Programmierung 2

DHBW Mannheim - WWI17SCB - May 2018

Opt1. Versionsverwaltung

Slides herunterladen

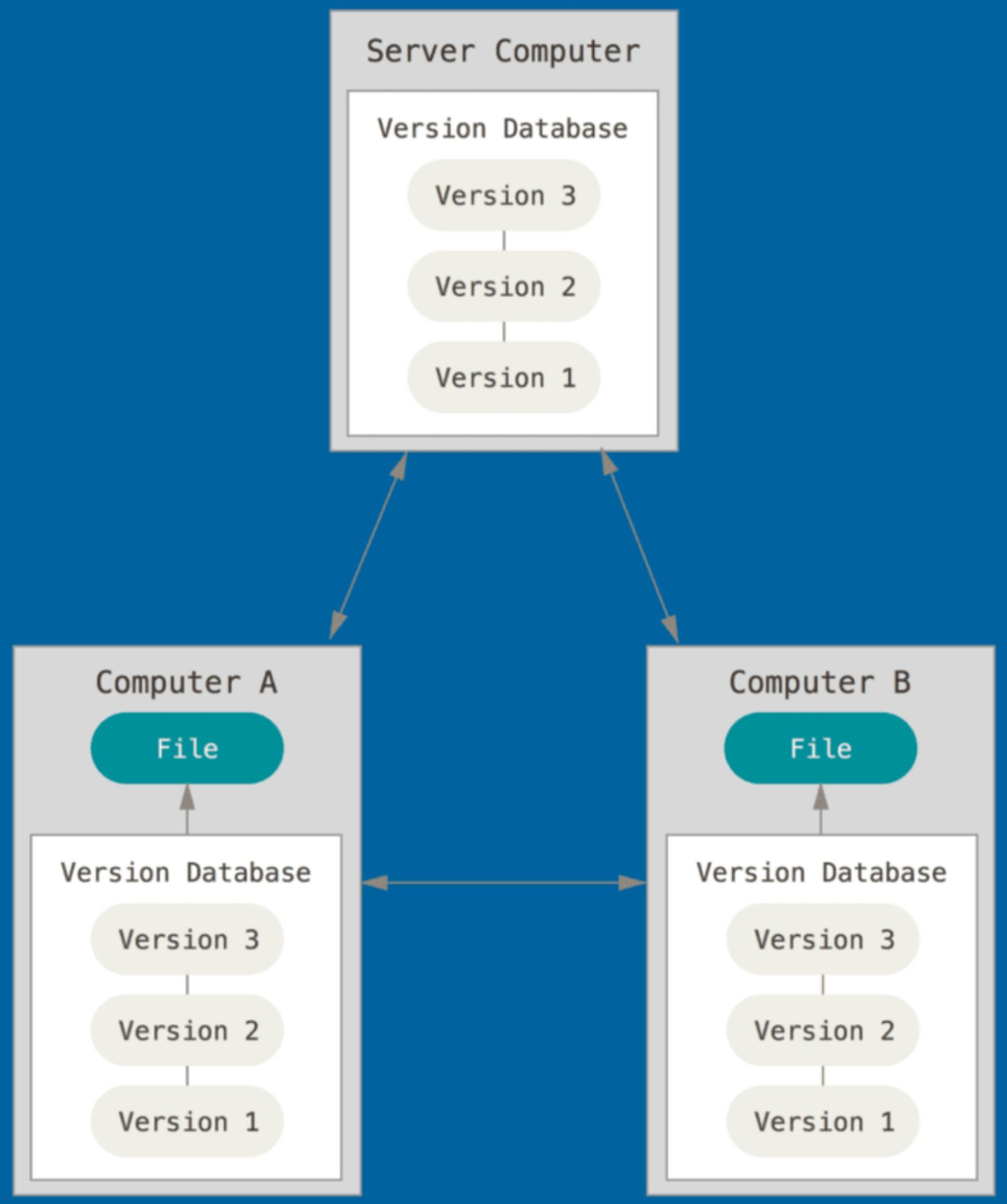
https://goo.gl/U45wY3

Was ist Versionsverwaltung?

