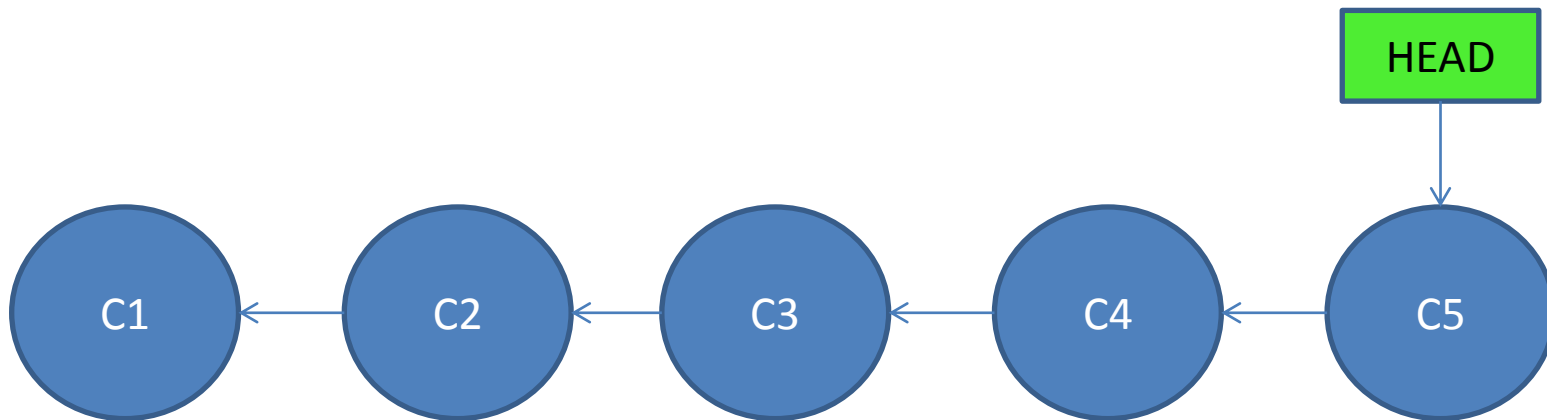


Wir gehen zurück zu C3 und fangen von dort neu an.

Kein Problem beim pushen auf Remote!

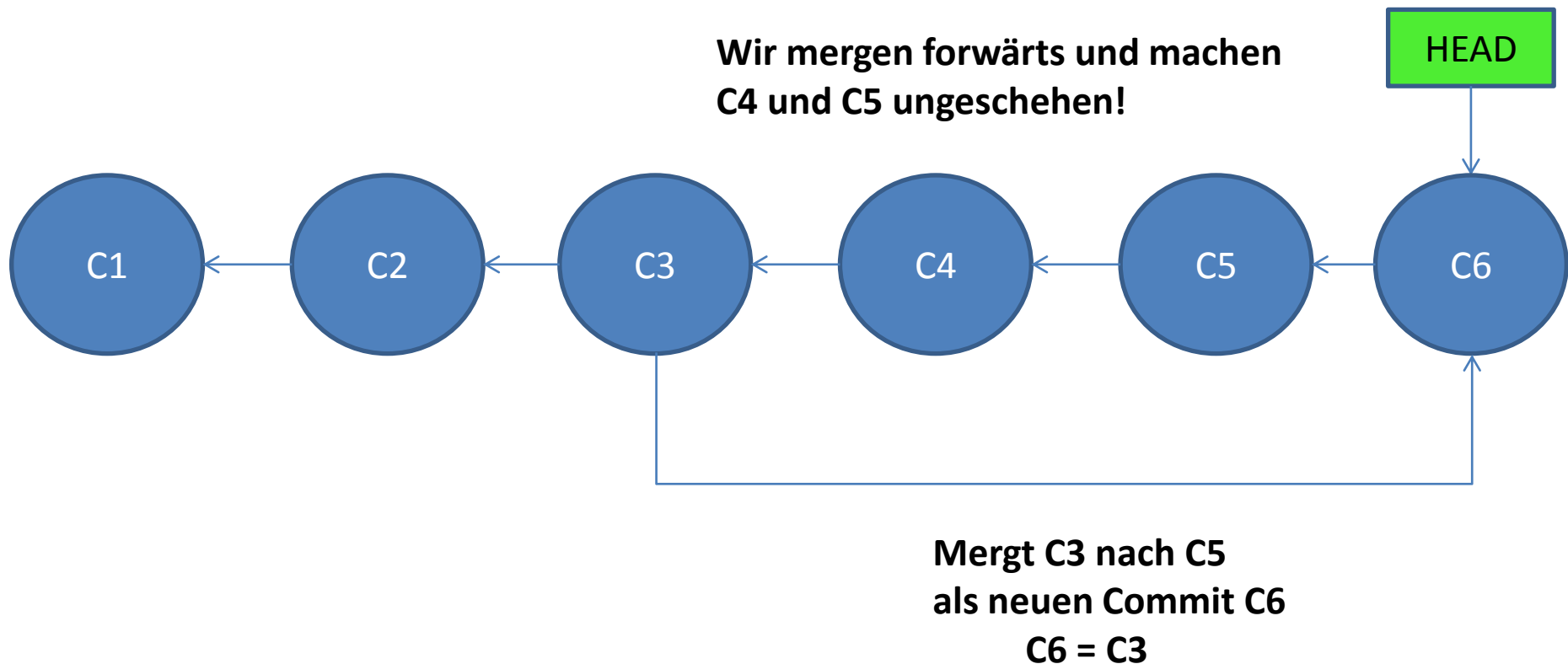
Reverting Changes

- **Option 3: Reverting**



Reverting Changes

- **Option 3: Reverting**



Git Reset, Revert, Checkout

Command	Scope	Common use cases
git reset	Commit-level	Mach uncommittete Änderungen rückgängig im Arbeitsverzeichnis
git reset	File-level	Unstage eine Datei
git checkout	Commit-level	Wechsele einen Branch oder gehe zu einem früheren Snapshot
git checkout	File-level	Mache Änderungen im Arbeitsverzeichnis rückgängig
git revert	Commit-level	Mache Änderungen rückgängig
git revert	File-level	(N/A)

Reverting Changes

- **Unstage files**
 - `git reset <filename>`
 - `git reset .`
- **Undo Changes** in the working directory
 - `git checkout -- <filename>`
 - `git checkout -- .`

7. Remote and Local Repository Commands

LOCAL

COMMIT
RESET
CHECKOUT
ADD
RM
STATUS
DIFF
MERGE

REMOTE

CLONE
FORK
PUSH
PULL