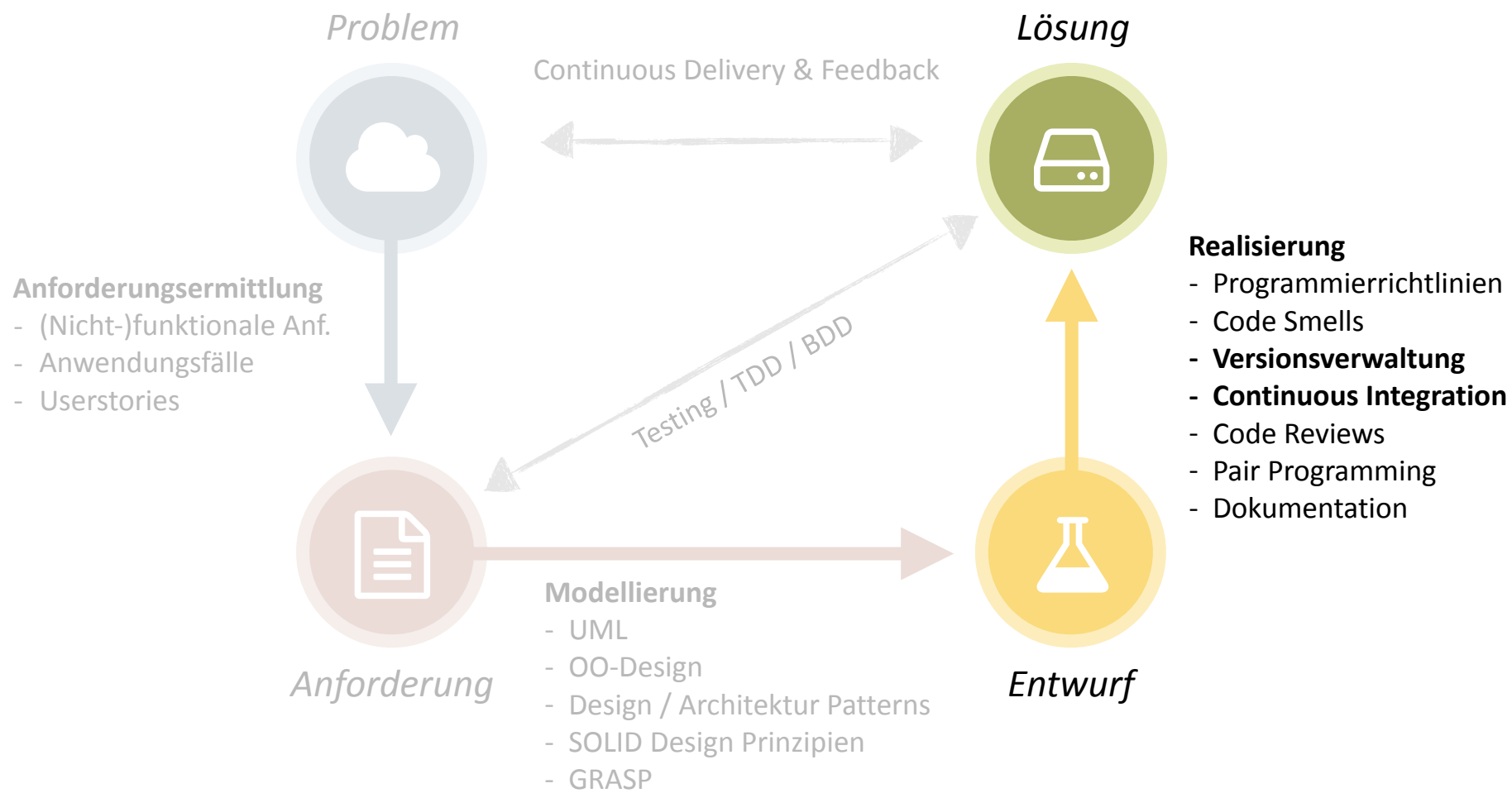
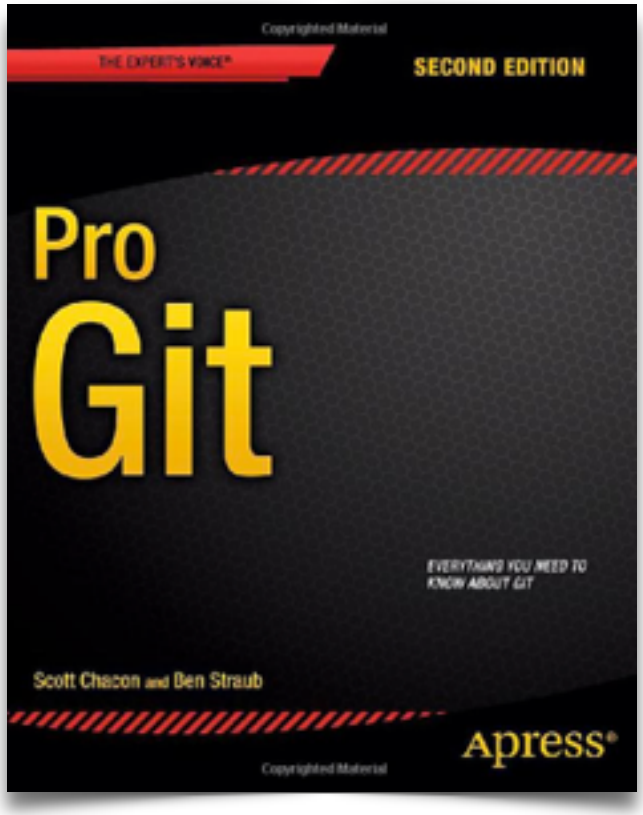
Software Engineering
8. Version Control with Git

Jonathan Brachthäuser

Einordnung: Zusammenfassung



Literaturempfehlung



Online verfügbar unter: https://git-scm.com/book/en/v2
Onlinetutorials auf: https://try.github.io und http://gitreal.codeschool.com/
Außerdem: Übungsblätter zu git sind auf der Vorlesungsseite zu finden

Versionskontrolle

Was ist Versionskontrolle?

- Ein Versionskontrollsystem (VCS) speichert
  - ... den Inhalt aller Dateien eines Projektes
  - ... die gesamte Änderungsgeschichte eines Projektes

  - **Wer** hat **wann** und **warum**, **welche** Dateien **wie** geändert?

Wozu Versionskontrolle?

- Zurücksetzen des Projektes auf einen früheren Stand
- Dokumentation der Projektgeschichte und der Beiträge verschiedener Entwickler
- Parallele Entwicklung an mehreren Entwicklungszweigen
- Koordination der Arbeit verschiedener Entwickler

Begriffe

- **Repository**
  - Änderungsgeschichte und Dateiinhalte für alle Versionen
- **Arbeitskopie**
  - Kopie aller Dateien einer Version
- **Status**
  - Zusammengefasster Unterschied zwischen der Working Copy und dem Repository
- **Diff**
  - Unterschiede der Dateien zwischen Repository und Arbeitskopie oder anderer Version
- **Checkout**
  - Dateiinhalte einer Version aus Repository in Working Copy kopieren
- **Commit**
  - Derzeitigen Stand der Working Copy als neue Version in das Repository übernehmen

Begriffe

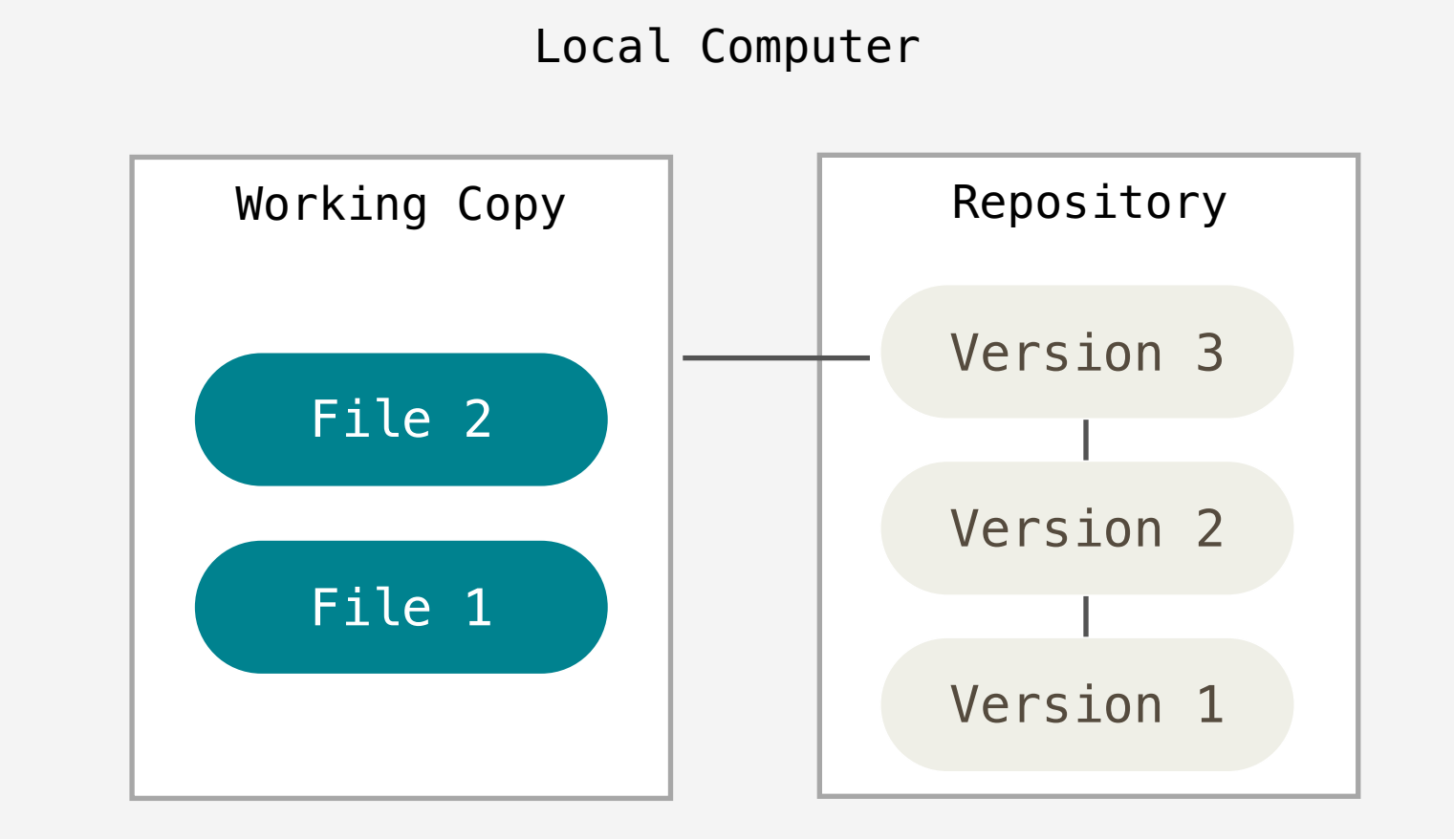
- **Repository**
  - Änderungsgeschichte und Dateiinhalte für alle Versionen
- **Arbeitskopie**
  - Kopie aller Dateien einer Version
- **Status**
  - Zusammengefasster Unterschied zwischen der Working Copy und dem Repository
- **Diff**
  - Unterschiede der Dateien zwischen Repository und Arbeitskopie oder anderer Version
- **Checkout**
  - Dateiinhalte einer Version aus Repository in Working Copy kopieren
- **Commit**
  - Derzeitigen Stand der Working Copy als neue Version in das Repository übernehmen

Begriffe

- **Repository**
  - Änderungsgeschichte und Dateiinhalte für alle Versionen
- **Arbeitskopie**
  - Kopie aller Dateien einer Version
- **Status**
  - Zusammengefasster Unterschied zwischen der Working Copy und dem Repository
- **Diff**
  - Unterschiede der Dateien zwischen Repository und Arbeitskopie oder anderer Version
- **Checkout**
  - Dateiinhalte einer Version aus Repository in Working Copy kopieren
- **Commit**
  - Derzeitigen Stand der Working Copy als neue Version in das Repository übernehmen

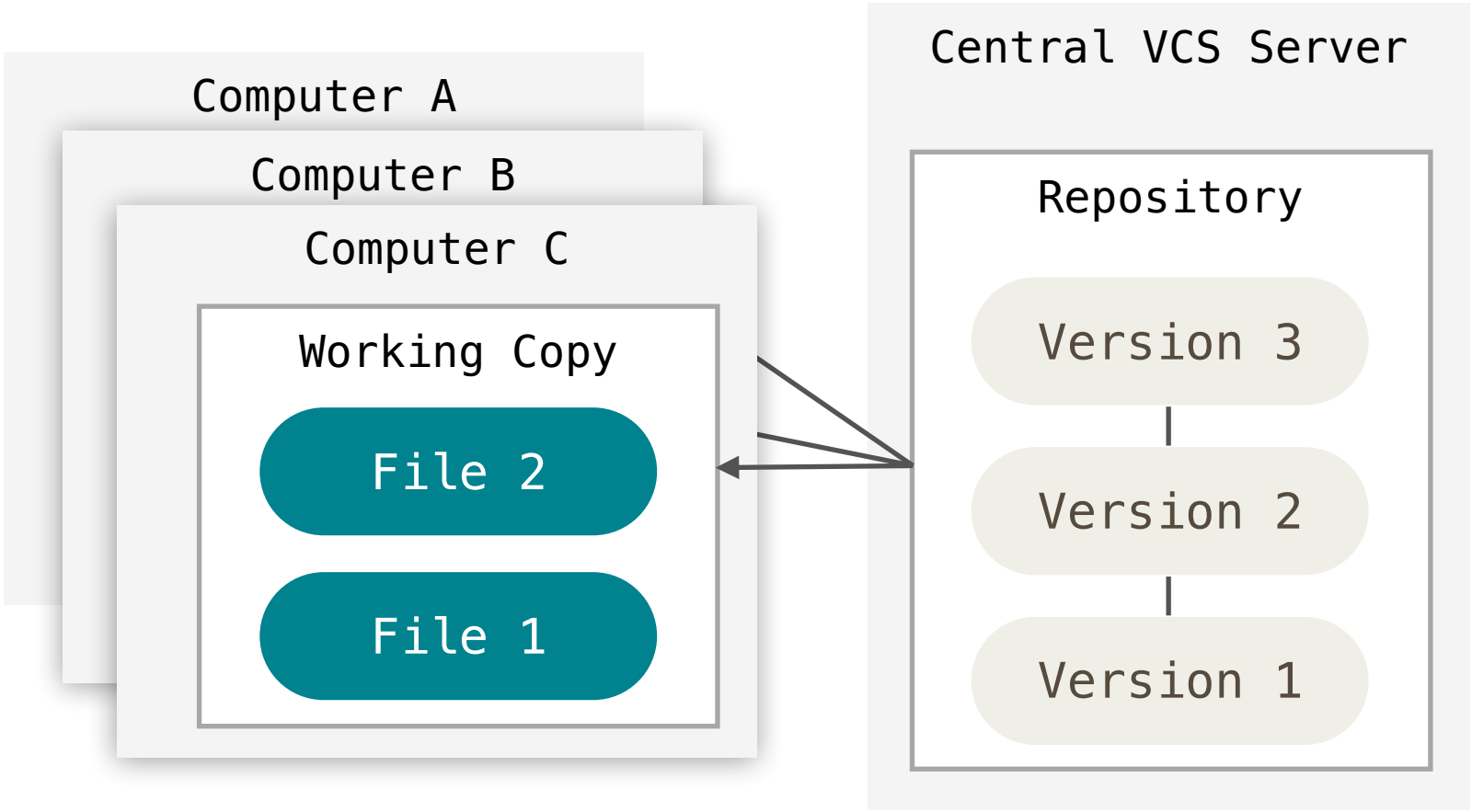
Wo ist das Repository gespeichert?

- Lokale Versionskontrollsysteme



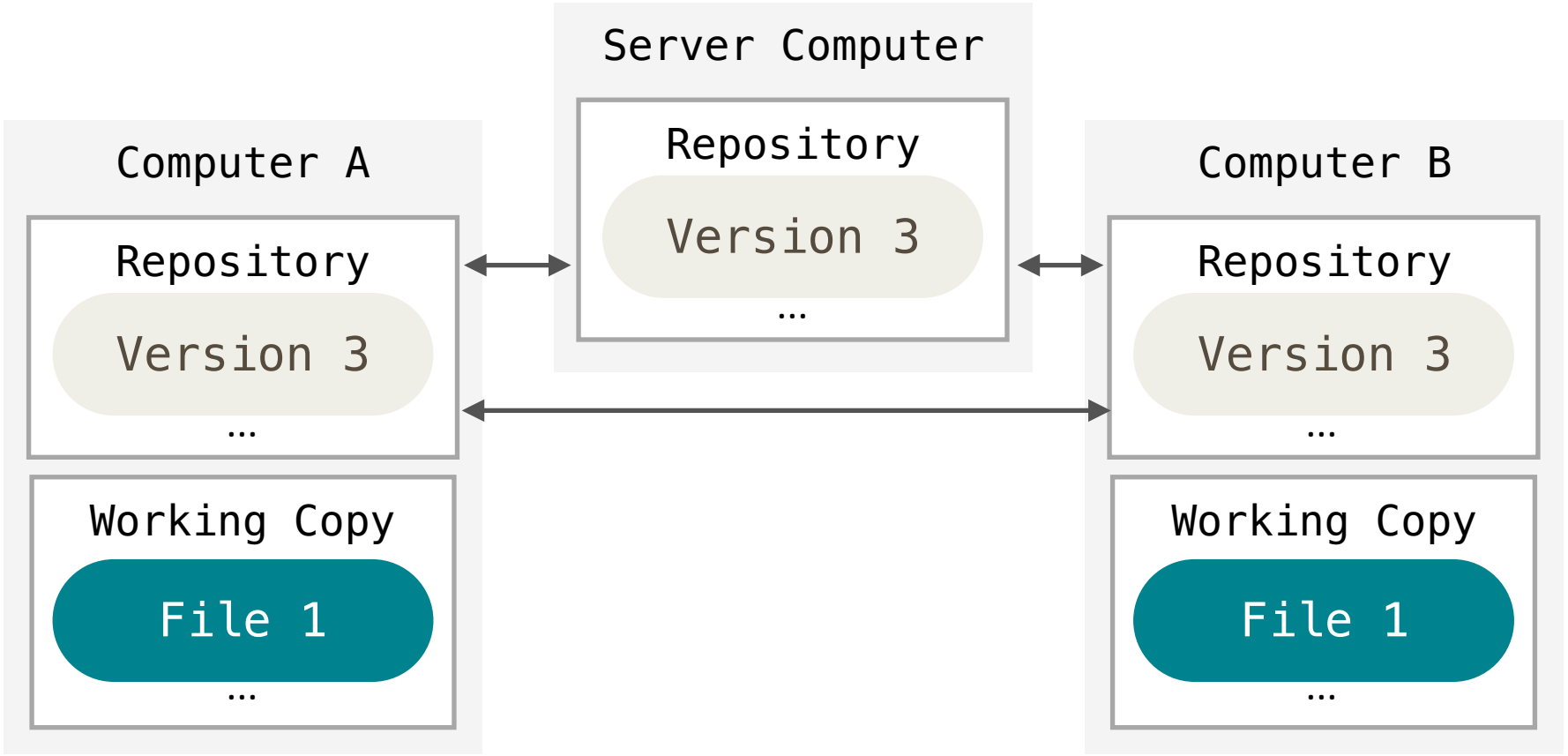
Wo ist das Repository gespeichert?

- Zentrale Versionskontrollsysteme



Wo ist das Repository gespeichert?

- Verteilte Versionskontrollsysteme



Begriffe

- **Repository**
  - Änderungsgeschichte und Dateiinhalte für alle Versionen
- **Arbeitskopie**
  - Kopie aller Dateien einer Version
- **Status**
  - Zusammengefasster Unterschied zwischen der Working Copy und dem Repository
- **Diff**
  - Unterschiede der Dateien zwischen Repository und Arbeitskopie oder anderer Version
- **Checkout**
  - Dateiinhalte einer Version aus Repository in Working Copy kopieren
- **Commit**
  - Derzeitigen Stand der Working Copy als neue Version in das Repository übernehmen

Begriffe

- **Repository**
  - Änderungsgeschichte und Dateiinhalte für alle Versionen
- **Arbeitskopie**
  - Kopie aller Dateien einer Version
- **Status**
  - Zusammengefasster Unterschied zwischen der Working Copy und dem Repository
- **Diff**
  - Unterschiede der Dateien zwischen Repository und Arbeitskopie oder anderer Version
- **Checkout**
  - Dateiinhalte einer Version aus Repository in Working Copy kopieren
- **Commit**
  - Derzeitigen Stand der Working Copy als neue Version in das Repository übernehmen

Diffs

```
answer = 7
factor = 3
while (answer < 60) {
  answer *= factor
  factor += 1
}
```

vorher

```
answer = 7
factor = 2
while (answer < 40) {
  answer *= factor
  factor += 1
}
```

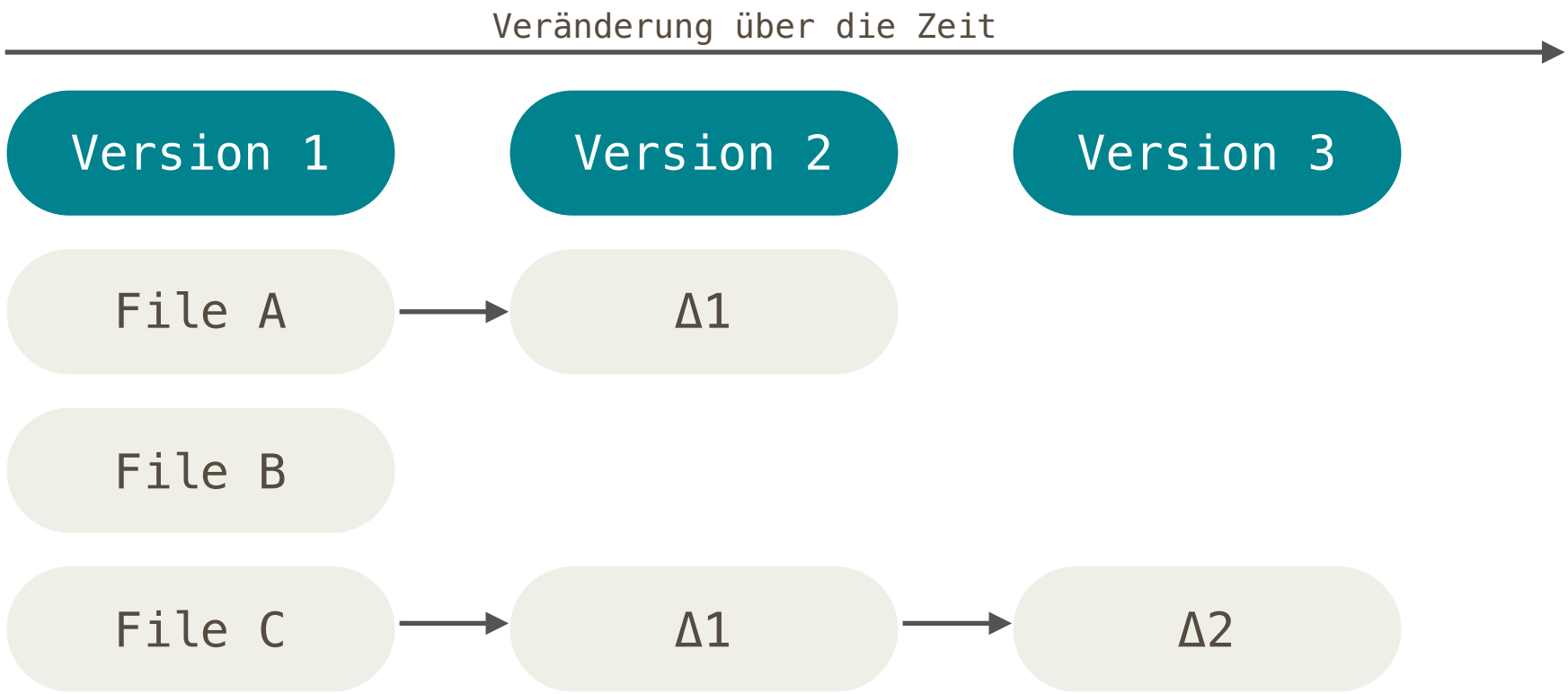
nachher

```
 answer = 7
-factor = 3
-while (answer < 60) {
+factor = 2
+while (answer < 40) {
  answer *= factor
  factor += 1
}
```

diff

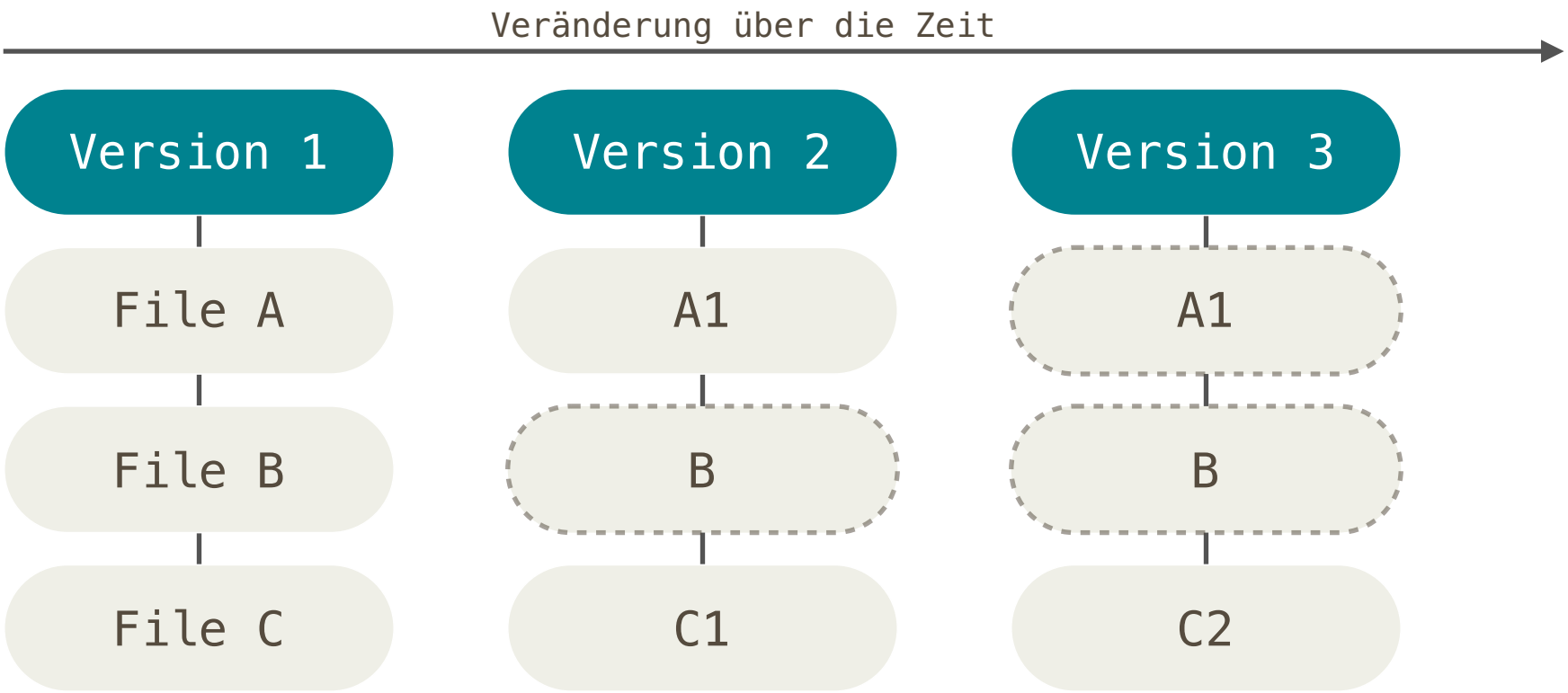
Was ist im Repository gespeichert?

- Diff basierte Versionskontrollsysteme



Was ist im Repository gespeichert?

- Snapshot basierte Versionskontrollsysteme



Begriffe

- **Repository**
  - Änderungsgeschichte und Dateiinhalte für alle Versionen
- **Arbeitskopie**
  - Kopie aller Dateien einer Version
- **Status**
  - Zusammengefasster Unterschied zwischen der Working Copy und dem Repository
- **Diff**
  - Unterschiede der Dateien zwischen Repository und Arbeitskopie oder anderer Version
- **Checkout**
  - Dateiinhalte einer Version aus Repository in Working Copy kopieren
- **Commit**
  - Derzeitigen Stand der Working Copy als neue Version in das Repository übernehmen

Begriffe

- **Repository**
  - Änderungsgeschichte und Dateiinhalte für alle Versionen
- **Arbeitskopie**
  - Kopie aller Dateien einer Version
- **Status**
  - Zusammengefasster Unterschied zwischen der Working Copy und dem Repository
- **Diff**
  - Unterschiede der Dateien zwischen Repository und Arbeitskopie oder anderer Version
- **Checkout**
  - Dateiinhalte einer Version aus Repository in Working Copy kopieren
- **Commit**
  - Derzeitigen Stand der Working Copy als neue Version in das Repository übernehmen

git

Was ist git?

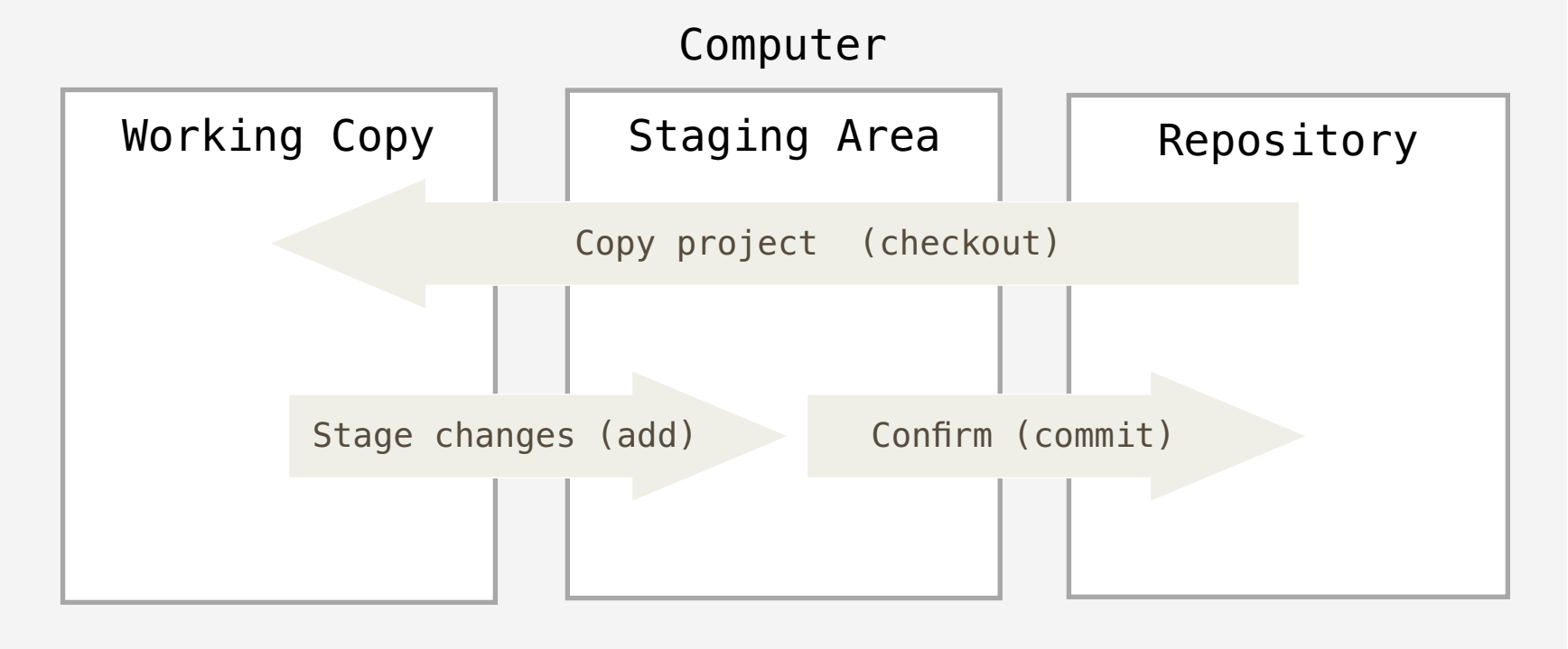
- Verteiltes Versionskontrollsystem
- Ursprünglich von Linus Torvalds zur Versionskontrolle des Linux Kernels entwickelt
- Heute weit verbreitet in Open Source, als auch zur proprietären Entwicklung

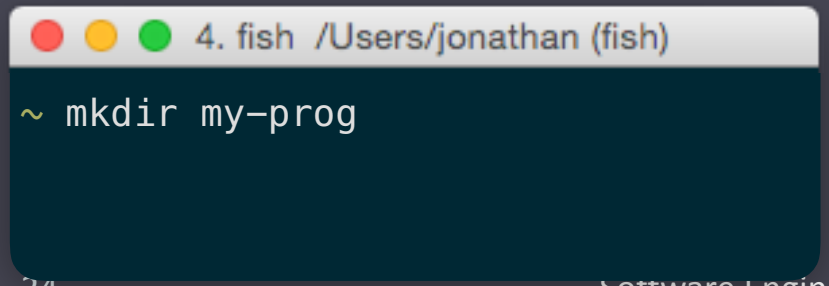
Wie funktioniert git?

- Verteilt
- Snapshot-basiert
- Effiziente Speicherung durch Hash-Adressierung
- Index
- Viele Kommandozeilenbefehle

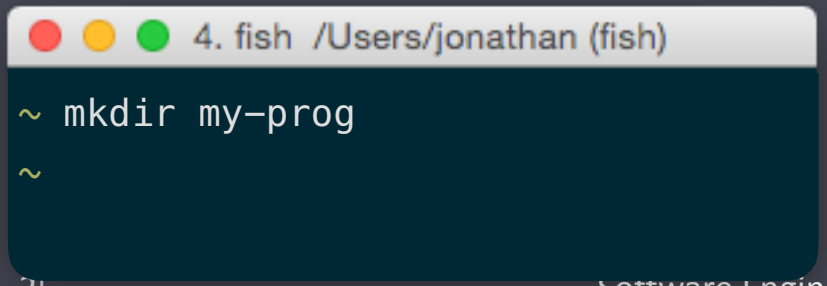
Drei Schritte in Git

- Git unterscheidet zwischen Working Copy, Staging Area und Repository

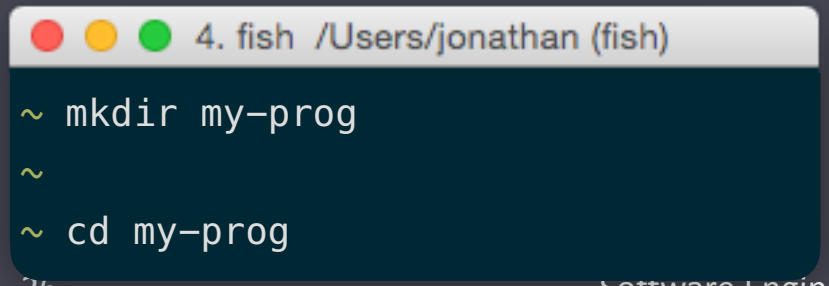



4. fish /Users/jonathan (fish)
~ mkdir my-prog

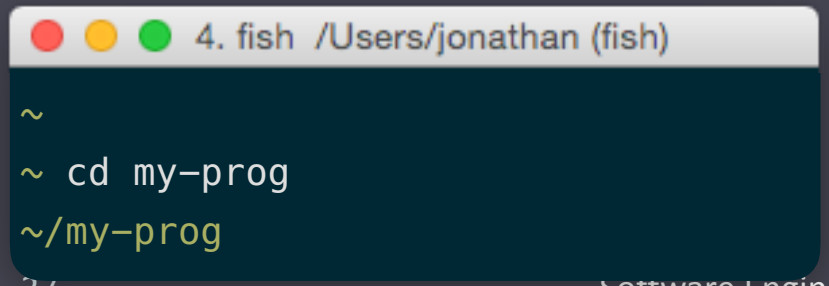
my-prog



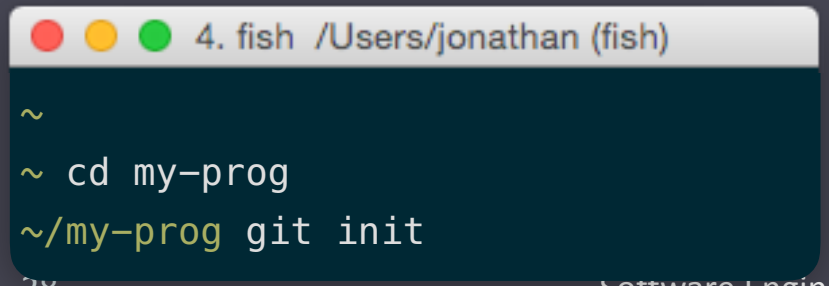
my-prog



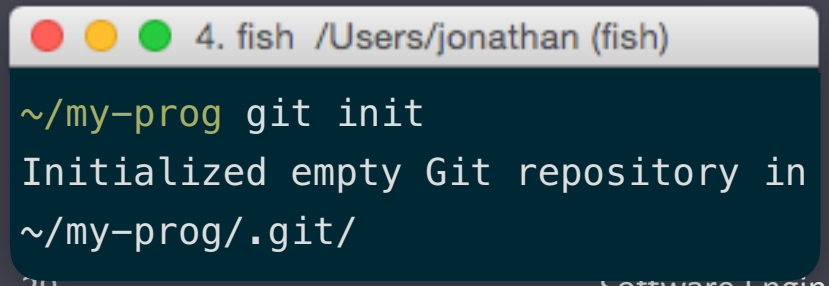
my-prog

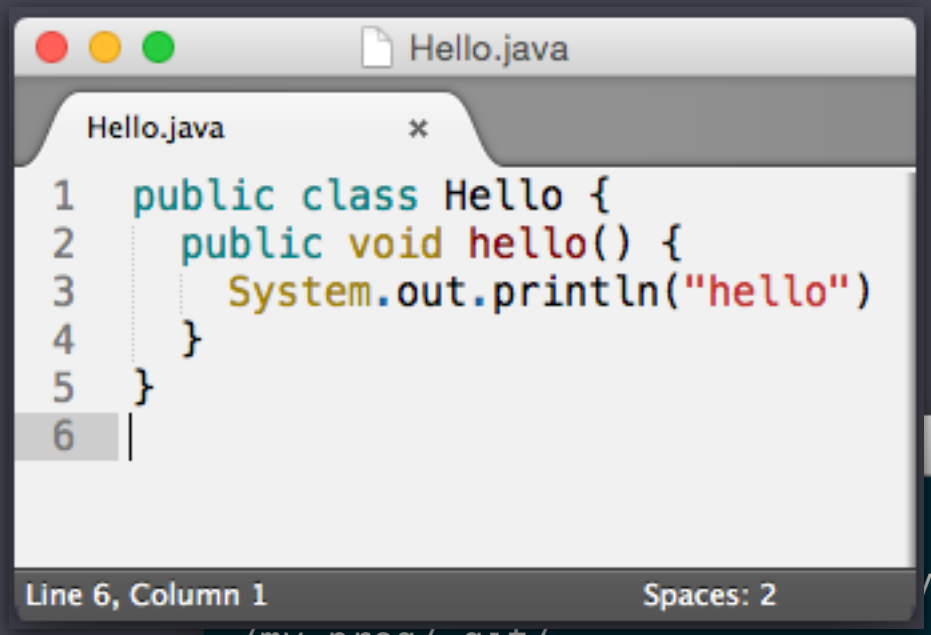


my-prog



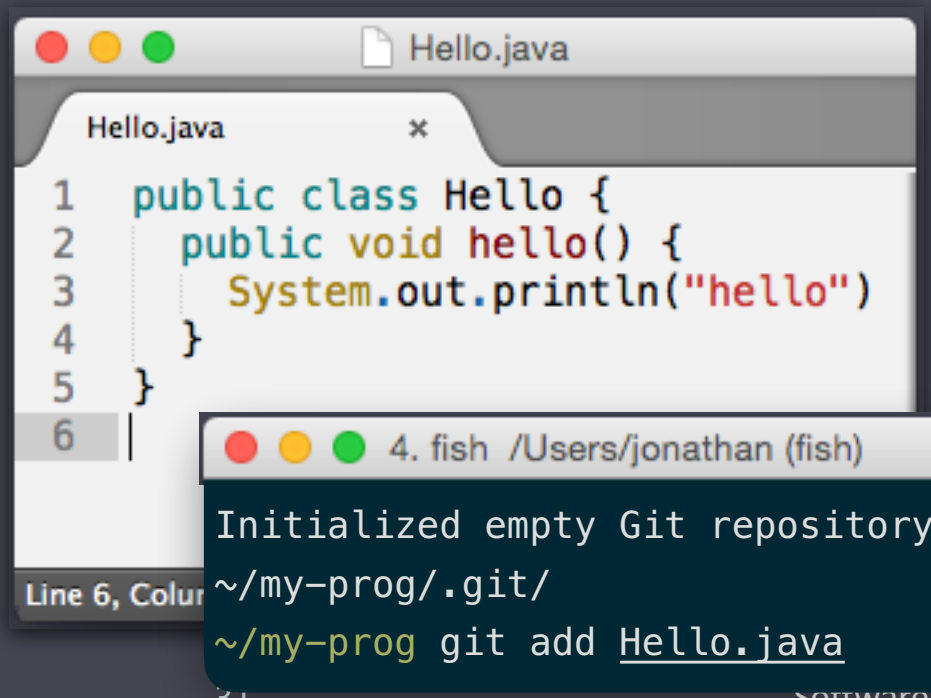

my-prog
.git


4. fish /Users/jonathan (fish)
~/my-prog git init
Initialized empty Git repository in
~/my-prog/.git/


Hello.java
Hello.java
1 public class Hello {
2 public void hello() {
3 System.out.println("hello")
4 }
5 }
6
Line 6, Column 1
Spaces: 2

in
~/my-prog/.git/

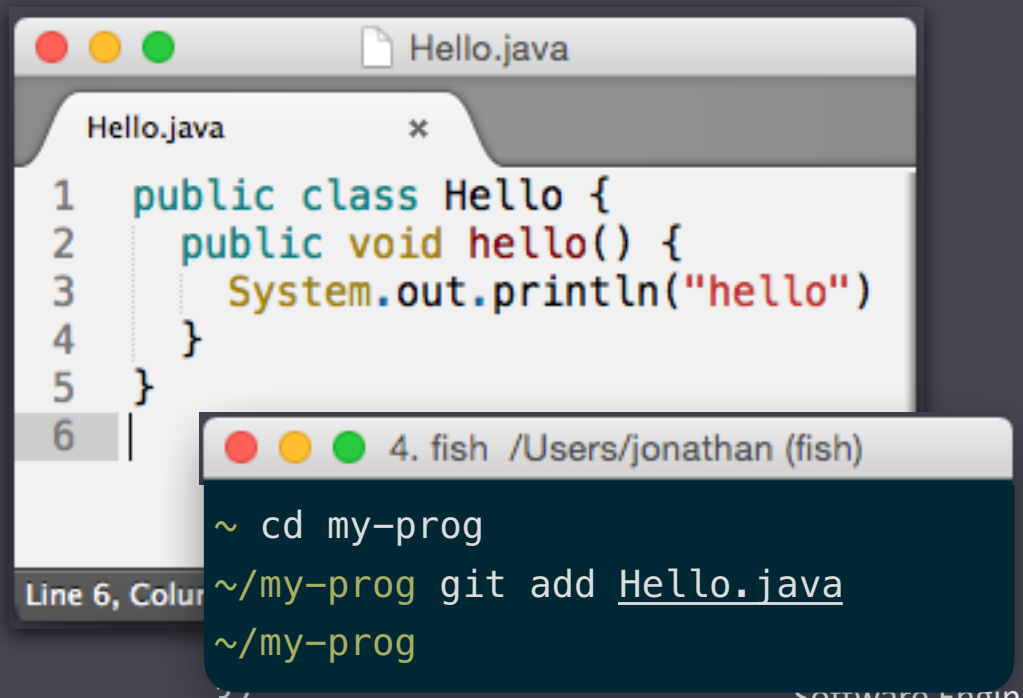

my-prog
.git
Hello.java


Hello.java
Hello.java
1 public class Hello {
2 public void hello() {
3 System.out.println("hello")
4 }
5 }
6
Line 6, Colu
4. fish /Users/jonathan (fish)
Initialized empty Git repository in
~/my-prog/.git/
~/my-prog git add Hello.java


my-prog
.git
Hello.java

index

Hello.java



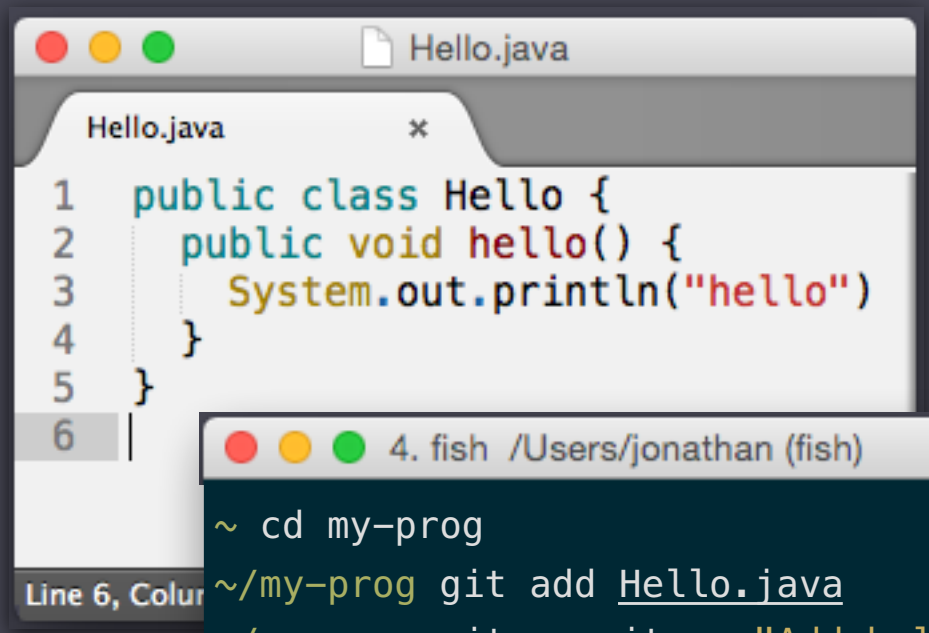
my-prog

.git

Hello.java

index

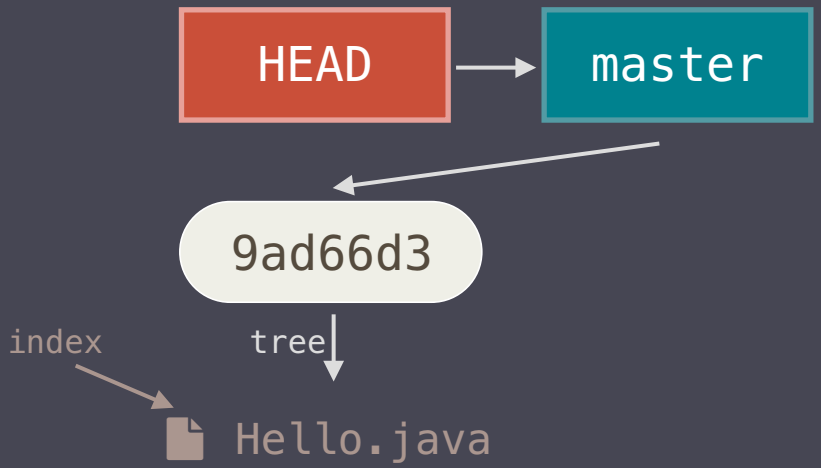
Hello.java



my-prog

.git

Hello.java



Hello.java

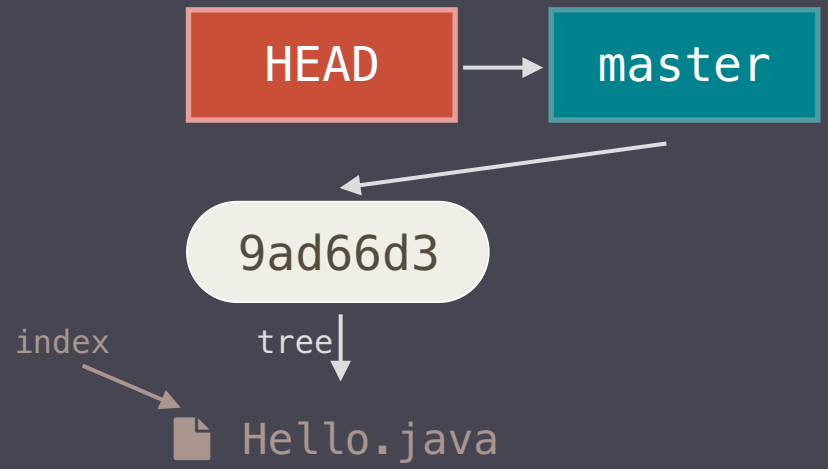
```
1  public class Hello {
2    public void hello() {
3      System.out.println("hello")
4    }
5  }
6
```

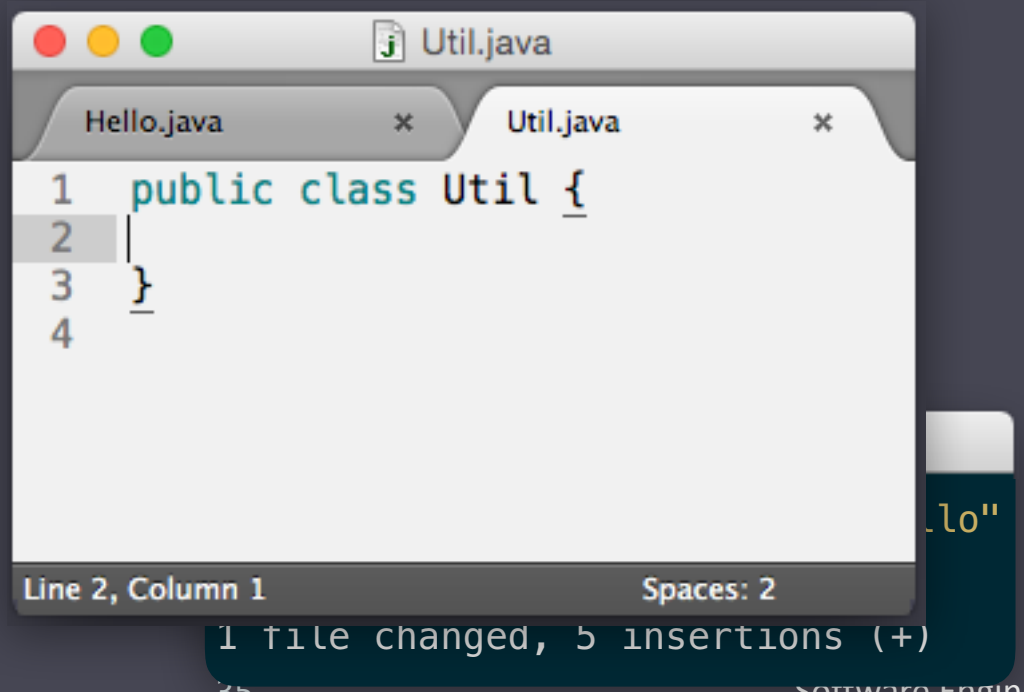
Line 6, Colu

4. fish /Users/jonathan (fish)

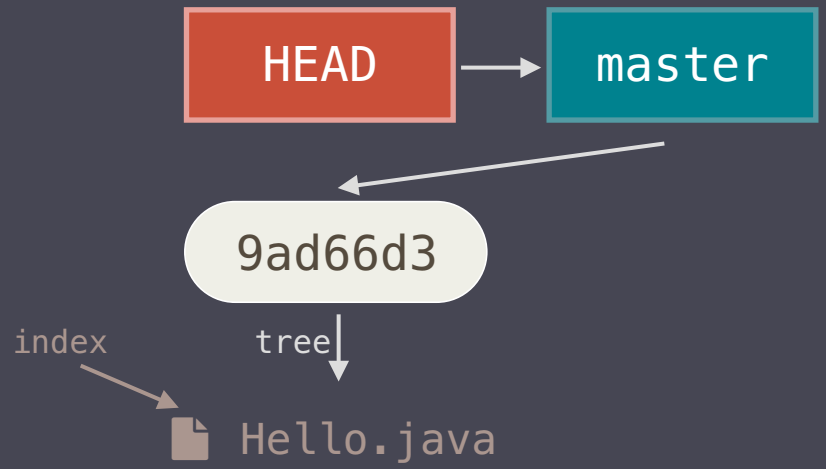
```
~/my-prog git commit -m "Add hello"
[master 9ad66d3] Add hello
1 file changed, 5 insertions (+)
```

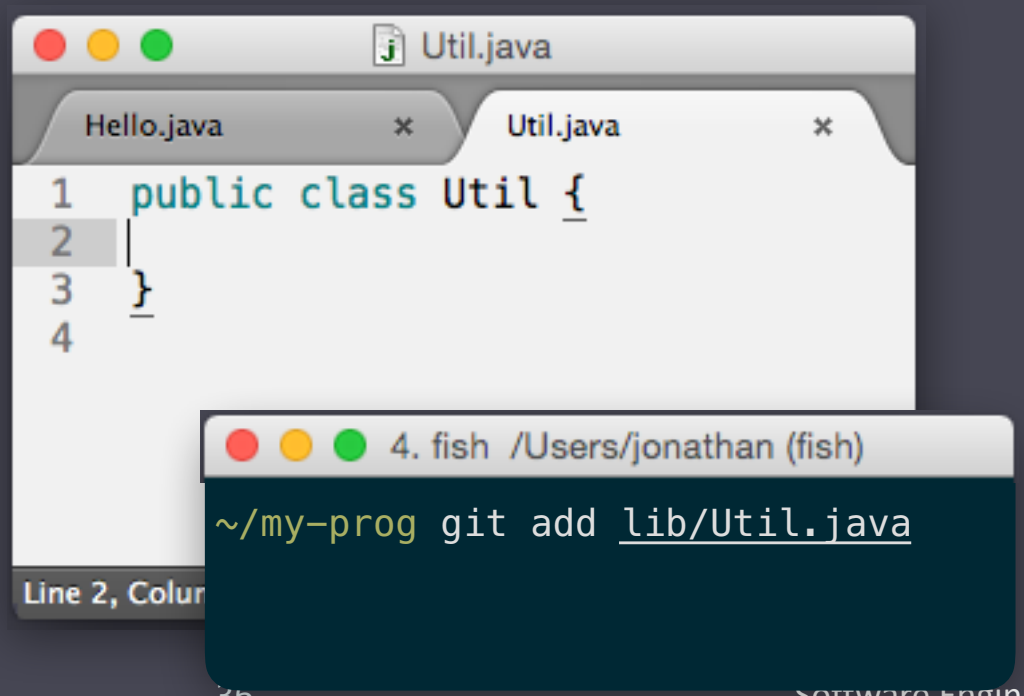
- my-prog
  - .git
  - Hello.java


HEAD
master
9ad66d3
index
tree
Hello.java

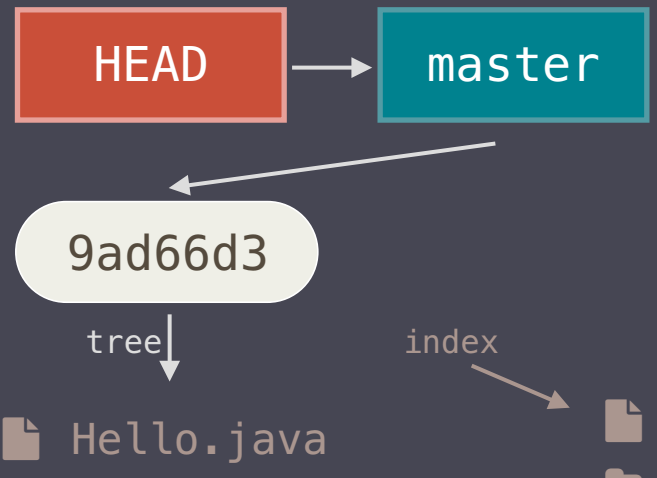

Util.java
Hello.java
Util.java
1 public class Util {
2
3 }
4
Line 2, Column 1
Spaces: 2
lo"
1 file changed, 5 insertions (+)

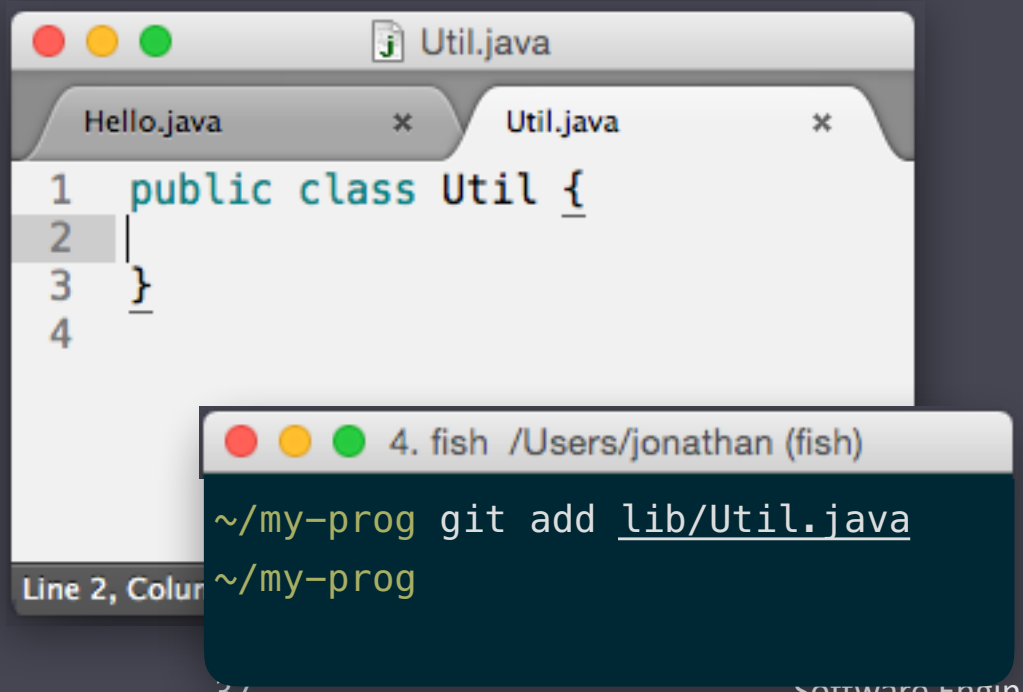

my-prog
.git
Hello.java
lib
Util.java

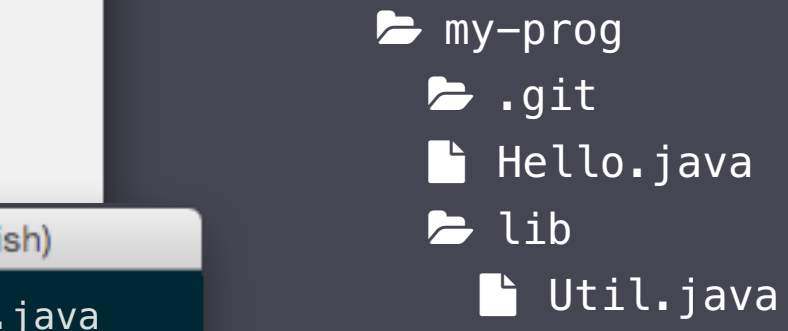

HEAD
master
9ad66d3
index
tree
Hello.java

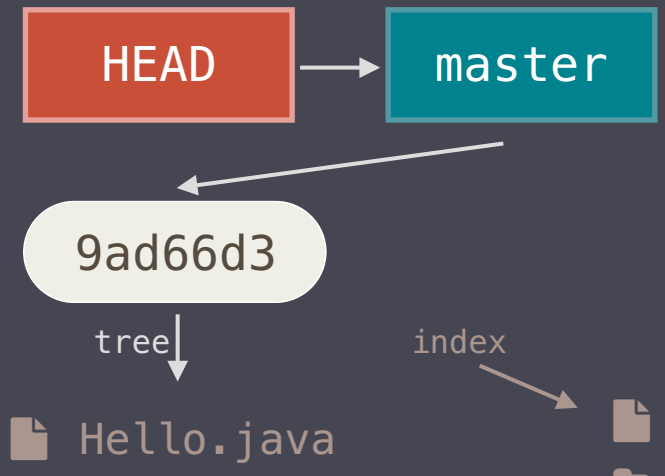

Util.java
Hello.java
Util.java
1 public class Util {
2
3 }
4
4. fish /Users/jonathan (fish)
~/my-prog git add lib/Util.java
Line 2, Colur

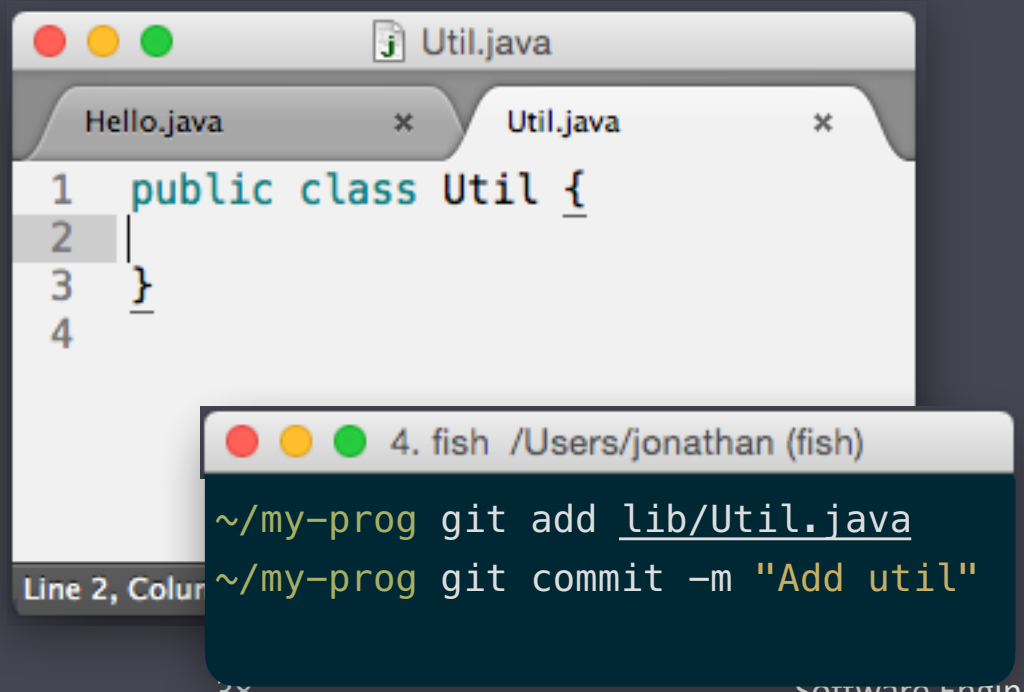

my-prog
.git
Hello.java
lib
Util.java

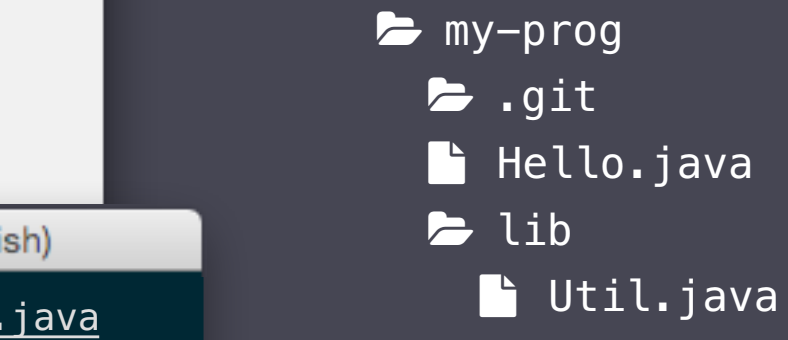

HEAD
master
9ad66d3
tree
index
Hello.java
Hello.java
lib
Util.java

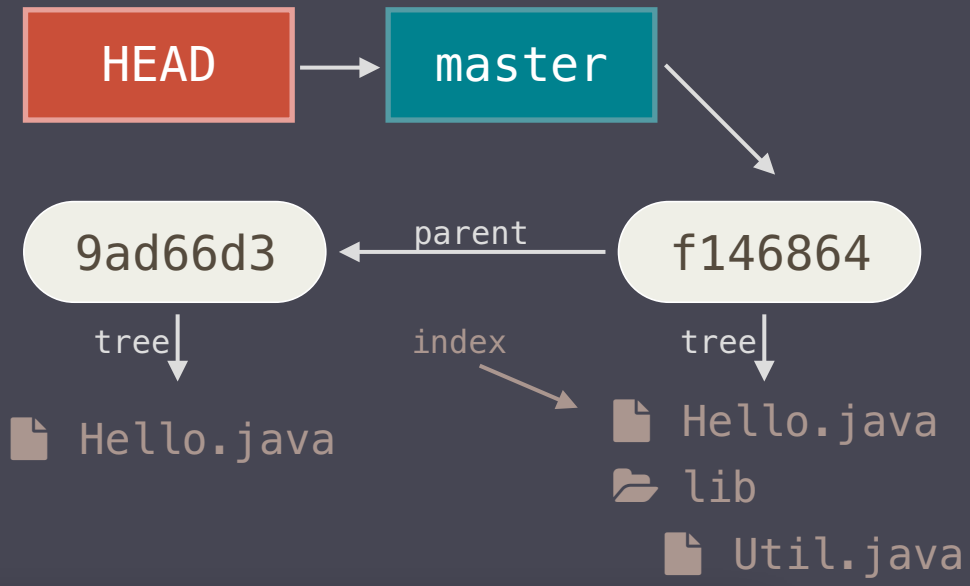

Util.java
Hello.java
Util.java
1 public class Util {
2
3 }
4
Line 2, Colur
4. fish /Users/jonathan (fish)
~/my-prog git add lib/Util.java
~/my-prog

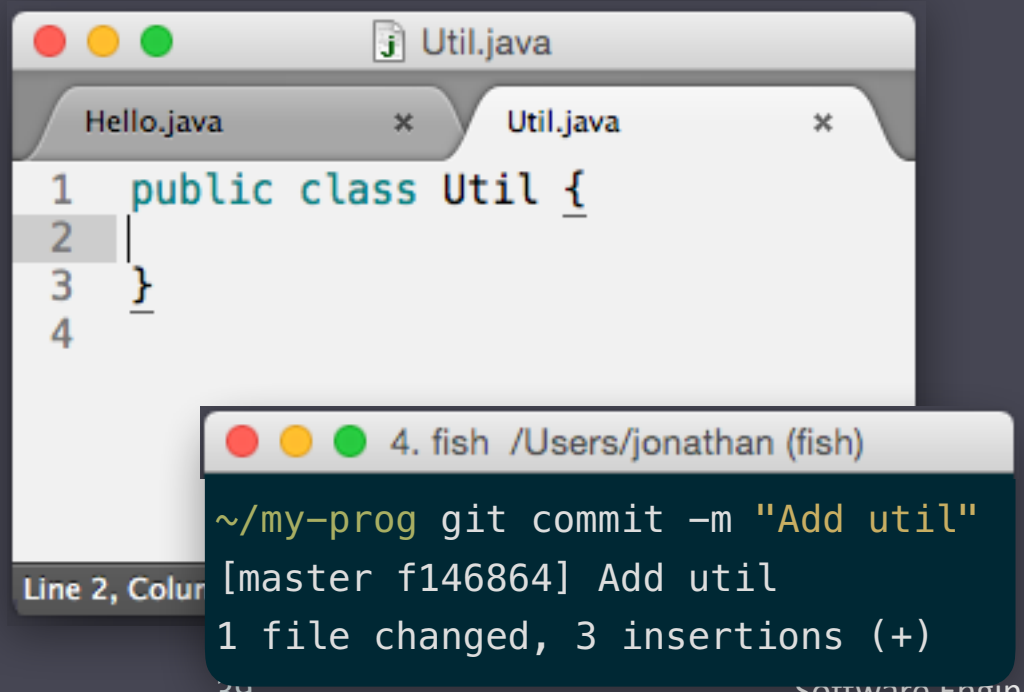

my-prog
.git
Hello.java
lib
Util.java

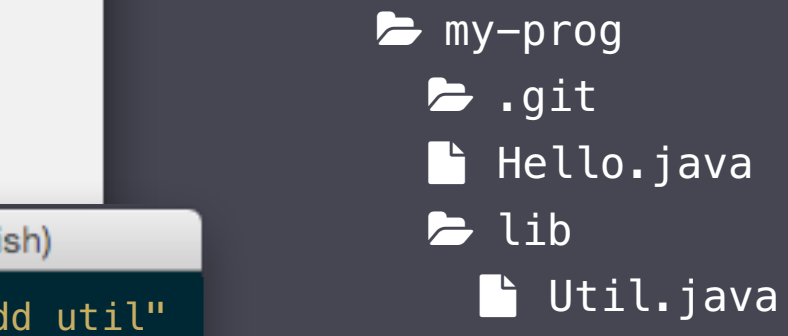

HEAD
master
9ad66d3
tree
index
Hello.java
Hello.java
lib
Util.java

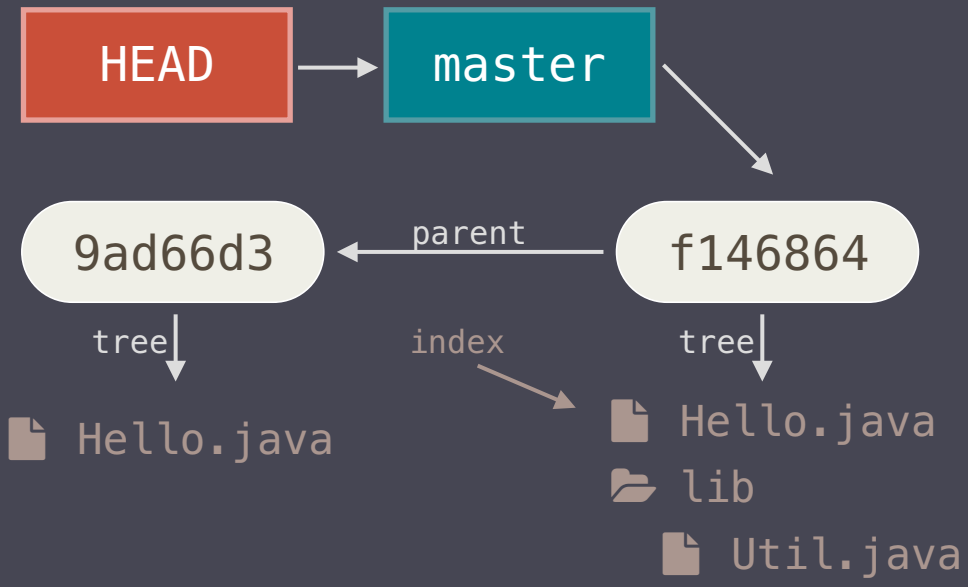

Util.java
Hello.java
Util.java
1 public class Util {
2
3 }
4
Line 2, Colur
4. fish /Users/jonathan (fish)
~/my-prog git add lib/Util.java
~/my-prog git commit -m "Add util"


my-prog
.git
Hello.java
lib
Util.java


HEAD
master
9ad66d3
parent
f146864
tree
index
tree
Hello.java
Hello.java
lib
Util.java


Util.java
Hello.java
Util.java
1 public class Util {
2
3 }
4
Line 2, Colur
4. fish /Users/jonathan (fish)
~/my-prog git commit -m "Add util"
[master f146864] Add util
1 file changed, 3 insertions (+)

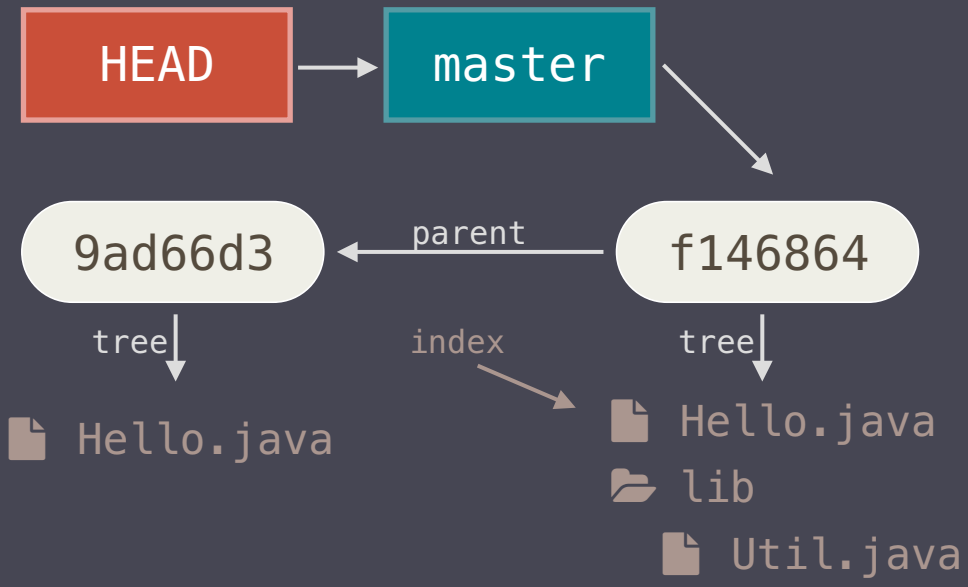

my-prog
.git
Hello.java
lib
Util.java


HEAD
master
9ad66d3
parent
f146864
tree
index
tree
Hello.java
Hello.java
lib
Util.java


Hello.java
Hello.java
Util.java
1 public class Hello {
2 public void hello() {
3 System.out.println("hello")
4 }
5 public void bye() {}
6 }
7
Line 5, Column 9
Spaces: 2

il"
1 file changed, 3 insertions (+)


my-prog
.git
Hello.java
lib
Util.java



Hello.java

Hello.java × Util.java ×

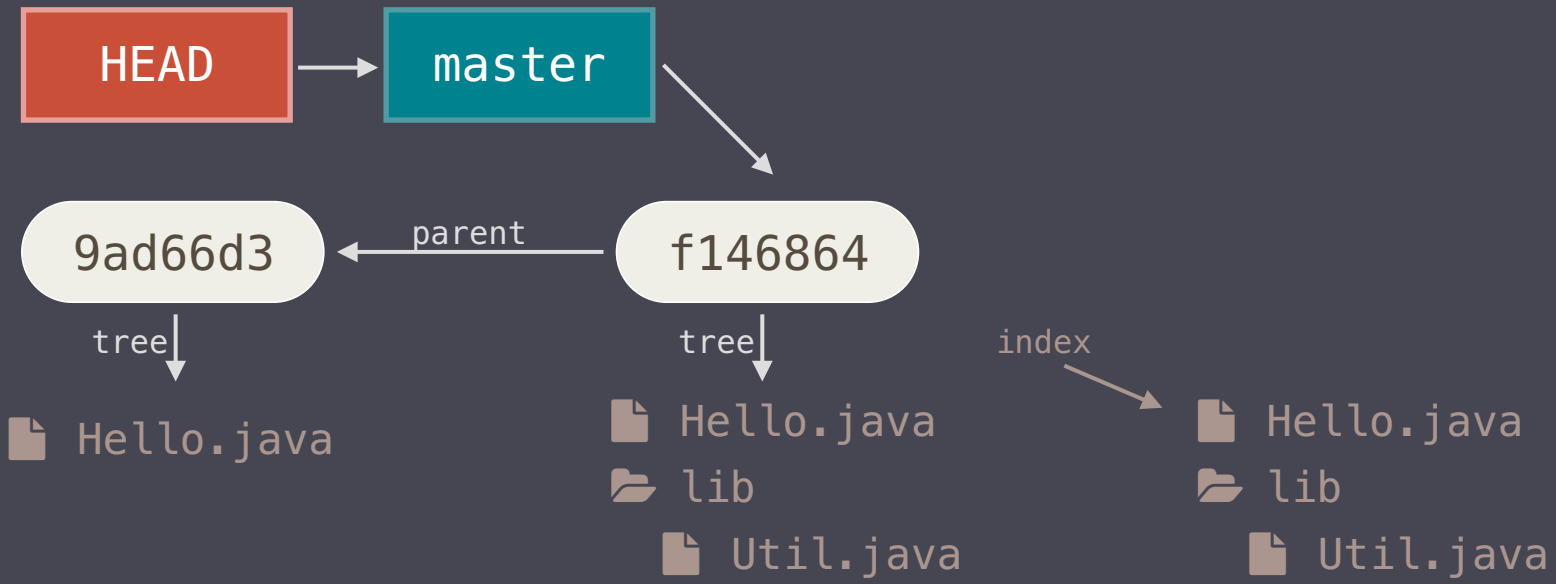
```
1  public class Hello {
2    public void hello() {
3      System.out.println("hello")
4    }
5    public void bye() {}
6  }
7
```

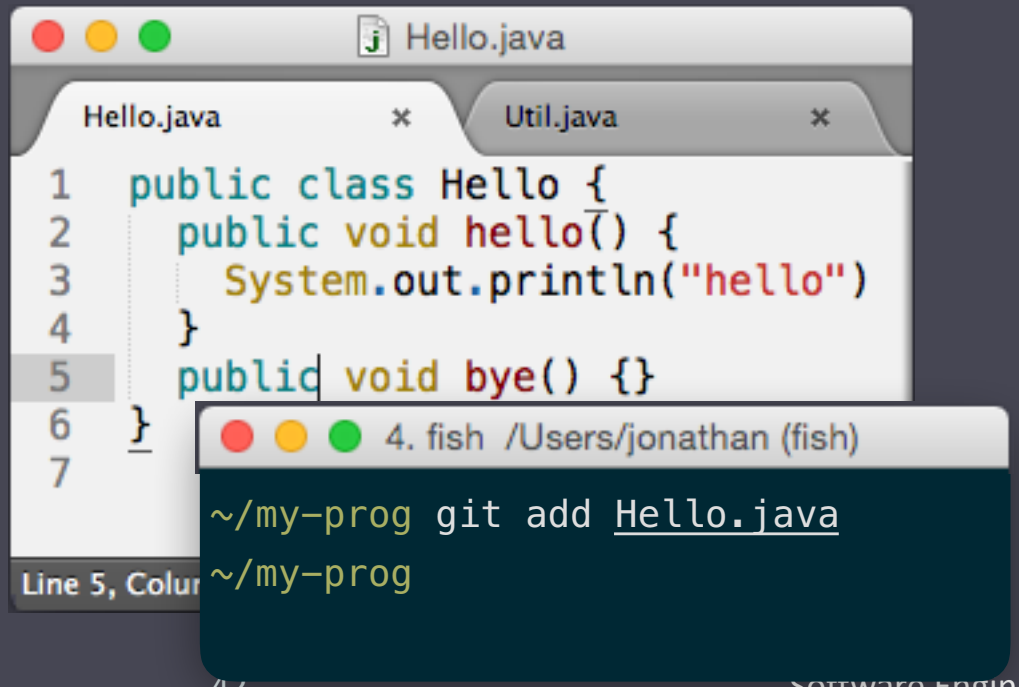
Line 5, Colu

4. fish /Users/jonathan (fish)

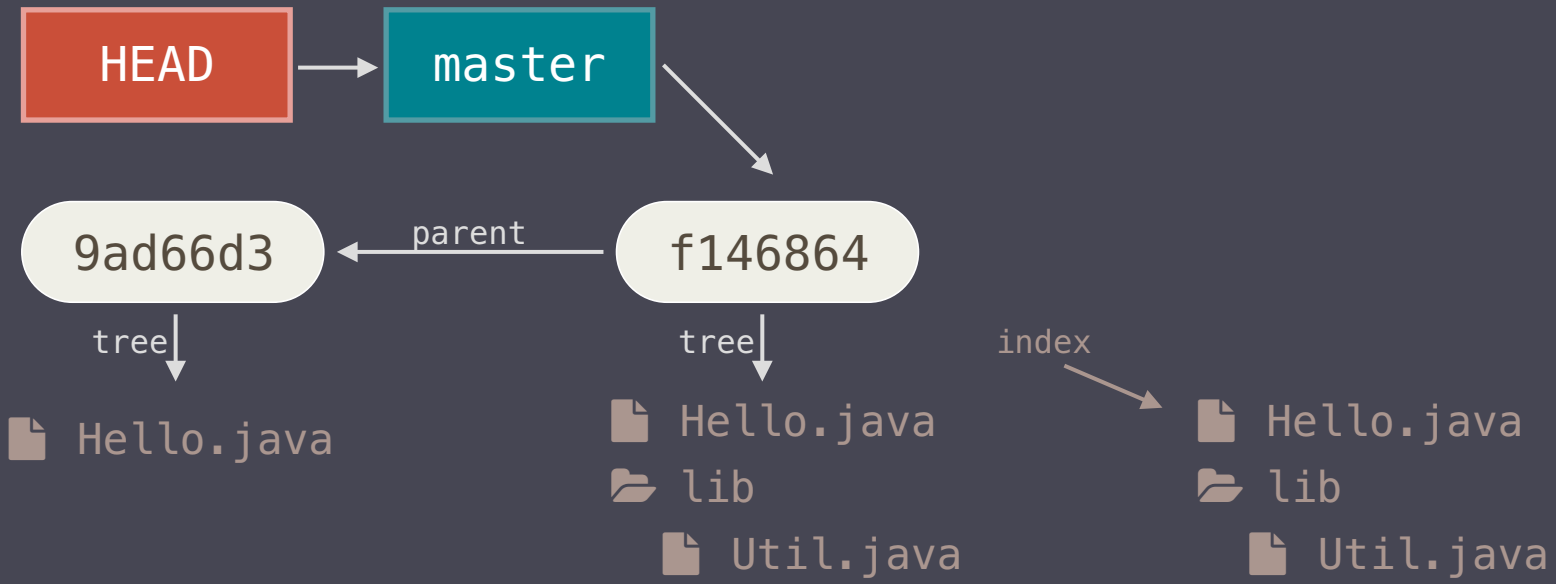
```
~/my-prog git add Hello.java
```

- my-prog
  - .git
  - Hello.java
  - lib
    - Util.java


HEAD
master
9ad66d3
parent
f146864
tree
tree
index
Hello.java
Hello.java
lib
Util.java
Hello.java
lib
Util.java


Hello.java
Hello.java
Util.java
1 public class Hello {
2 public void hello() {
3 System.out.println("hello")
4 }
5 public void bye() {}
6 }
7
Line 5, Colu
4. fish /Users/jonathan (fish)
~/my-prog git add Hello.java
~/my-prog


my-prog
.git
Hello.java
lib
Util.java



Hello.java

Hello.java | Util.java

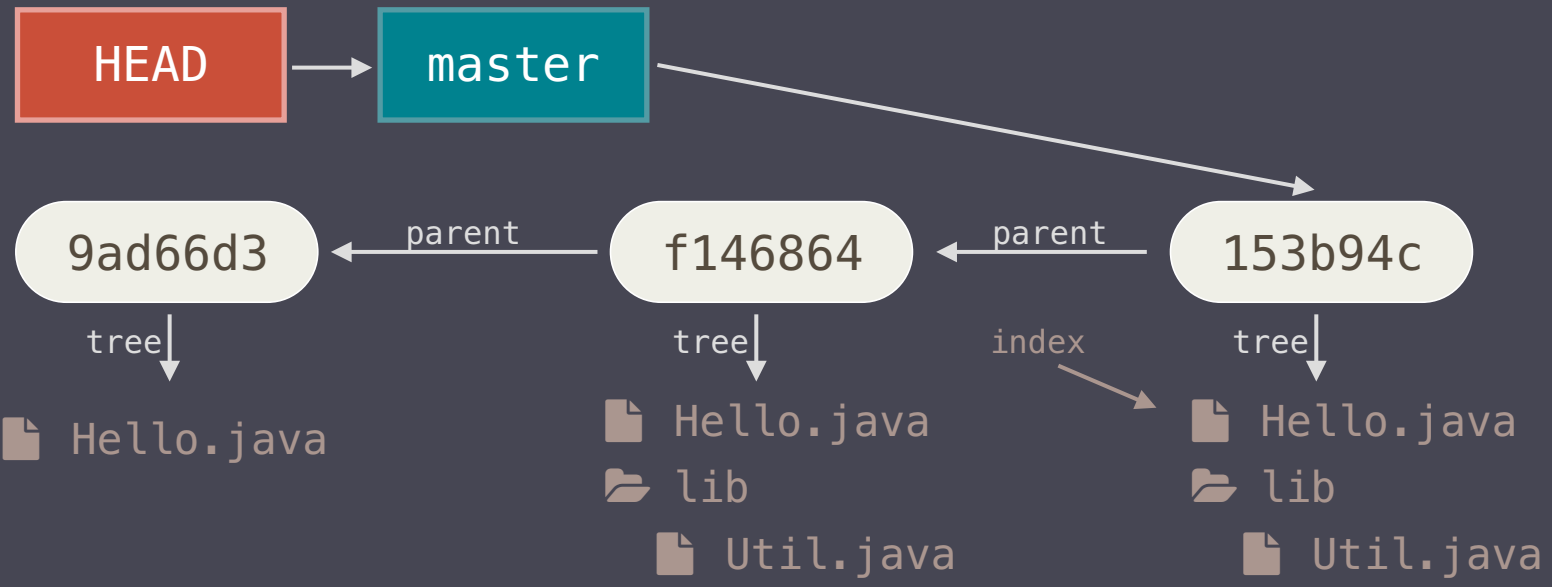
```
1  public class Hello {
2    public void hello() {
3      System.out.println("hello")
4    }
5    public void bye() {}
6  }
7
```

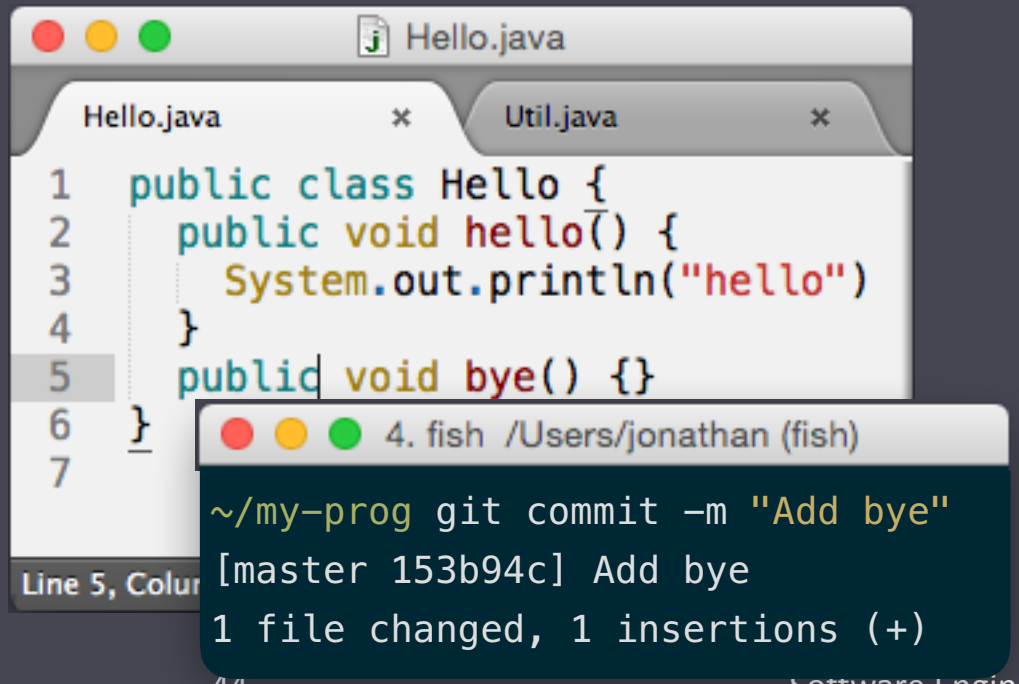
Line 5, Colu

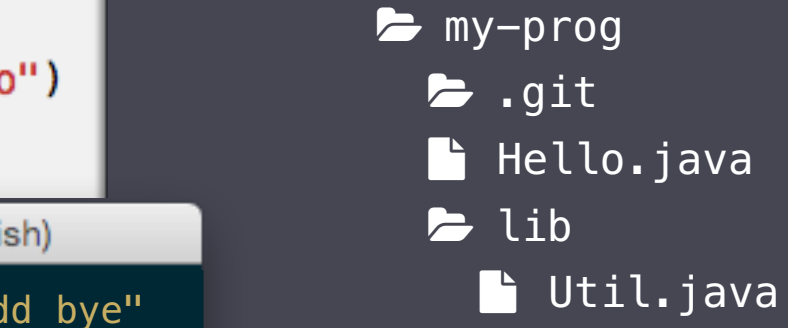
4. fish /Users/jonathan (fish)

```
~/my-prog git add Hello.java
~/my-prog git commit -m "Add bye"
```

- my-prog
  - .git
  - Hello.java
  - lib
    - Util.java


HEAD
master
9ad66d3
parent
f146864
parent
153b94c
tree
Hello.java
tree
Hello.java
lib
Util.java
index
tree
Hello.java
lib
Util.java


Hello.java
Hello.java
Util.java
1 public class Hello {
2 public void hello() {
3 System.out.println("hello")
4 }
5 public void bye() {}
6 }
7
Line 5, Colu
4. fish /Users/jonathan (fish)
~/my-prog git commit -m "Add bye"
[master 153b94c] Add bye
1 file changed, 1 insertions (+)


my-prog
.git
Hello.java
lib
Util.java

Demo

(init, status, diff, add (-p), diff -- staged, checkout, commit, log)

Die wichtigsten Befehle (1 / 4)

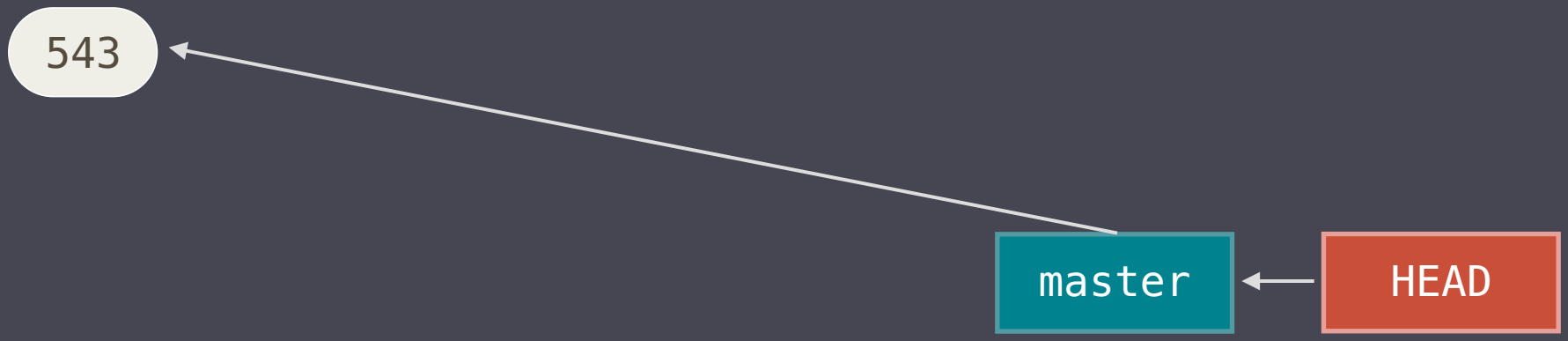
- **`git help`**
  Liste wichtiger Befehle
- **`git help` *`command`***
  Hilfe zu einzelnen Befehlen
- **`git init`**
  Neues Repository initialisieren
- **`git add`**
  Arbeitskopie → Index
- **`git checkout`**
  Index → Arbeitskopie
- **`git commit`**
  Index → Repository

Die wichtigsten Befehle (2 / 4)

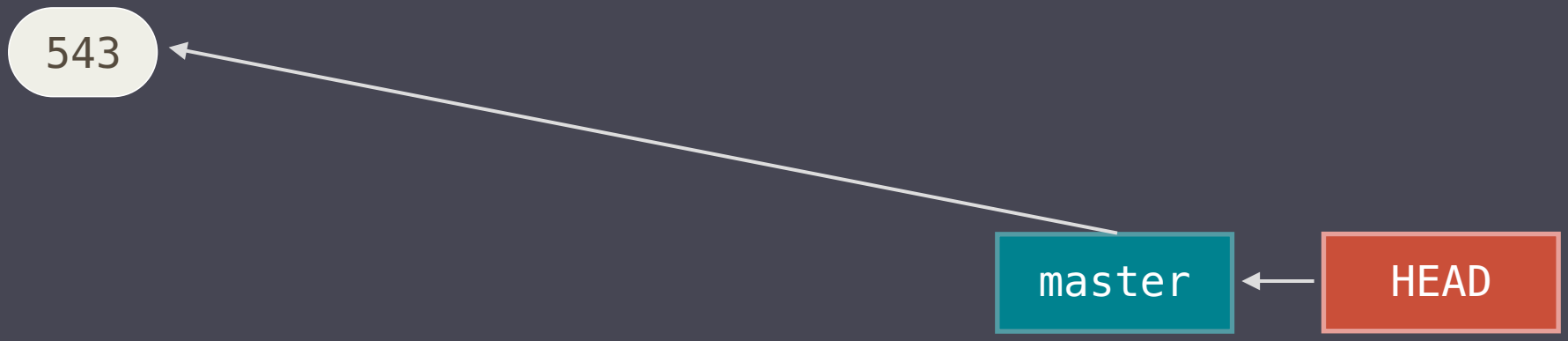
- **`git status`**
  Informationen zur Arbeitskopie und HEAD anzeigen
- **`git show commit`**
  Informationen zu Commit zeigen
- **`git reflog`**
  Entwicklung von HEAD zeigen
- **`git diff`**
  Arbeitskopie vs. Index
- **`git diff --staged`**
  Index vs. Head
- **`git log`**
  Änderungsgeschichte zeigen

Branches & Merging

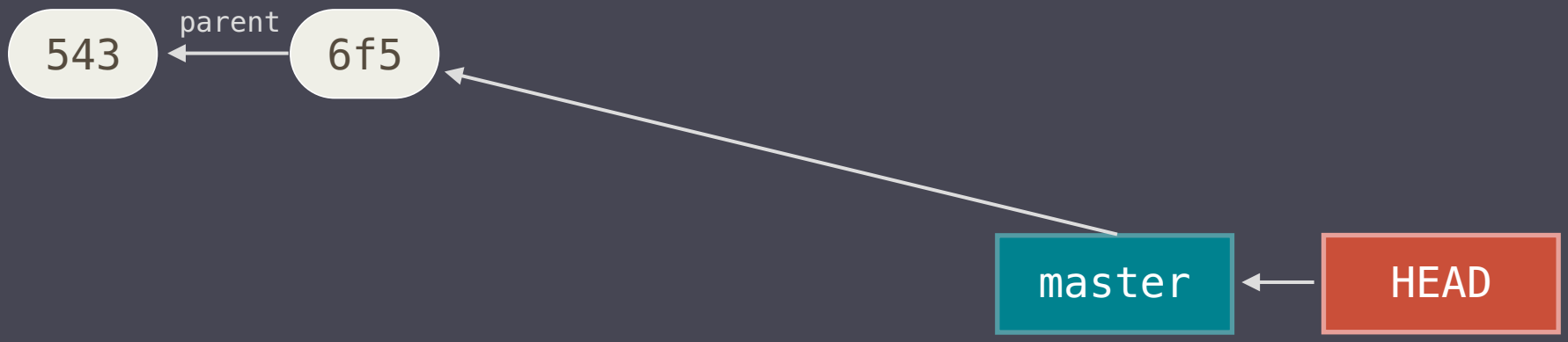
~/my-prog



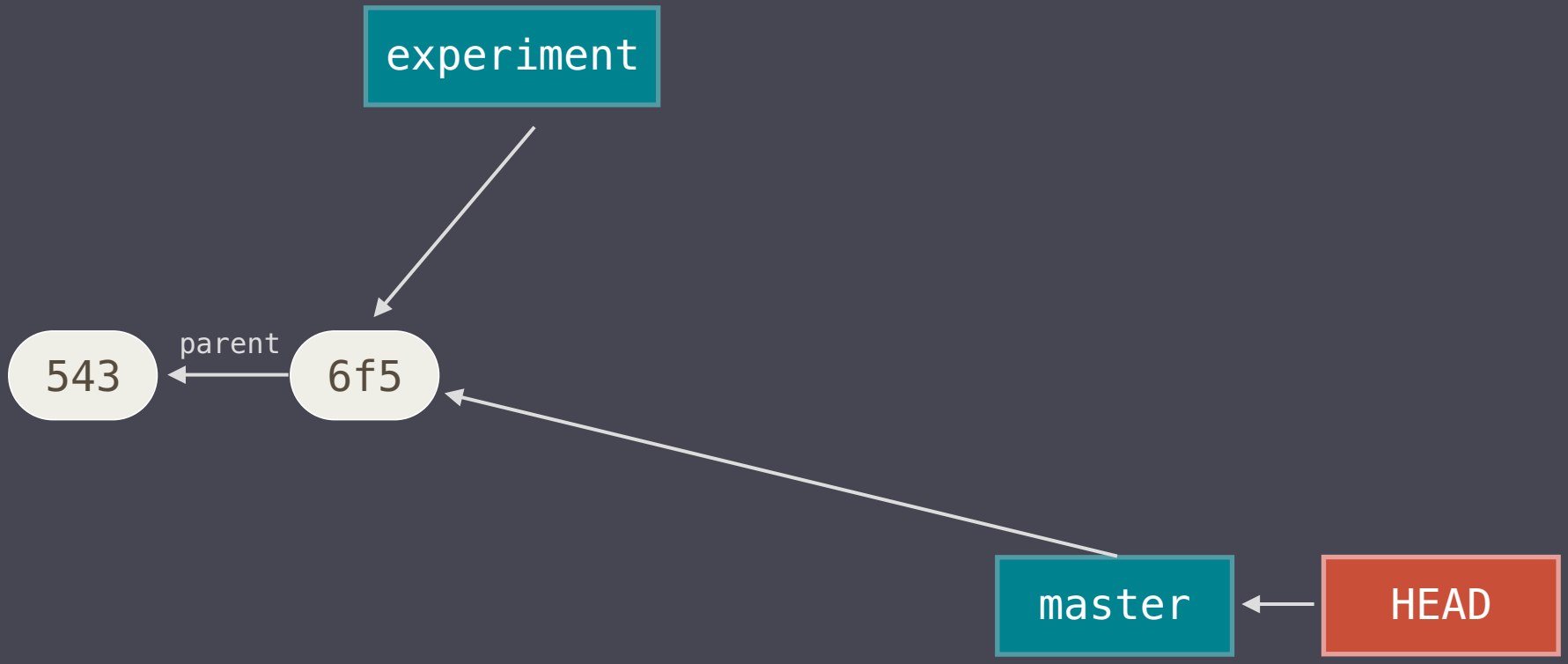
```
~/my-prog git add
~/my-prog
```



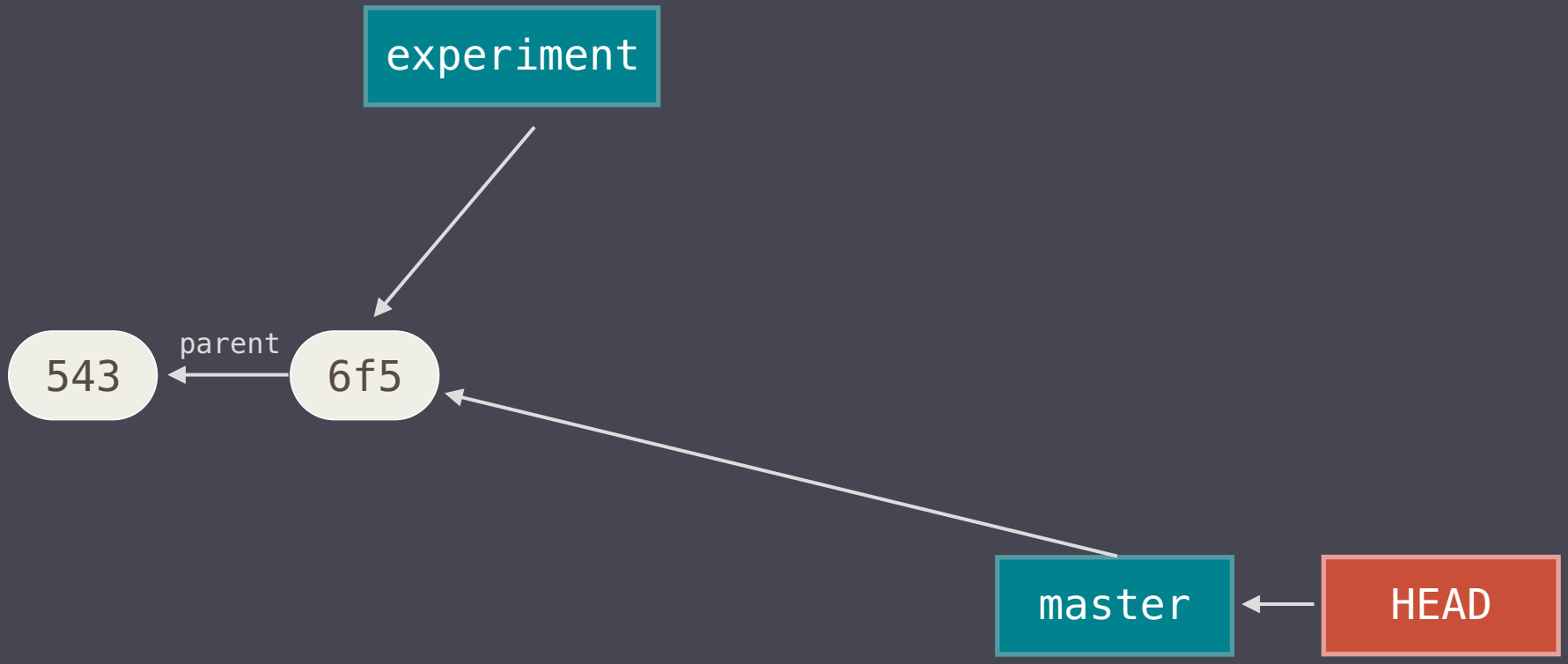
```
~/my-prog git add
~/my-prog git commit
~/my-prog
```



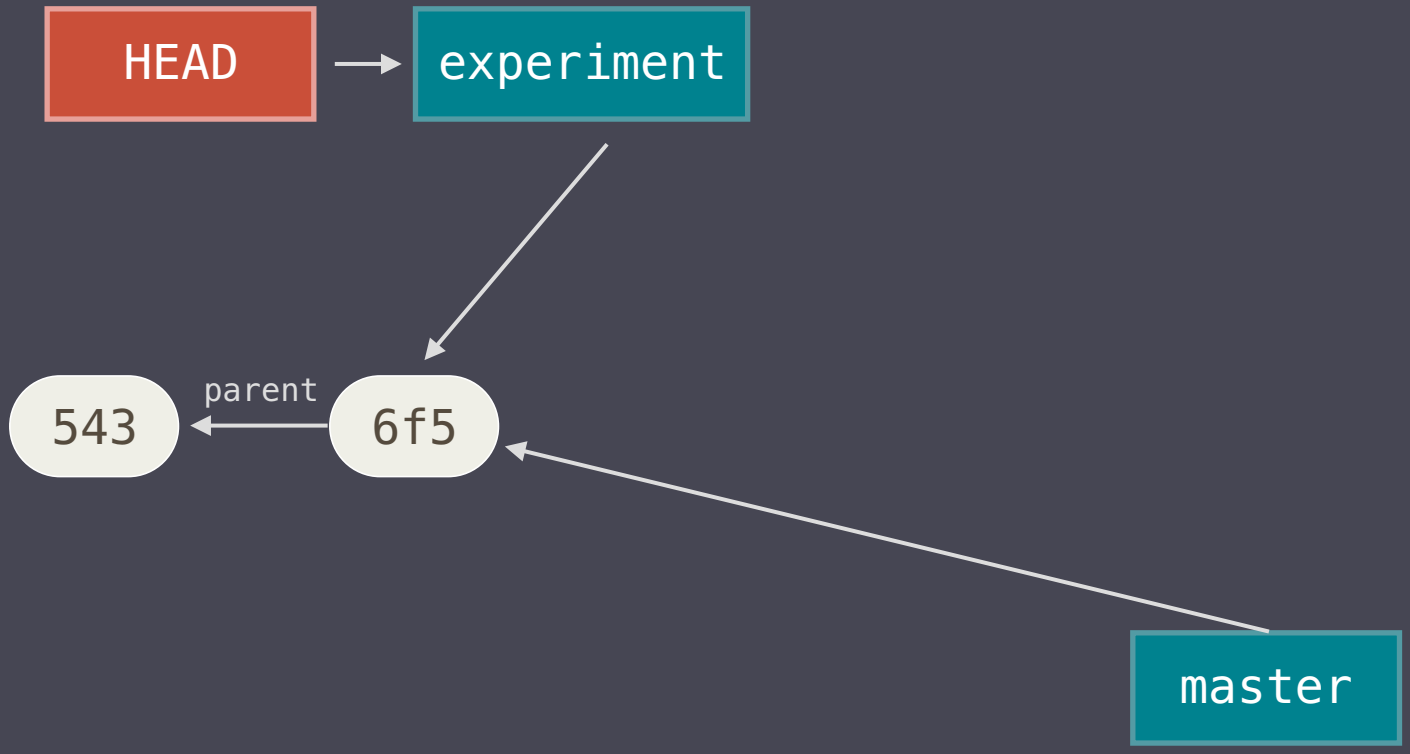
```
~/my-prog git add
~/my-prog git commit
~/my-prog git branch experiment
~/my-prog
```



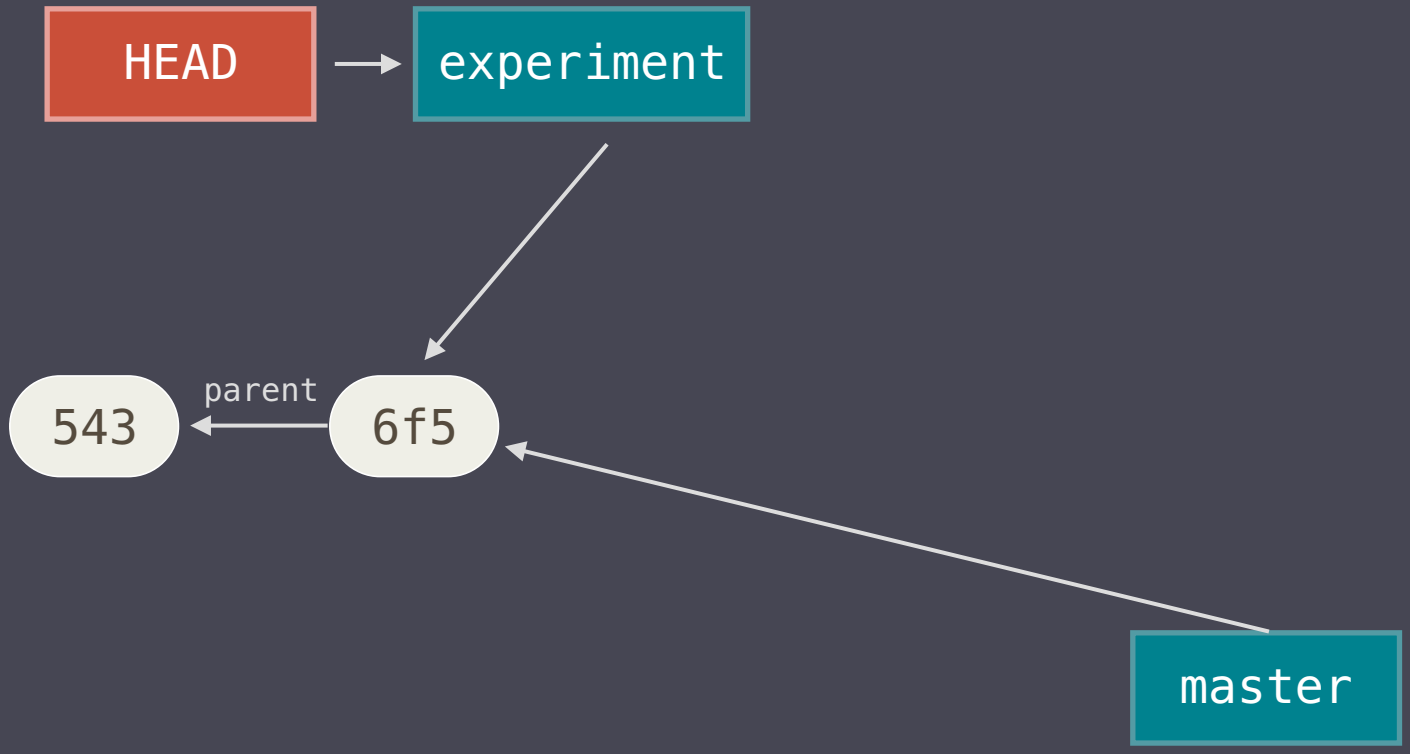
```
~/my-prog git add
~/my-prog git commit
~/my-prog git branch experiment
~/my-prog git checkout experiment
```



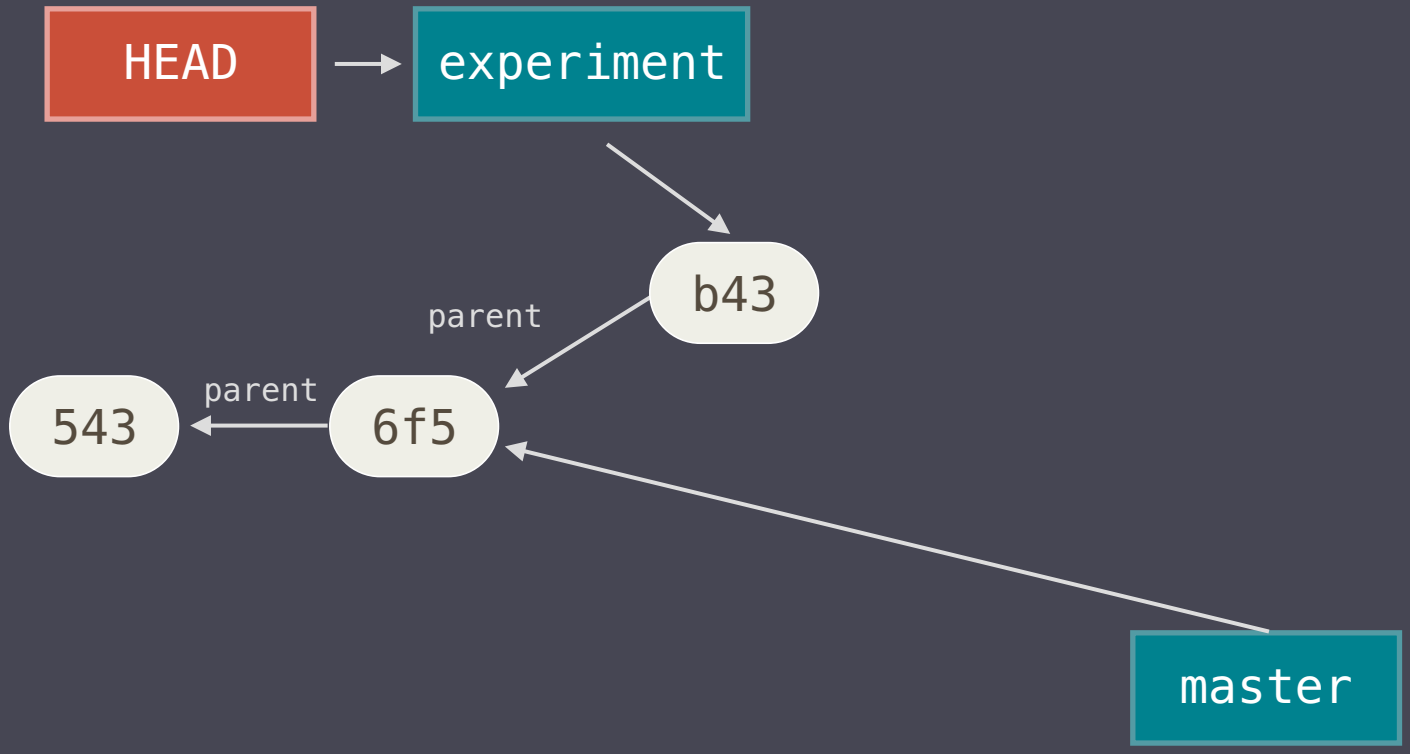
```
~/my-prog git checkout experiment
~/my-prog
```



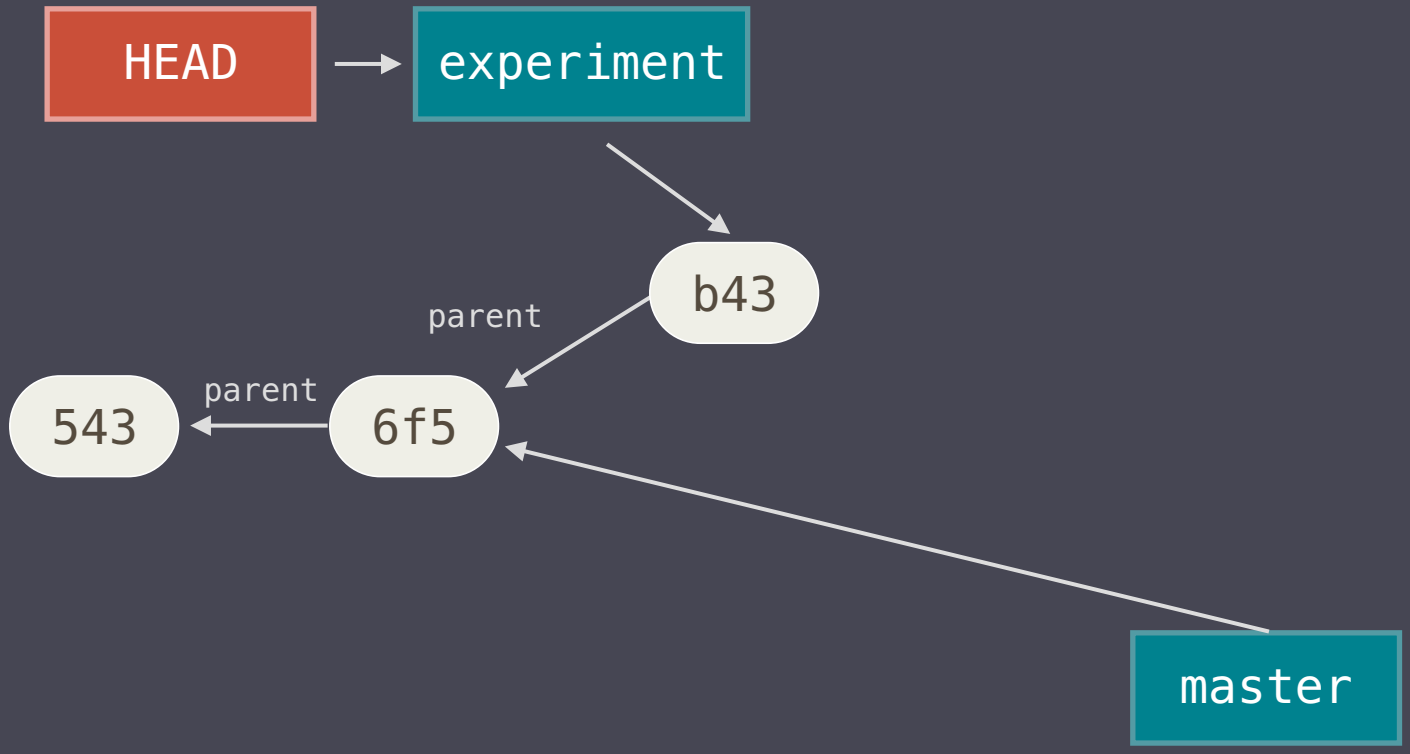
```
~/my-prog git checkout experiment
~/my-prog git add
~/my-prog
```



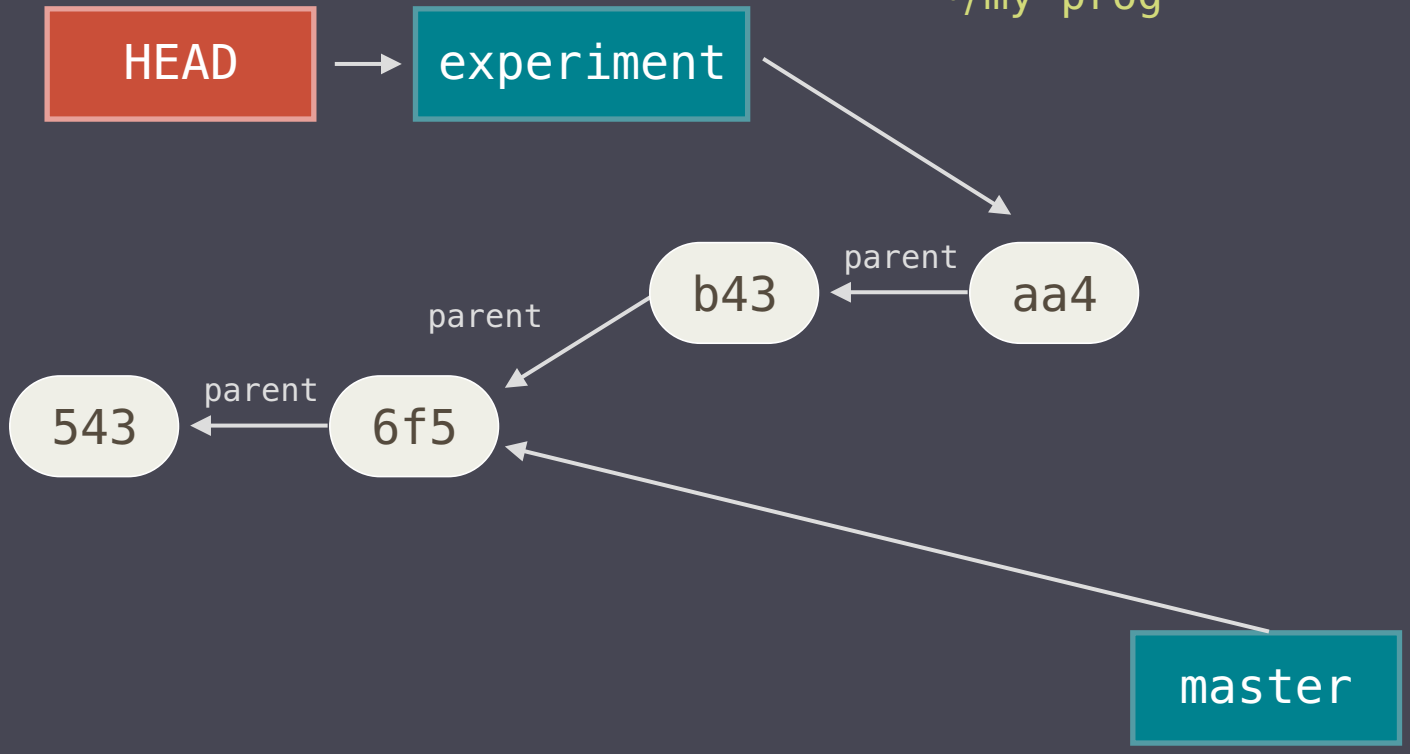
```
~/my-prog git checkout experiment
~/my-prog git add
~/my-prog git commit
~/my-prog
```



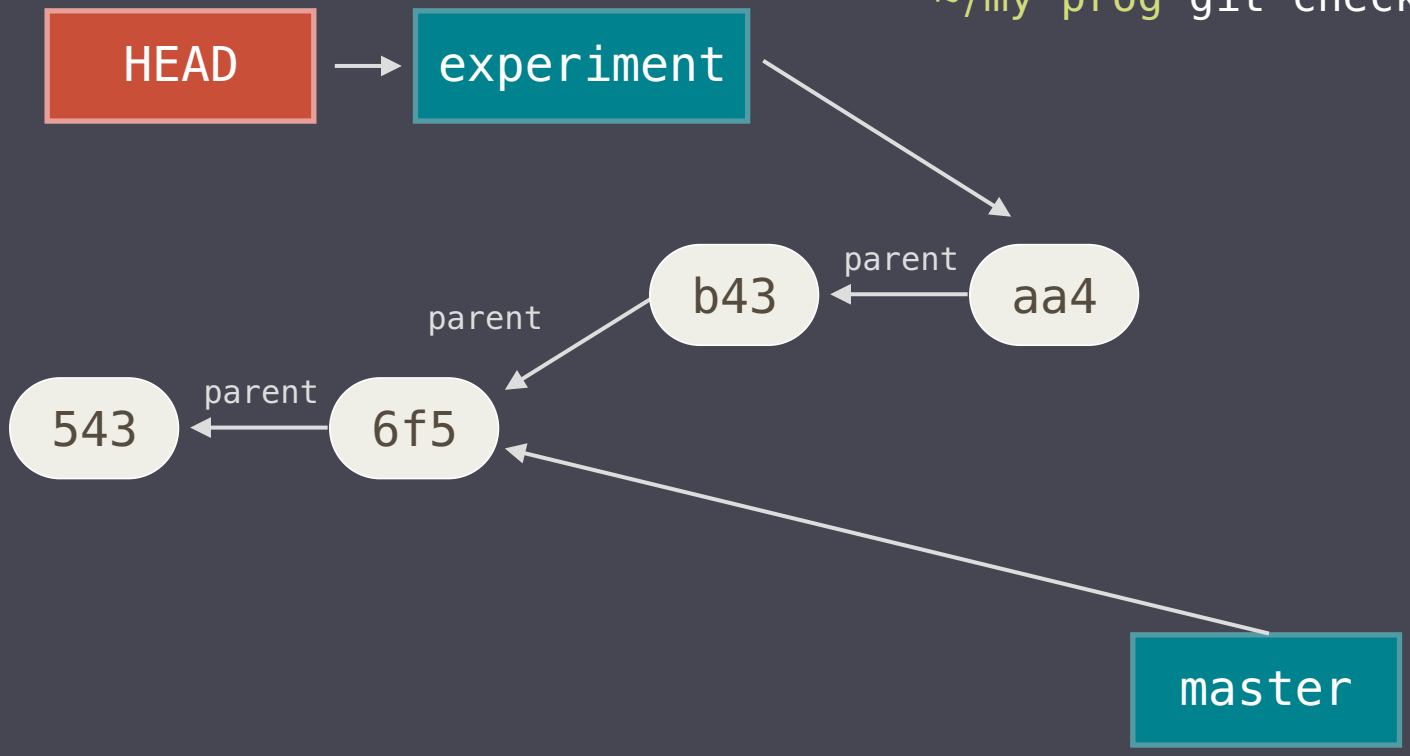
```
~/my-prog git checkout experiment
~/my-prog git add
~/my-prog git commit
~/my-prog git add
~/my-prog
```



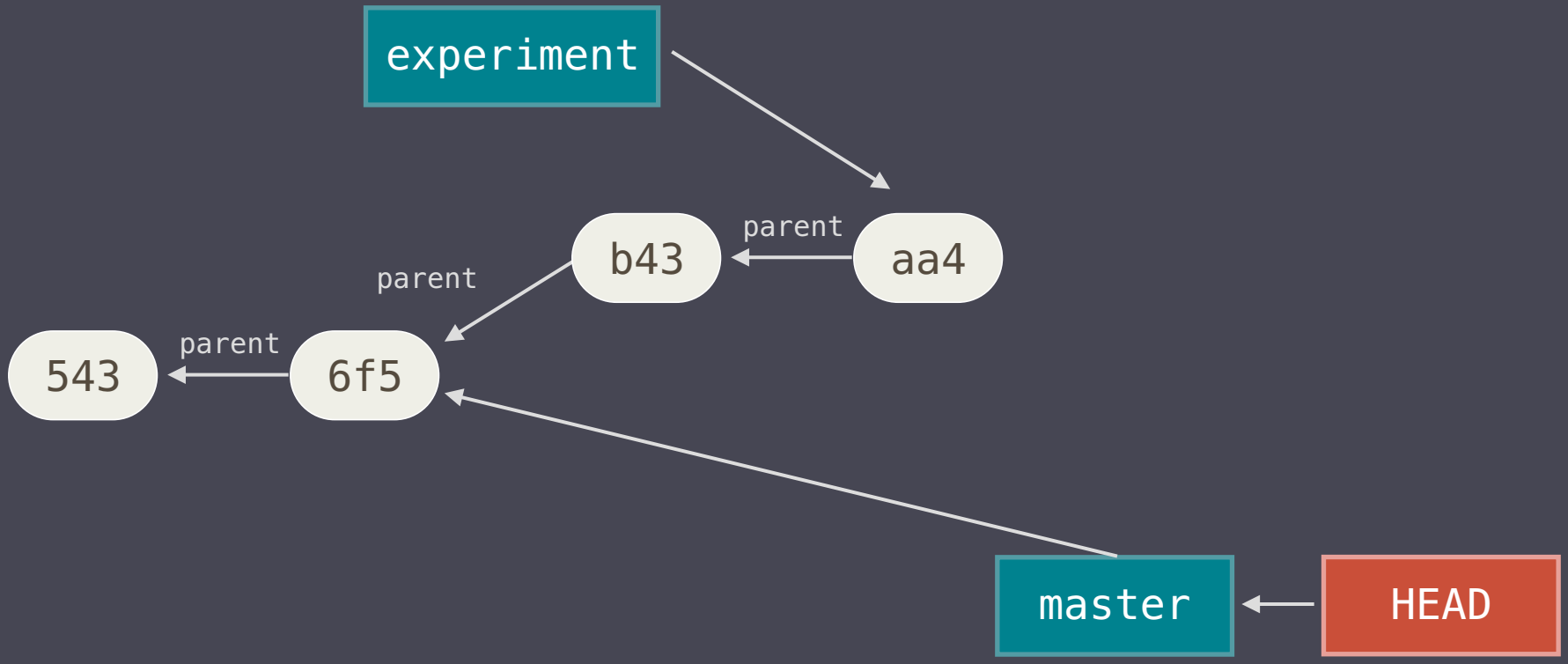
```
~/my-prog git checkout experiment
~/my-prog git add
~/my-prog git commit
~/my-prog git add
~/my-prog git commit
~/my-prog
```



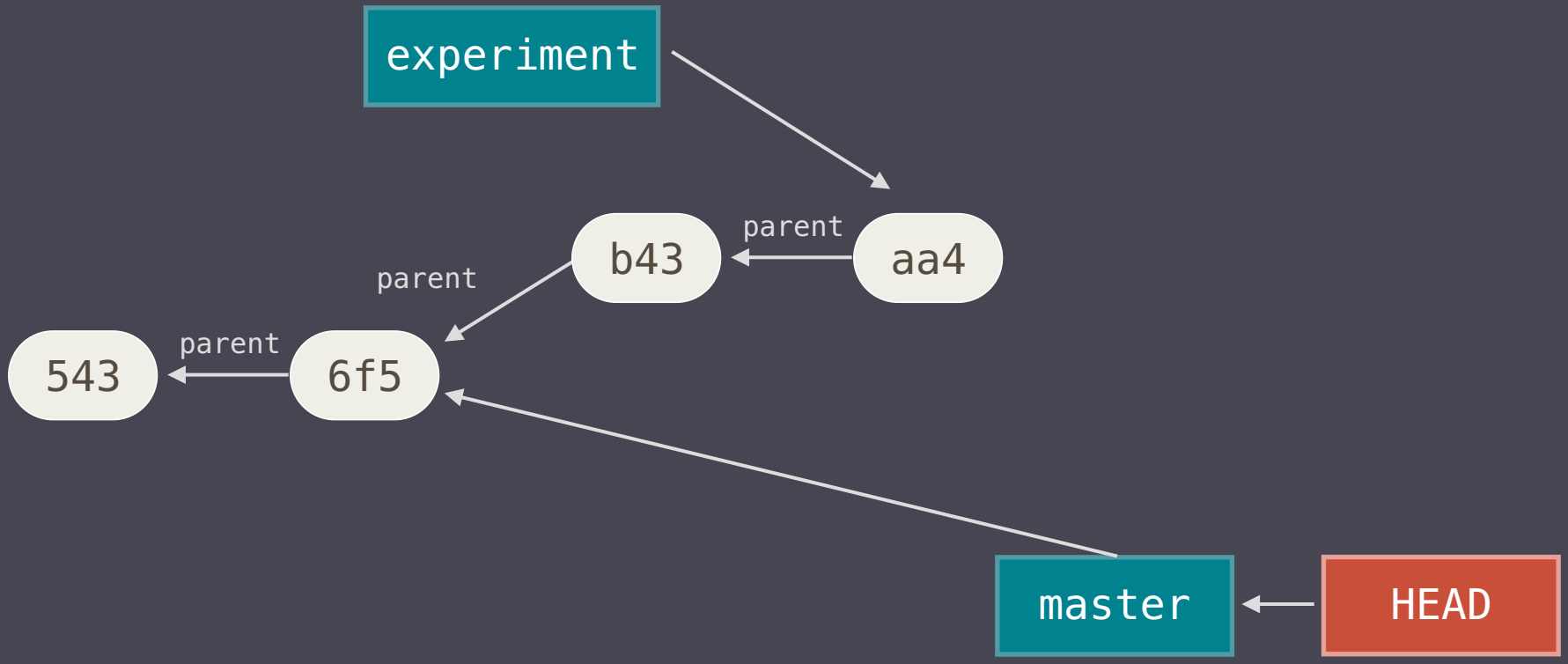
```
~/my-prog git checkout experiment
~/my-prog git add
~/my-prog git commit
~/my-prog git add
~/my-prog git commit
~/my-prog git checkout master
```



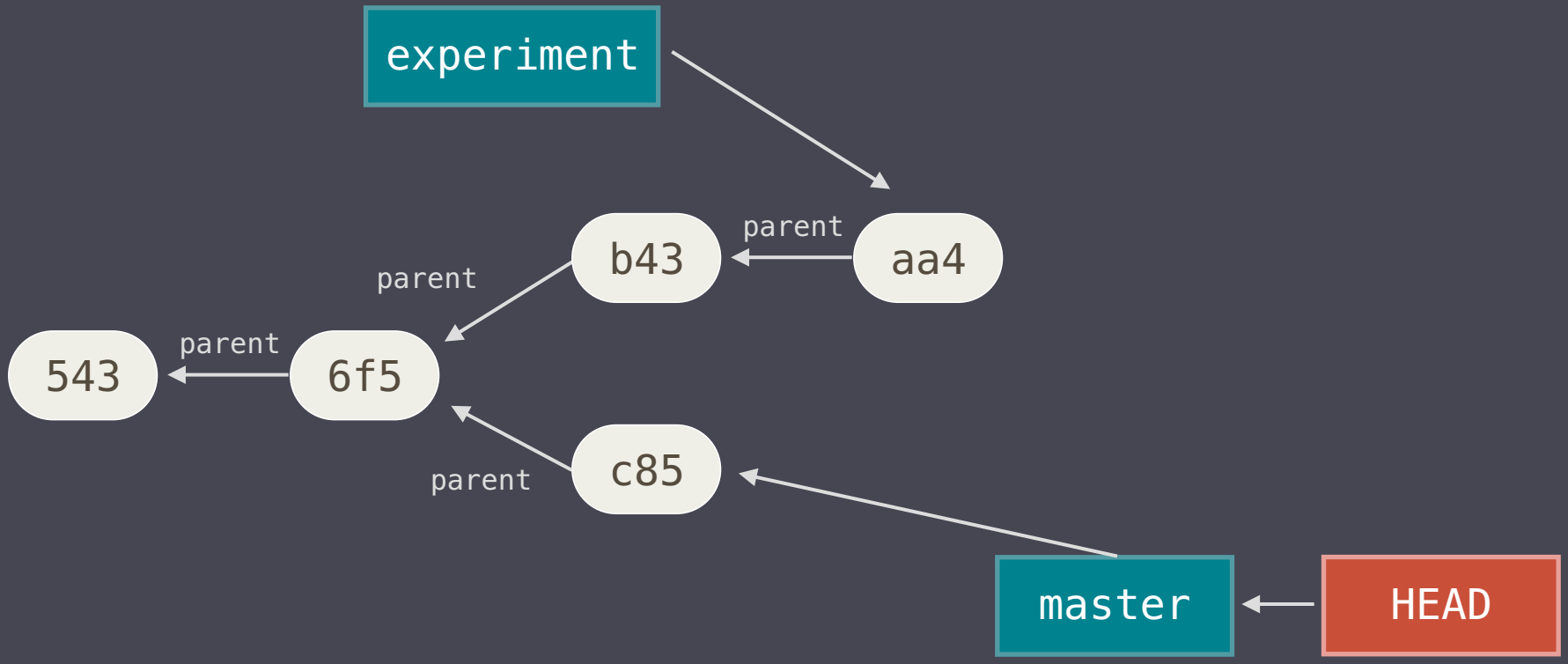
```
~/my-prog git checkout master
~/my-prog
```



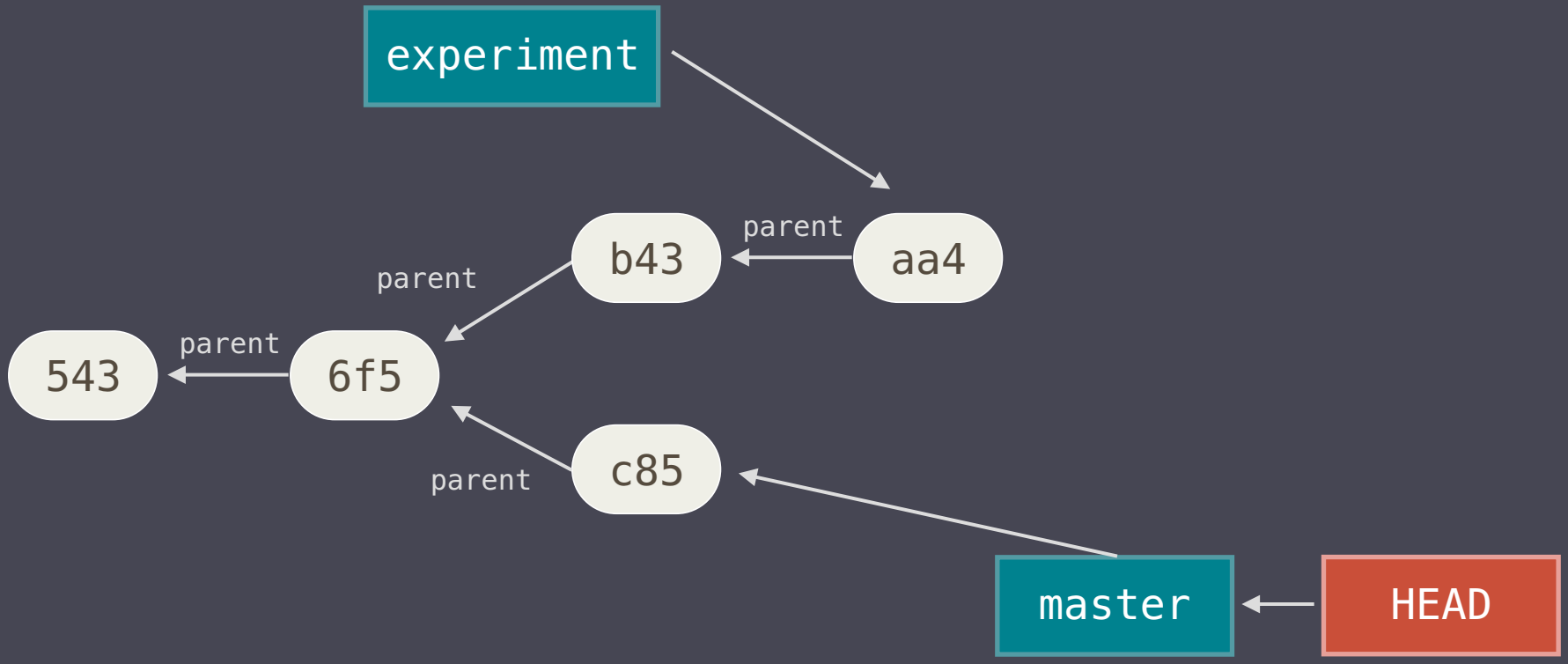
```
~/my-prog git checkout master
~/my-prog git add
~/my-prog
```



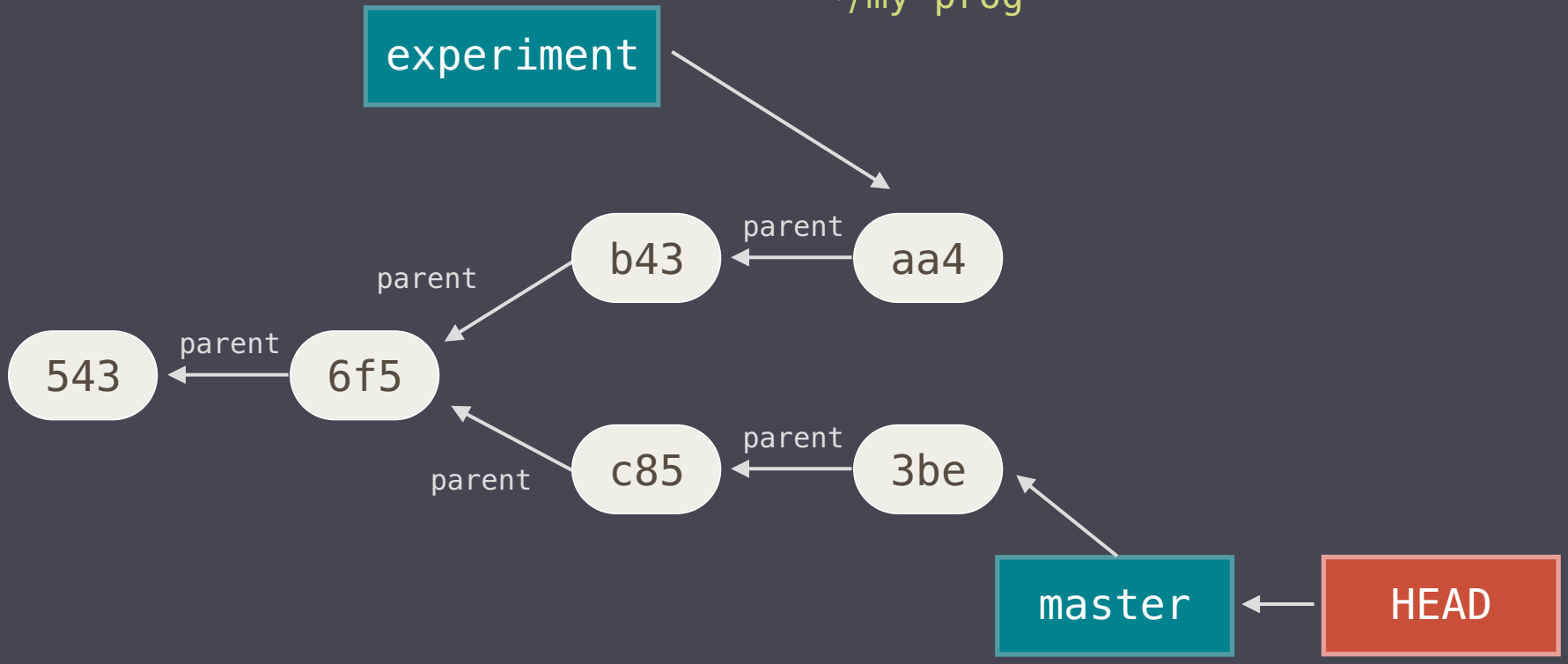
```
~/my-prog git checkout master
~/my-prog git add
~/my-prog git commit
~/my-prog
```



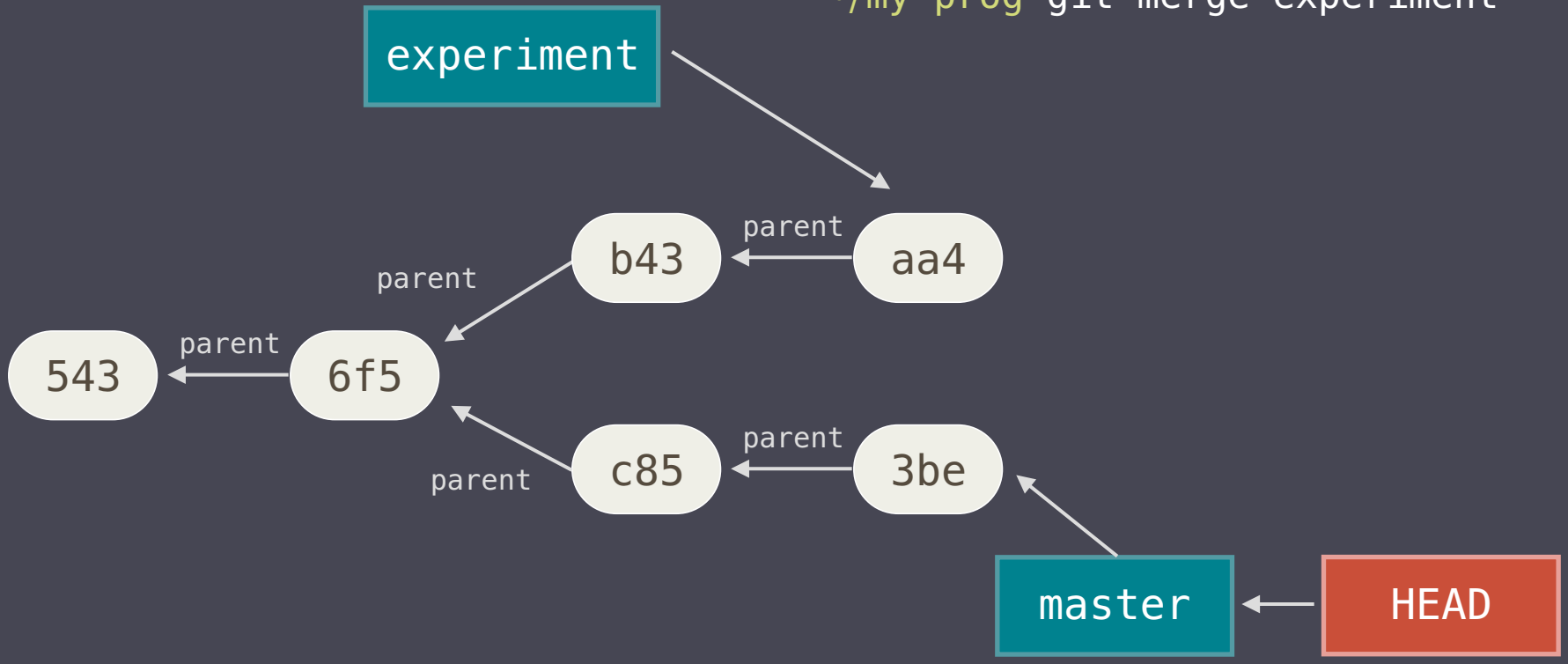
```
~/my-prog git checkout master
~/my-prog git add
~/my-prog git commit
~/my-prog git add
~/my-prog
```



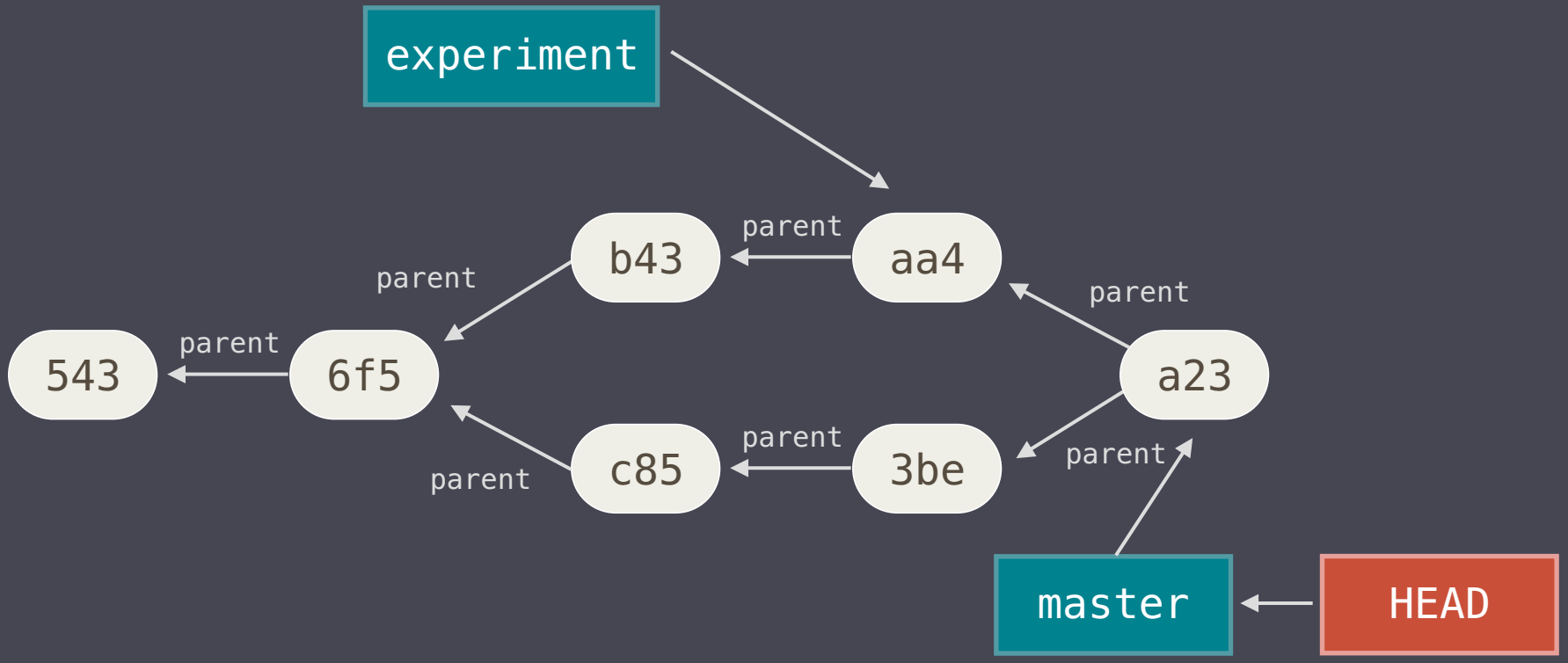
```
~/my-prog git checkout master
~/my-prog git add
~/my-prog git commit
~/my-prog git add
~/my-prog git commit
~/my-prog
```



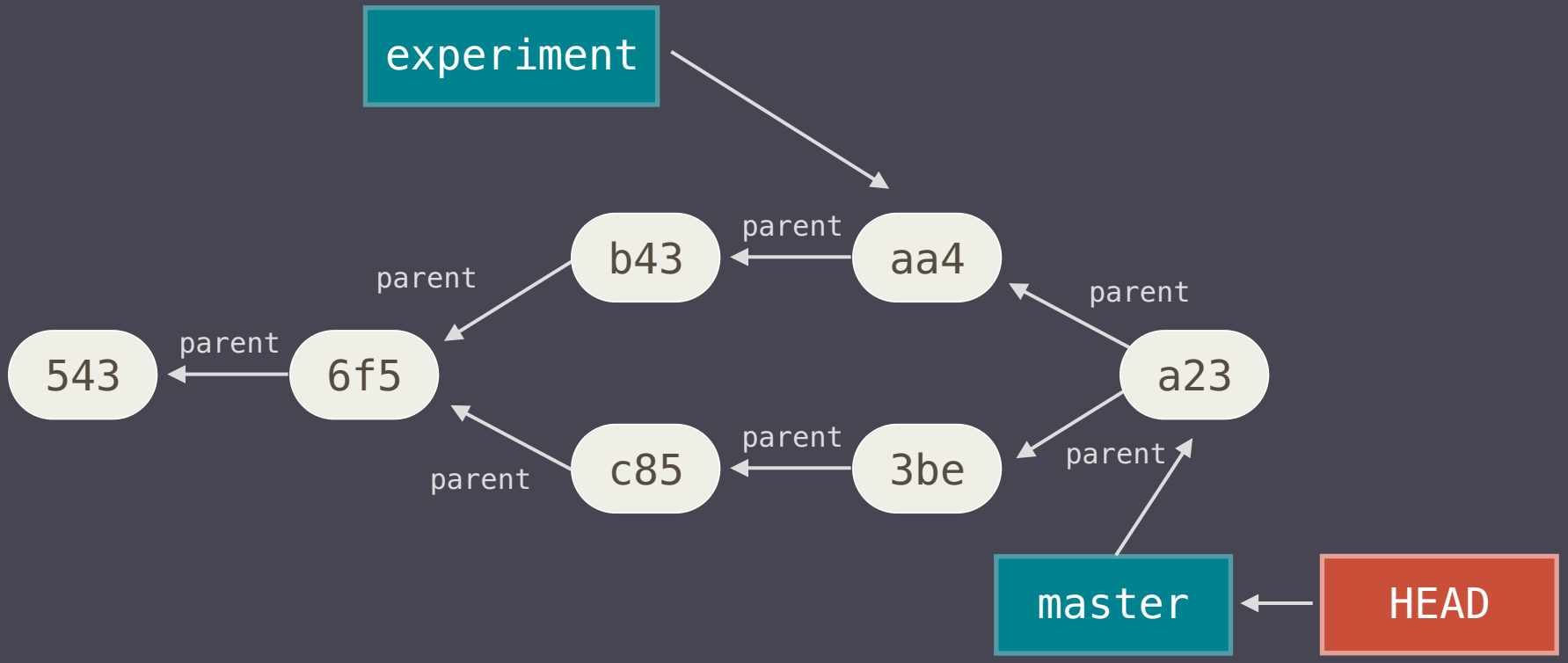
```
~/my-prog git checkout master
~/my-prog git add
~/my-prog git commit
~/my-prog git add
~/my-prog git commit
~/my-prog git merge experiment
```



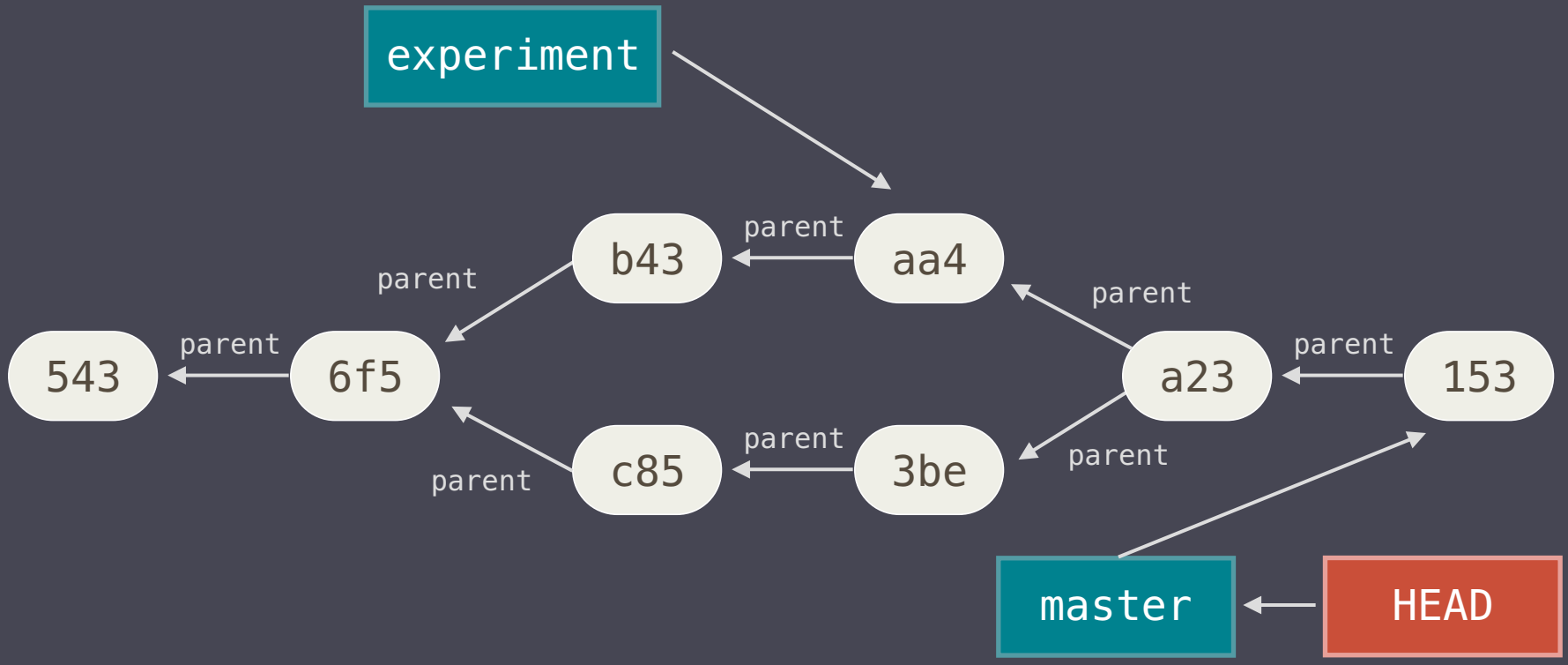
```
~/my-prog git merge experiment
~/my-prog
```



```
~/my-prog git merge experiment
~/my-prog git add
~/my-prog
```



```
~/my-prog git merge experiment
~/my-prog git add
~/my-prog git commit
~/my-prog
```



Merge Konflikte

- Merge Konflikte treten auf, wenn die gleiche Datei in den beiden Branches auf konflingierende Weise geändert wurden.
- Führt zu Pausieren des Mergevorgangs
- Beide Änderungen sind in der Konfliktdatei enthalten
- Die Konfliktdatei muss manuell editiert werden
- Danach kann der Merge fortgesetzt werden

```
<<<<<<< HEAD:index.html
<div id="footer">contact :
email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

Demo 2

(branch, checkout *commit*, merge, cherrypick, blame)

Die wichtigsten Befehle (3 / 4)

- **`git checkout` *`commit`***

  Commit → Index / Arbeitskopie

- **`git rm`**

  Datei aus Index und Arbeitskopie löschen

- **`git blame`**

  Letzte Änderung an Datei Zeile für Zeile anzeigen

- **`git merge`**

  Mehrere Branches zusammenführen

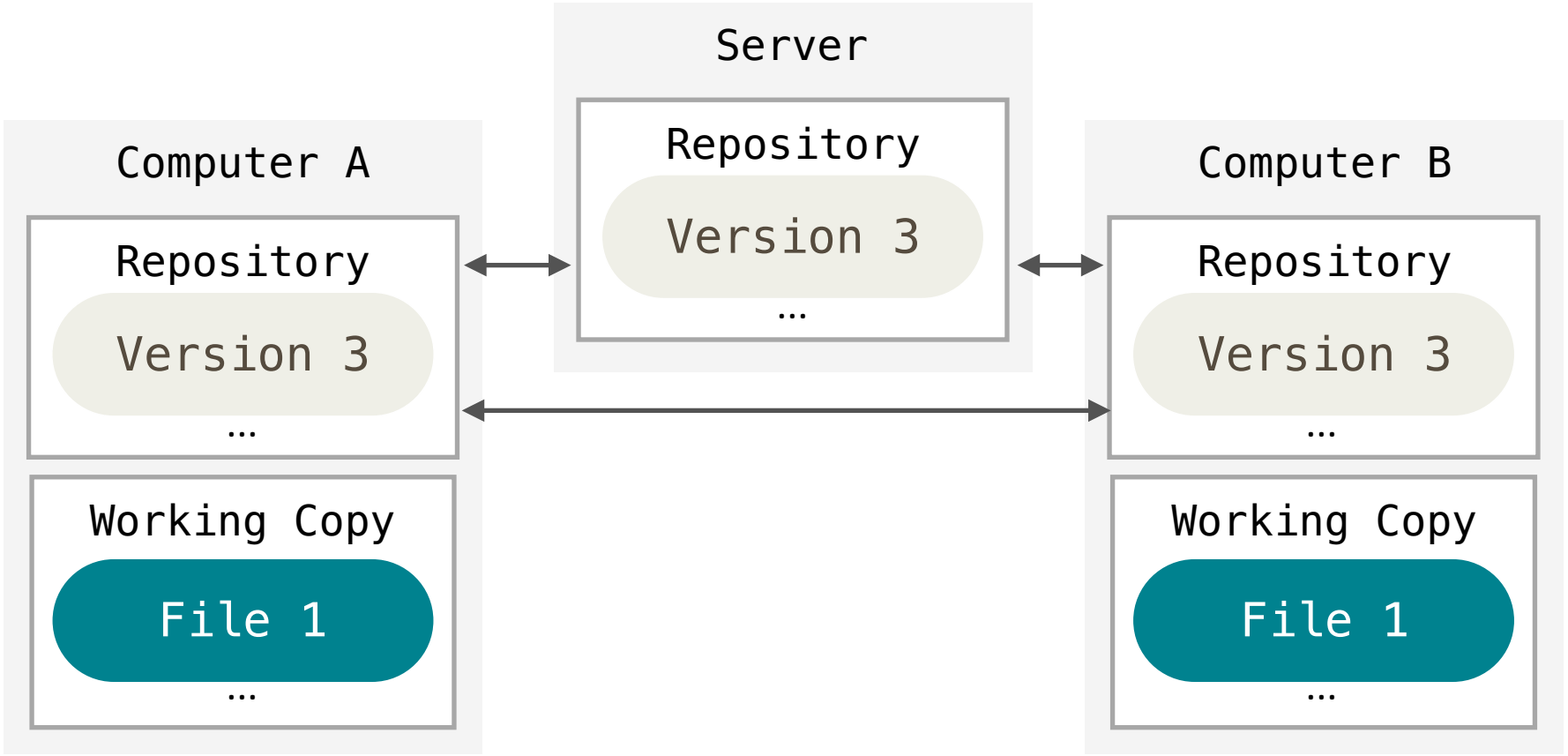
- **`git cherrypick`**

  Einzelnen Commit an aktuellen Branch dranhängen

Arbeiten im Team

Verteilte Versionskontrolle mit Server

- Ein zentrales Repository ermöglicht vereinfachten Austausch



Begriffe

- **Remote Repository**
  - Ein externes Repository im Netzwek / Internet
- **Fetch**
  - Die lokale Kopie des Remote Repositories im eigenen Repo aktualisieren
- **Push**
  - Die lokale Kopie des Remote Repositories zum externen Repo hochladen
- **Remote Branch**
  - Referenz auf einen Branch in einem Remote Repository
- **Remote Tracking Branch**
  - Remote Branch, welcher automatisch aktualisiert wird
- **Tracking Branch**
  - Lokale Kopie eines Remote Tracking Branches mit einer Referenz auf den "Upstream Branch"

Begriffe

- **Remote Repository**
  - Ein externes Repository im Netzwek / Internet
- **Fetch**
  - Die lokale Kopie des Remote Repositories im eigenen Repo aktualisieren
- **Push**
  - Die lokale Kopie des Remote Repositories zum externen Repo hochladen
- **Remote Branch**
  - Referenz auf einen Branch in einem Remote Repository
- **Remote Tracking Branch**
  - Remote Branch, welcher automatisch aktualisiert wird
- **Tracking Branch**
  - Lokale Kopie eines Remote Tracking Branches mit einer Referenz auf den "Upstream Branch"

Begriffe

- **Remote Repository**
  - Ein externes Repository im Netzwek / Internet
- **Fetch**
  - Die lokale Kopie des Remote Repositories im eigenen Repo aktualisieren
- **Push**
  - Die lokale Kopie des Remote Repositories zum externen Repo hochladen
- **Remote Branch**
  - Referenz auf einen Branch in einem Remote Repository
- **Remote Tracking Branch**
  - Remote Branch, welcher automatisch aktualisiert wird
- **Tracking Branch**
  - Lokale Kopie eines Remote Tracking Branches mit einer Referenz auf den "Upstream Branch"

Begriffe

- **Remote Repository**
  - Ein externes Repository im Netzwek / Internet
- **Fetch**
  - Die lokale Kopie des Remote Repositories im eigenen Repo aktualisieren
- **Push**
  - Die lokale Kopie des Remote Repositories zum externen Repo hochladen
- **Remote Branch**
  - Referenz auf einen Branch in einem Remote Repository
- **Remote Tracking Branch**
  - Remote Branch, welcher automatisch aktualisiert wird
- **Tracking Branch**
  - Lokale Kopie eines Remote Tracking Branches mit einer Referenz auf den "Upstream Branch"

Begriffe

- **Remote Repository**
  - Ein externes Repository im Netzwek / Internet
- **Fetch**
  - Die lokale Kopie des Remote Repositories im eigenen Repo aktualisieren
- **Push**
  - Die lokale Kopie des Remote Repositories zum externen Repo hochladen
- **Remote Branch**
  - Referenz auf einen Branch in einem Remote Repository
- **Remote Tracking Branch**
  - Remote Branch, welcher automatisch aktualisiert wird
- **Tracking Branch**
  - Lokale Kopie eines Remote Tracking Branches mit einer Referenz auf den "Upstream Branch"

Begriffe

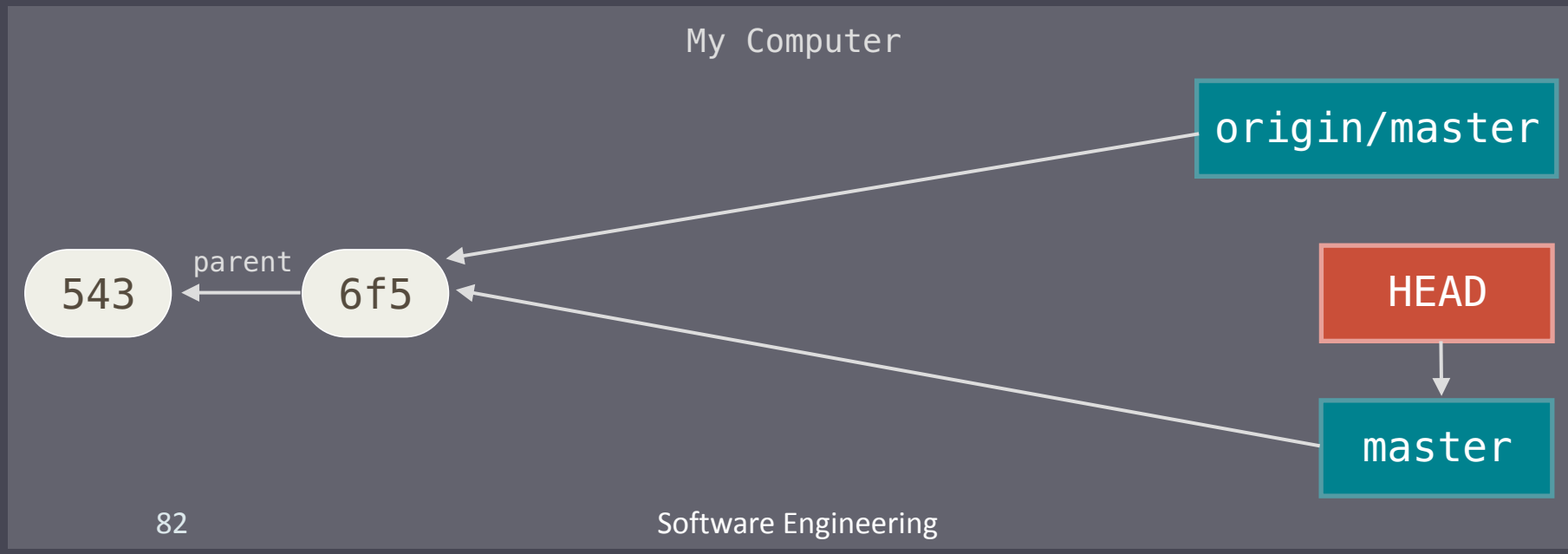
- **Remote Repository**
  - Ein externes Repository im Netzwek / Internet
- **Fetch**
  - Die lokale Kopie des Remote Repositories im eigenen Repo aktualisieren
- **Push**
  - Die lokale Kopie des Remote Repositories zum externen Repo hochladen
- **Remote Branch**
  - Referenz auf einen Branch in einem Remote Repository
- **Remote Tracking Branch**
  - Remote Branch, welcher automatisch aktualisiert wird
- **Tracking Branch**
  - Lokale Kopie eines Remote Tracking Branches mit einer Referenz auf den "Upstream Branch"

Begriffe

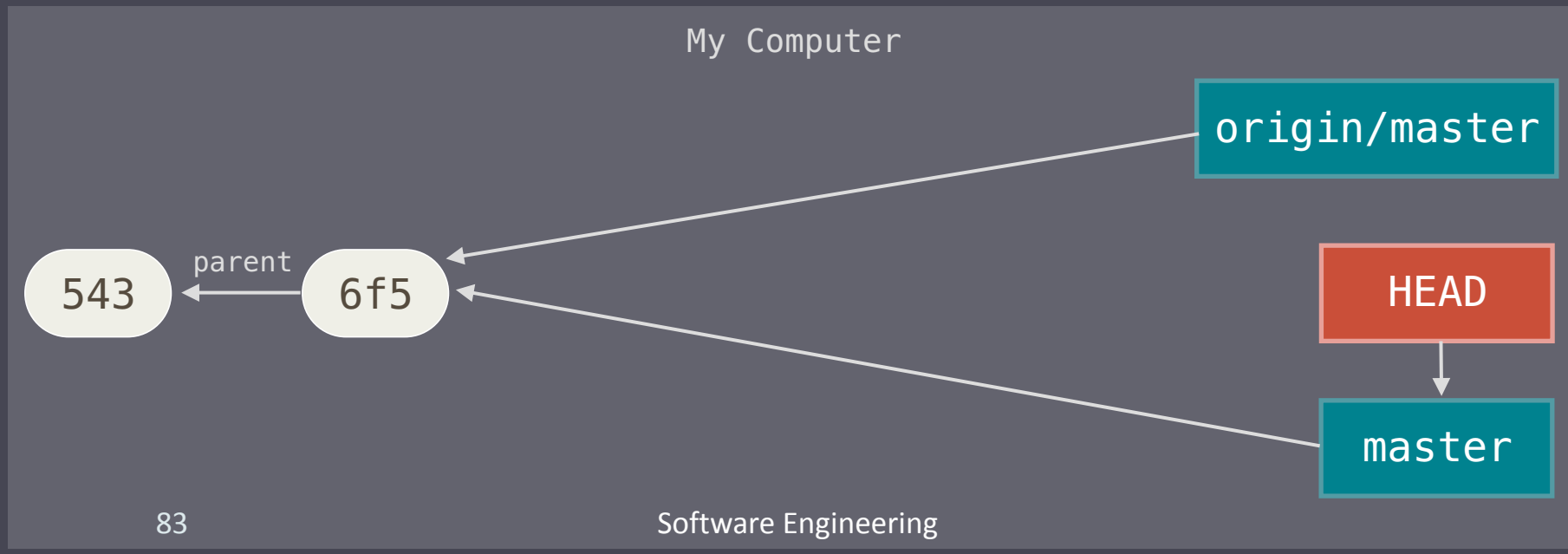
- **Remote Repository**
  - Ein externes Repository im Netzwek / Internet
- **Fetch**
  - Die lokale Kopie des Remote Repositories im eigenen Repo aktualisieren
- **Push**
  - Die lokale Kopie des Remote Repositories zum externen Repo hochladen
- **Remote Branch**
  - Referenz auf einen Branch in einem Remote Repository
- **Remote Tracking Branch**
  - Remote Branch, welcher automatisch aktualisiert wird
- **Tracking Branch**
  - Lokale Kopie eines Remote Tracking Branches mit einer Referenz auf den "Upstream Branch"

```
~/project git clone me@git.company.com:project
```

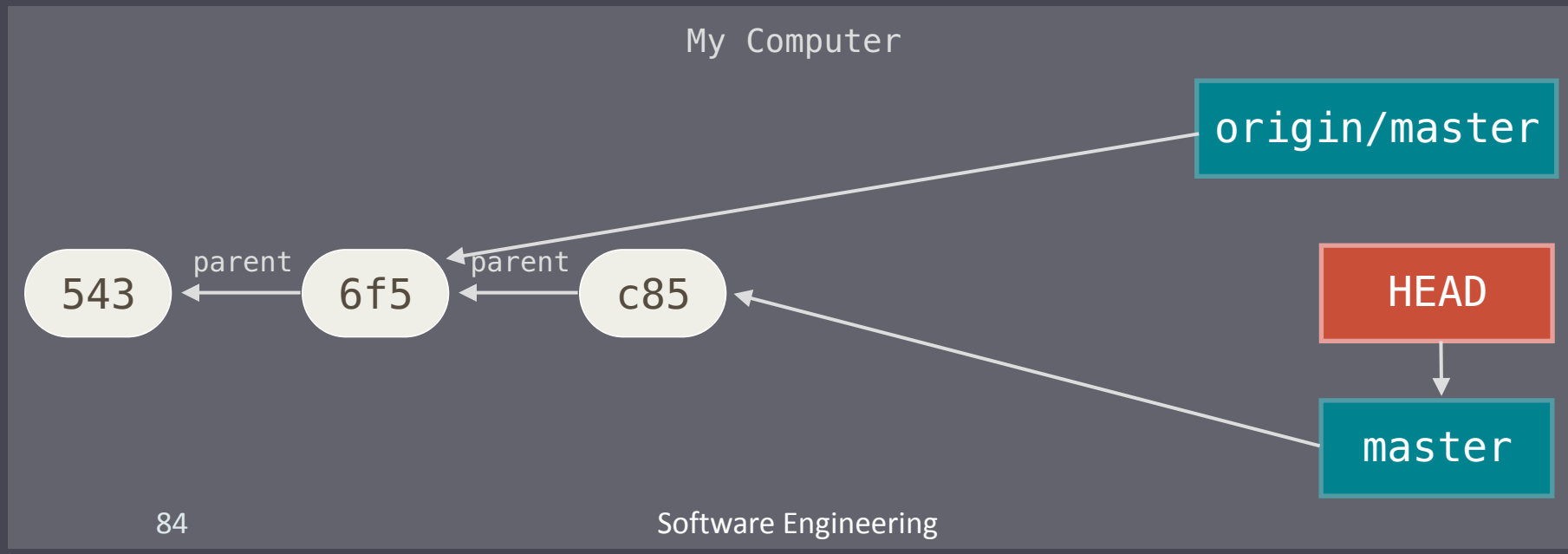
```
~/project git clone me@git.company.com:project
~/project
```



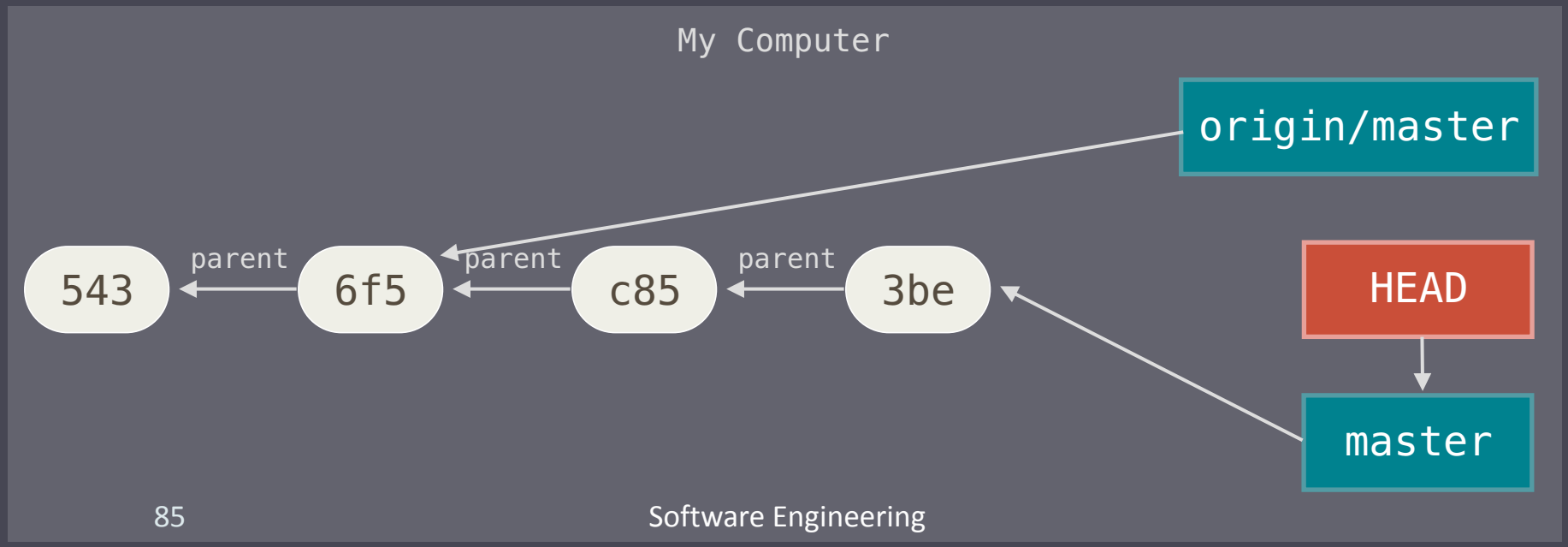
```
~/project git add
~/project git commit
~/project
```



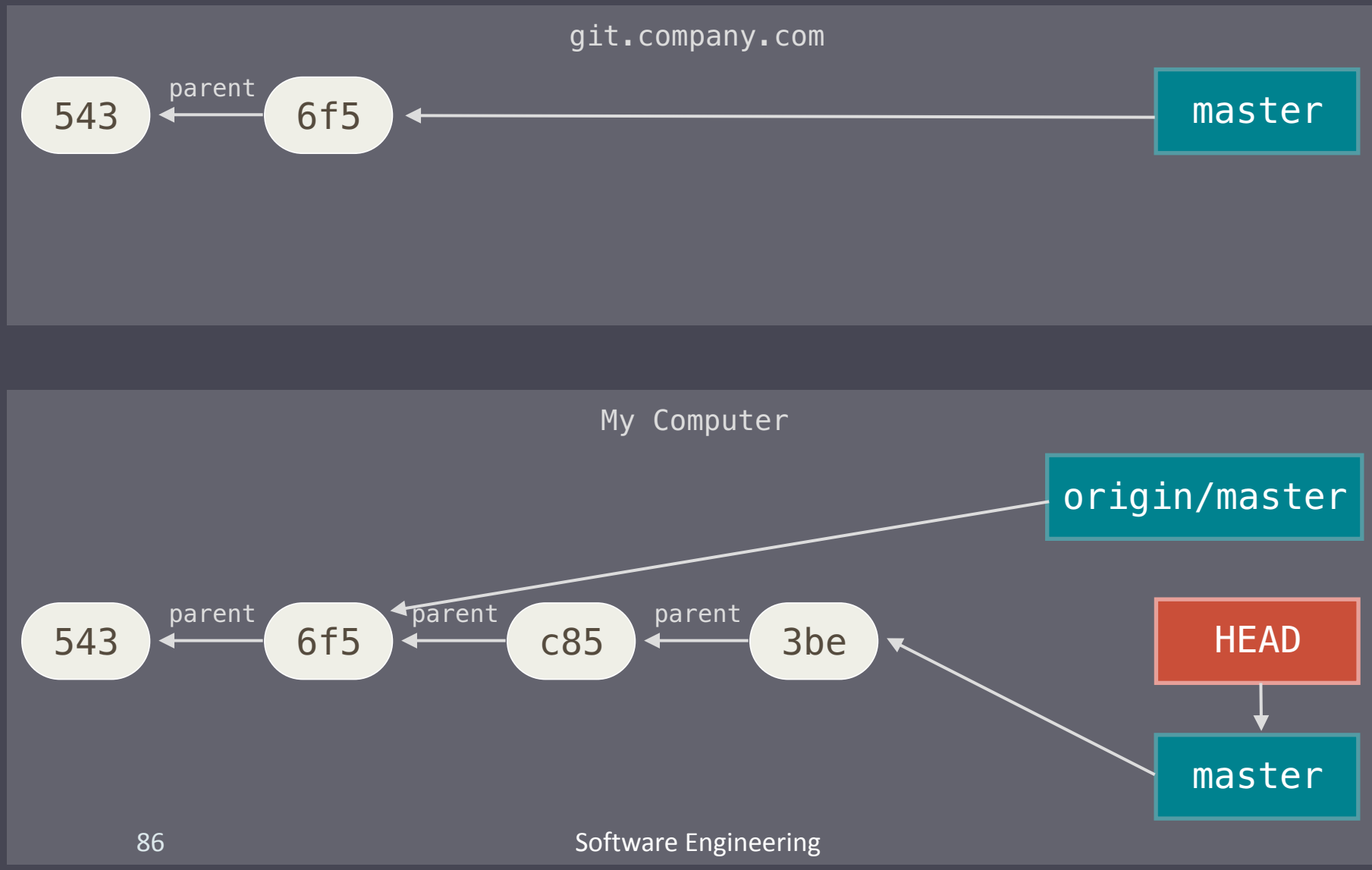
```
~/project git add
~/project git commit
~/project
```



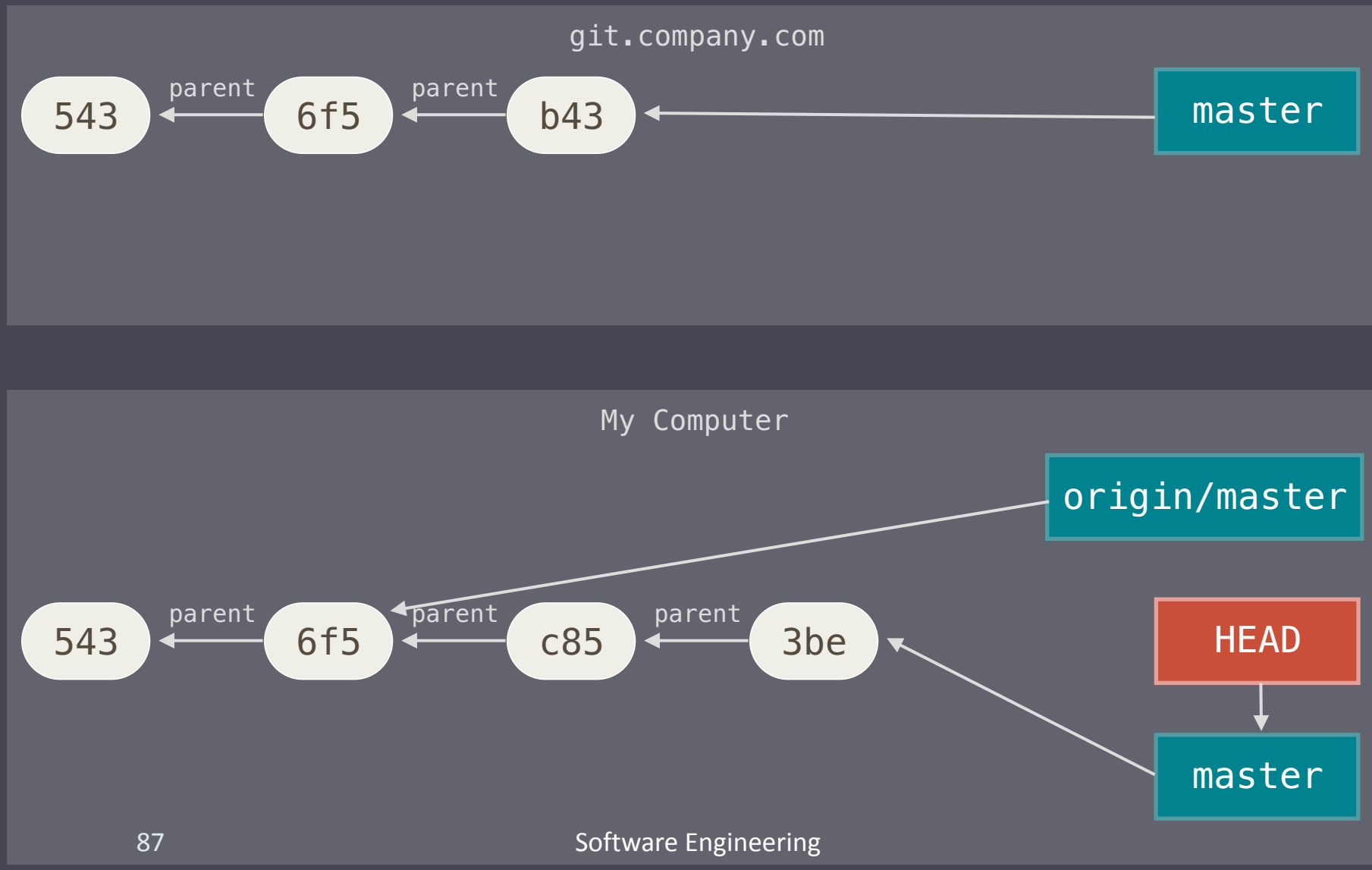
```
~/project git add
~/project git commit
~/project
```



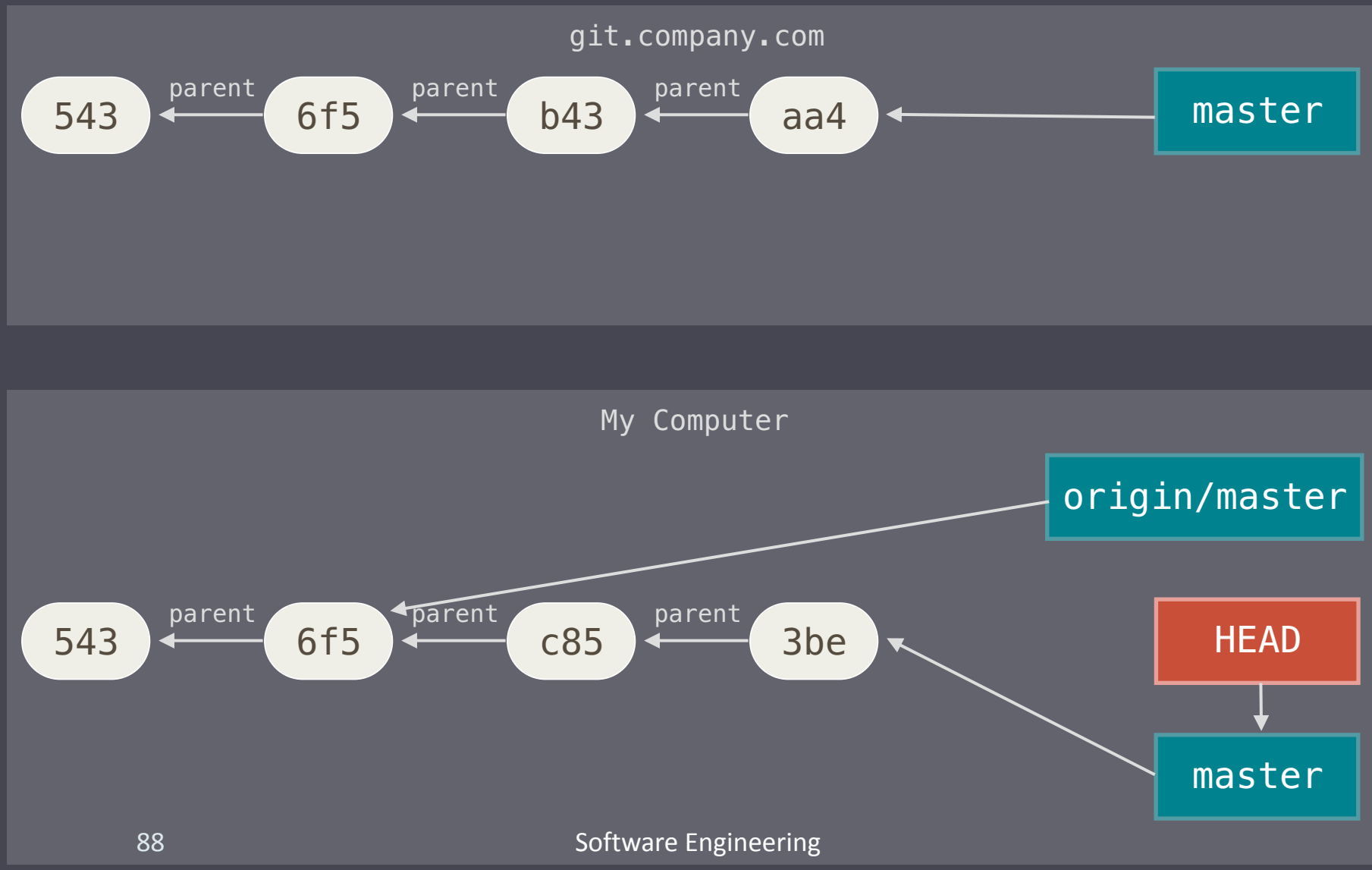
Jemand anderes arbeitet parallel und aktualisiert das Remote Repository vor uns



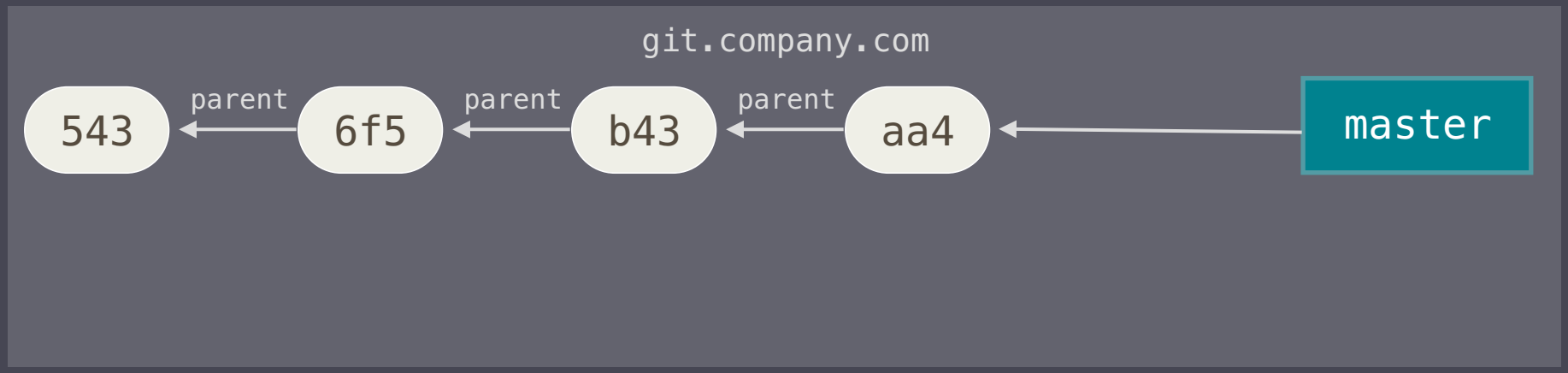
Jemand anderes arbeitet parallel und aktualisiert das Remote Repository vor uns

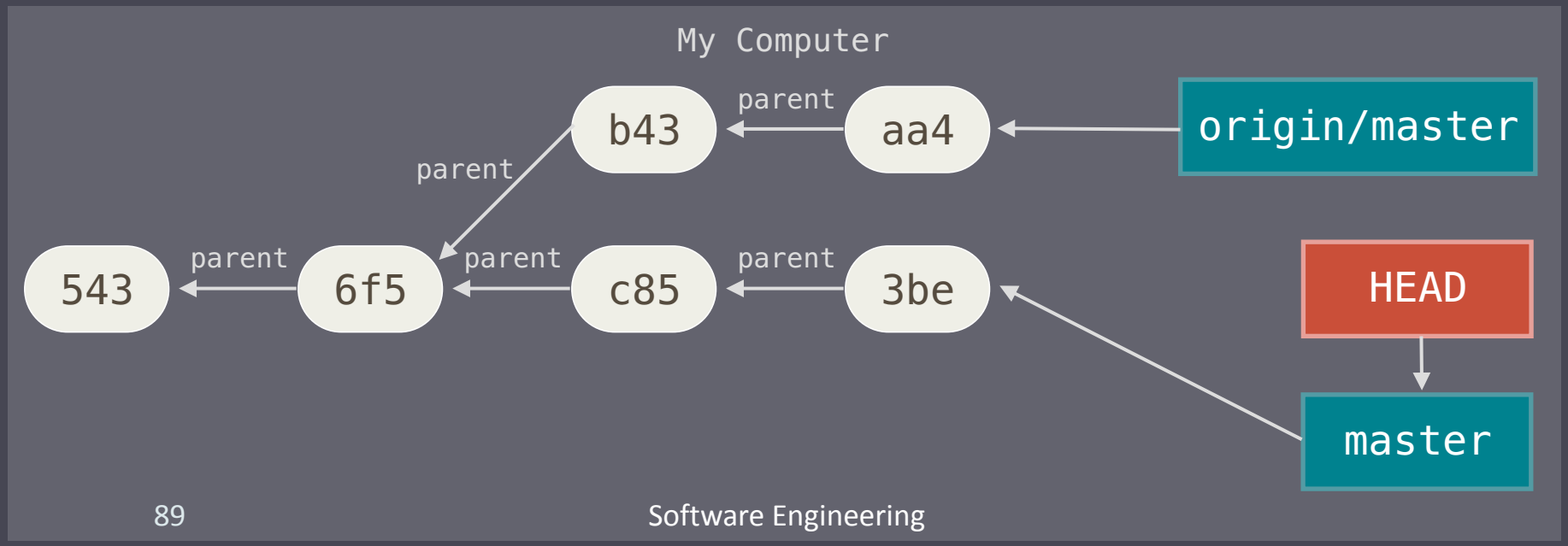


Jemand anderes arbeitet parallel und aktualisiert das Remote Repository vor uns

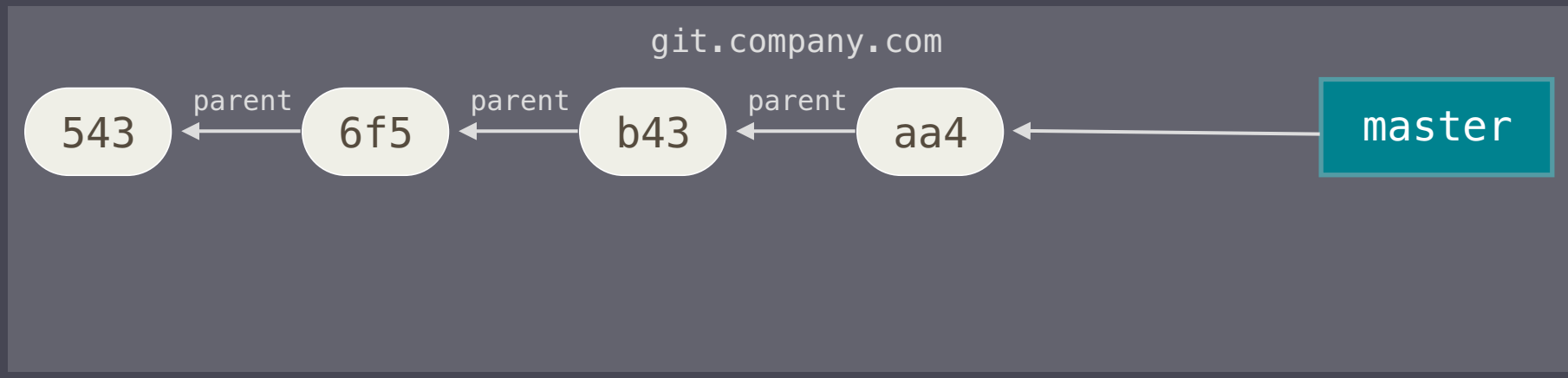


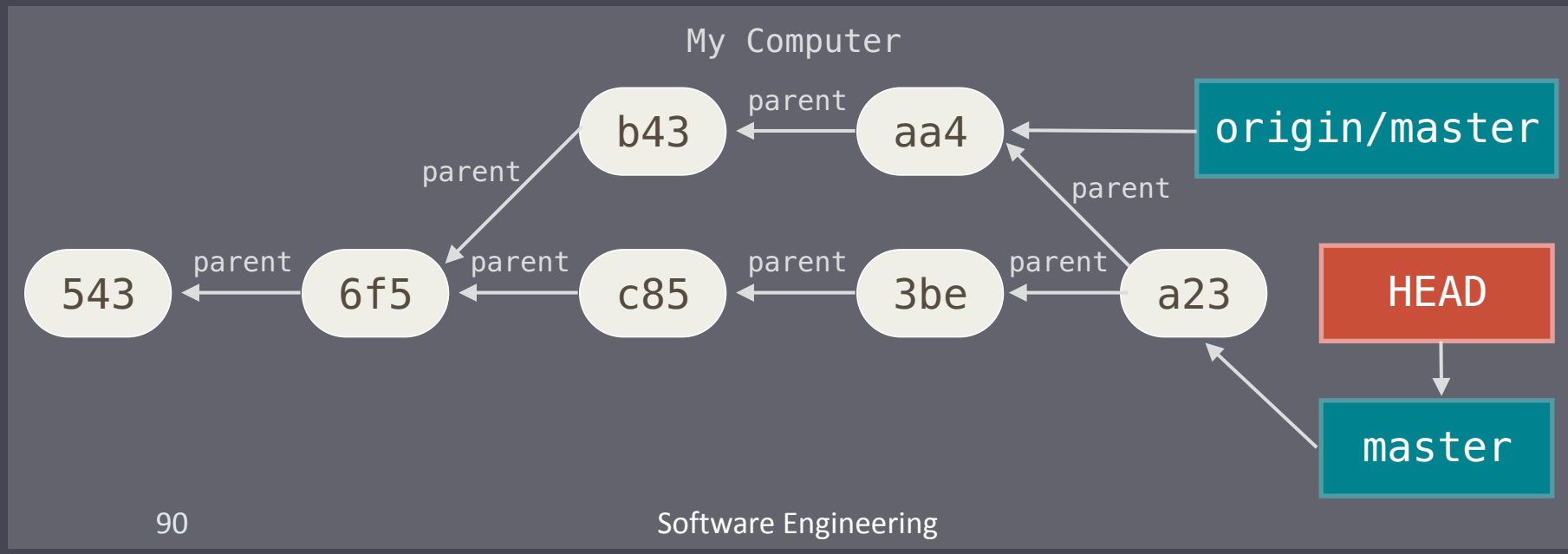
```
~/project git fetch
~/project
```



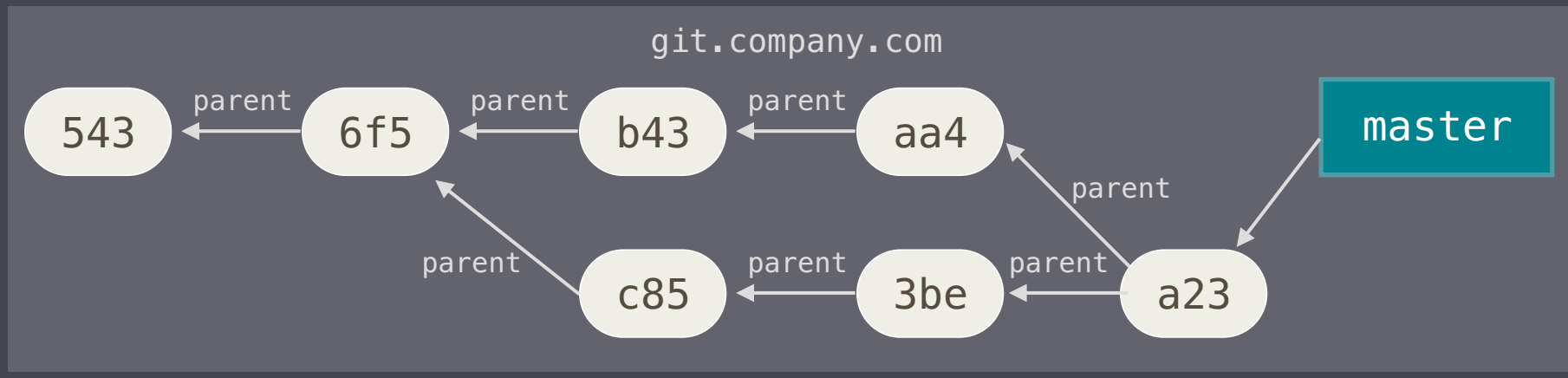


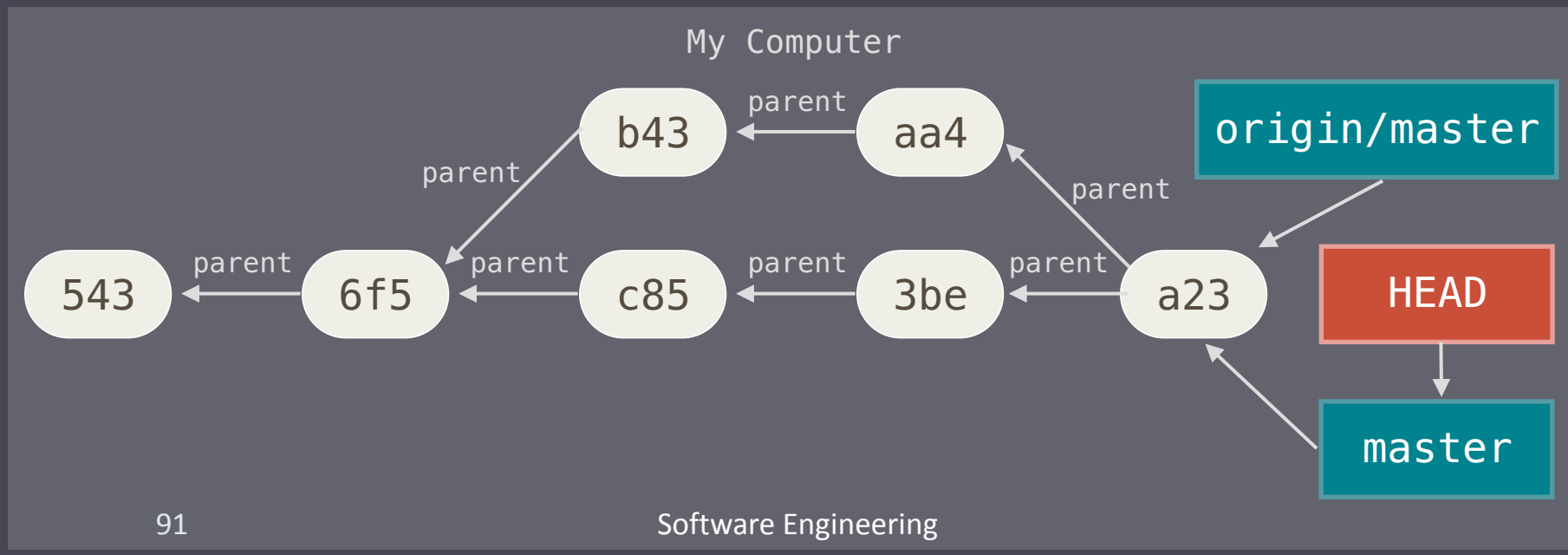
```
~/project git merge origin/master
~/project
```





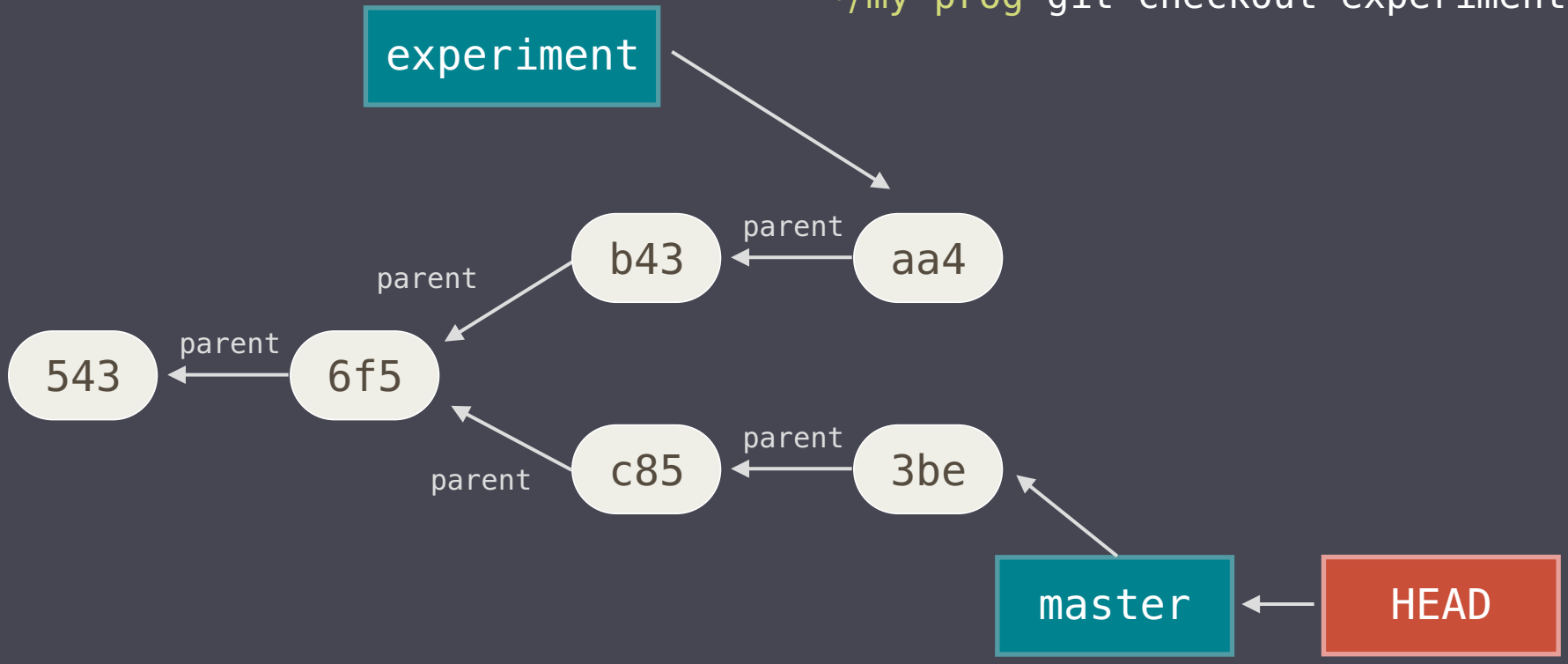
```
~/project git push origin master:master
~/project
```



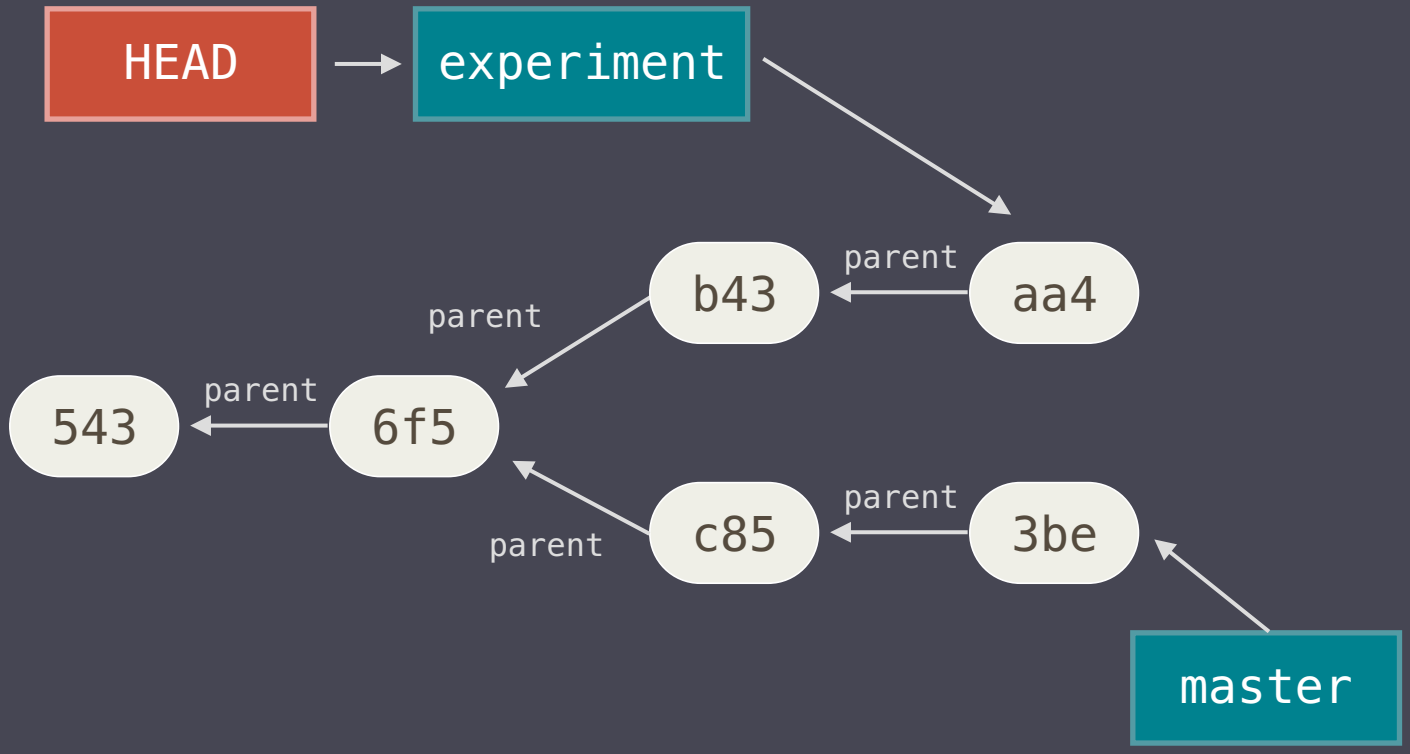


Rebase Workflow

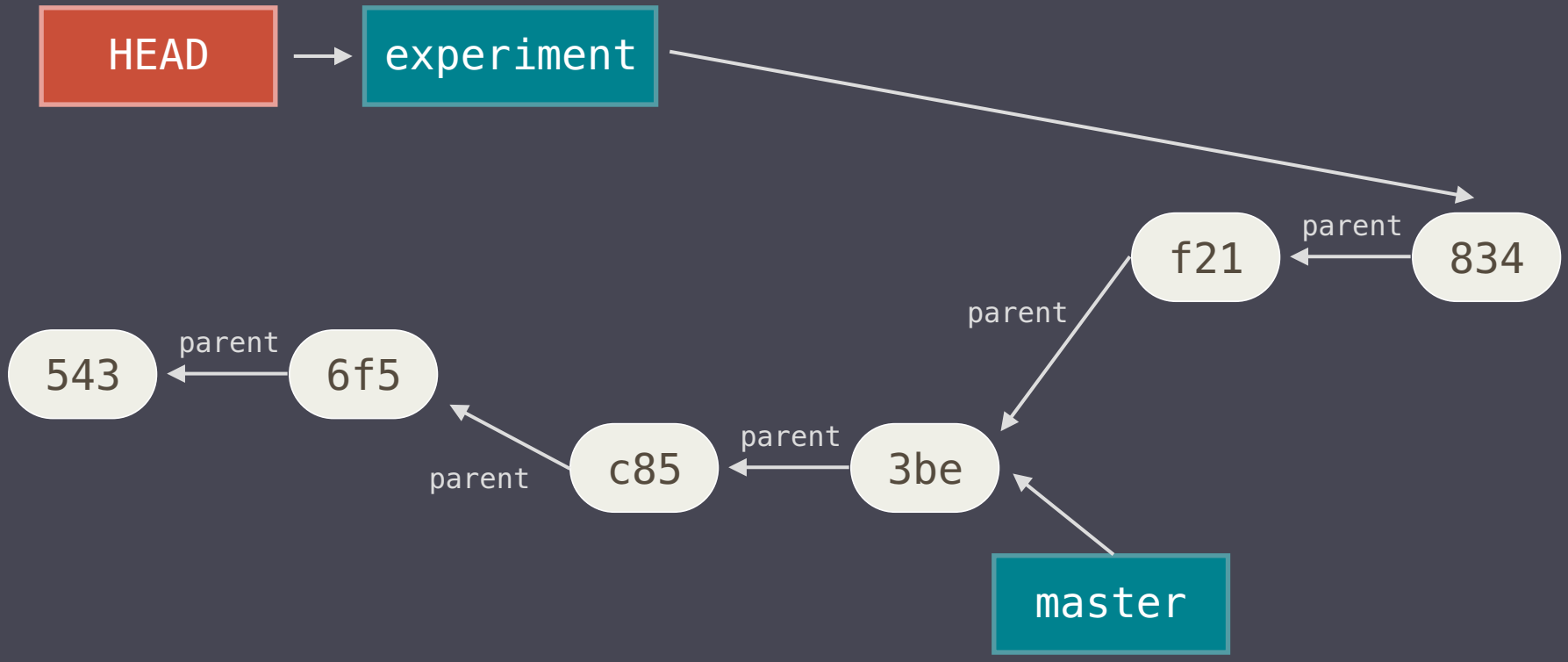
```
~/my-prog git checkout master
~/my-prog git add
~/my-prog git commit
~/my-prog git add
~/my-prog git commit
~/my-prog git checkout experiment
```



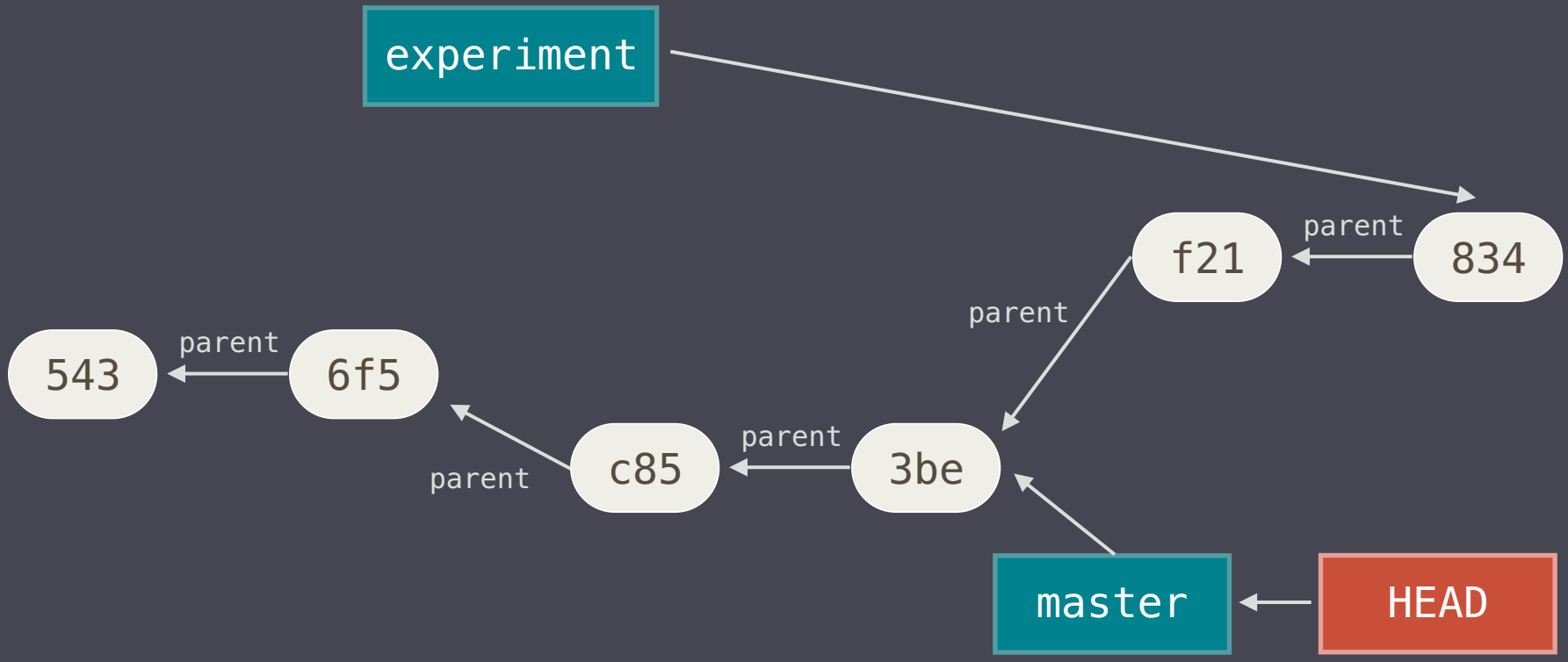
```
~/my-prog git checkout experiment
~/my-prog git rebase master
```



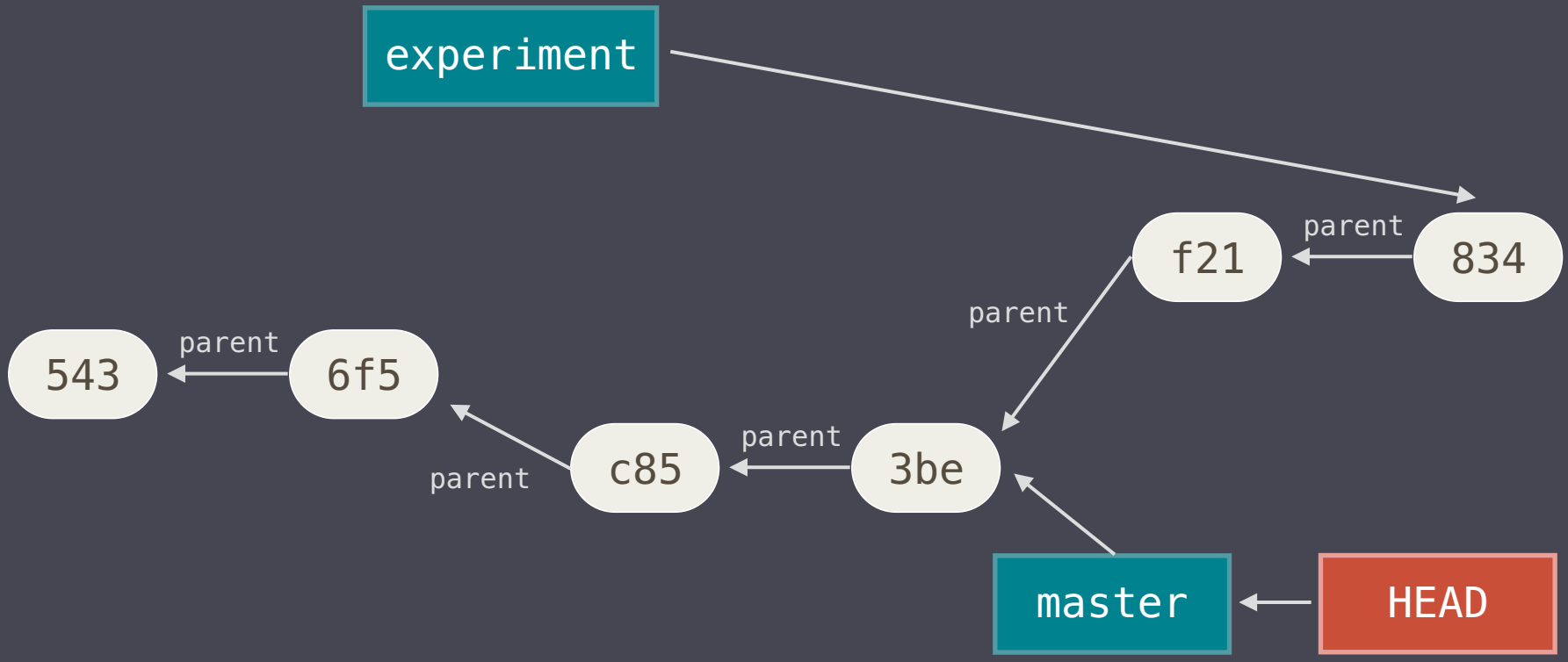
```
~/my-prog git checkout experiment
~/my-prog git rebase master
~/my-prog
```



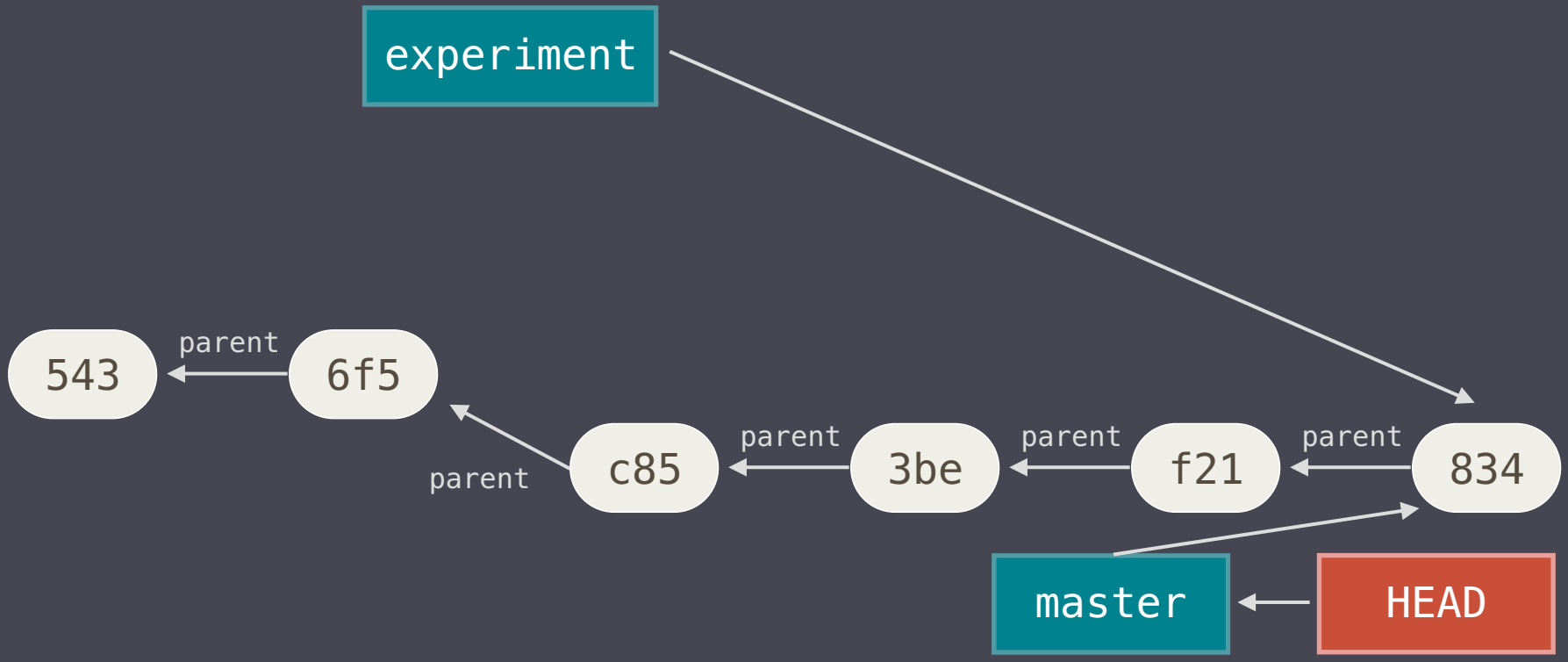
```
~/my-prog git checkout experiment
~/my-prog git rebase master
~/my-prog git checkout master
~/my-prog
```



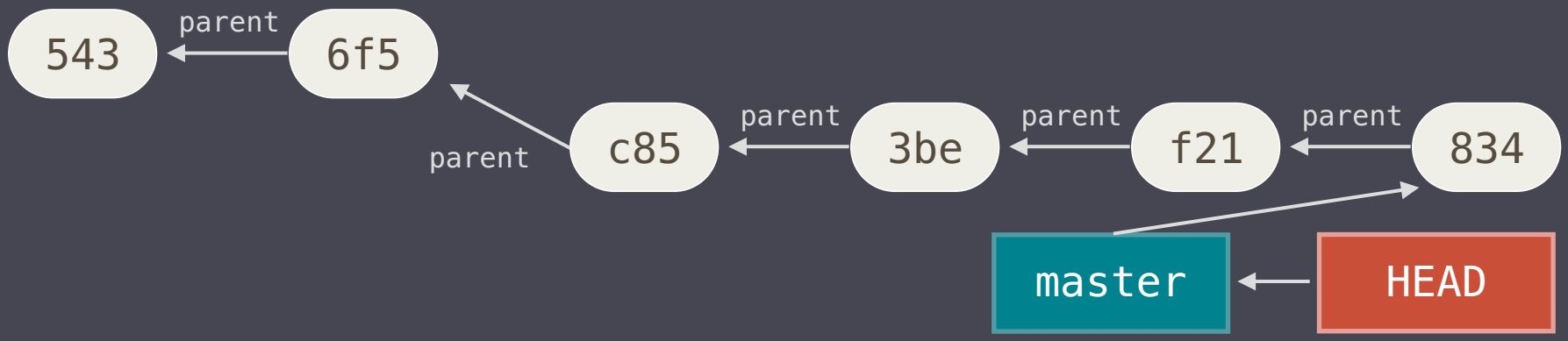
```
~/my-prog git checkout experiment
~/my-prog git rebase master
~/my-prog git checkout master
~/my-prog git merge experiment
```



```
~/my-prog git checkout experiment
~/my-prog git rebase master
~/my-prog git checkout master
~/my-prog git merge experiment
~/my-prog
```



```
~/my-prog git checkout experiment
~/my-prog git rebase master
~/my-prog git checkout master
~/my-prog git merge experiment
~/my-prog git branch -d experiment
~/my-prog
```



Demo 3

(clone, fetch, push, pull, rebase)

Die wichtigsten Befehle (4 / 4)

- **`git clone`** *`url`*

  Remote Repo → Lokales Repo

- **`git fetch`** *`remote`*

  Remote Tracking Branches des lokalen Repos aktualisieren

- **`git push`** *`remote branch`*

  Lokales Repo → Remote Repo

- **`git pull`**

  *Bei **Tracking Branches**: Abkürzung für* `git fetch; git merge`

- **`git rebase`** *`branch`*

  Die Commits vom aktuellen Branch in Diffs umwandeln und den angegebenen Branch als Basis verwenden

Probeklausur

Tutorinnen & Tutoren gesucht für Info 1