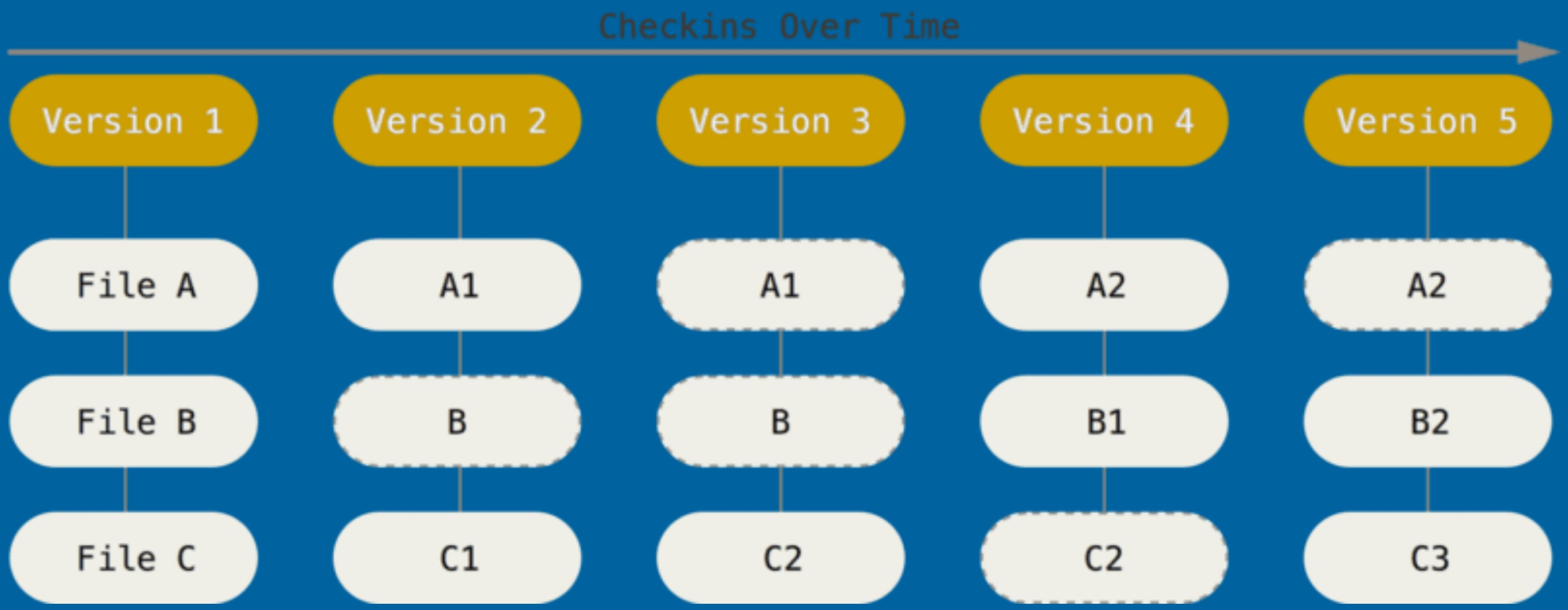
* Protokollieren von Änderungen an Dateien
* Verfolgung der Änderungshistorie über die Lebensdauer
* Herausfinden, wer Änderungen durchgeführt hat
* Änderungen finden, die Probleme verursachen
* Dateien auf einen früheren Zustand zurücksetzen
* Änderungen mit anderen teilen und deren Änderungen erhalten
* Grundlage für Continuous Integration / Delivery

Was ist Versionsverwaltung?

* Das mit Abstand meistverwendete VCS: Git (https://git-scm.com)
* Quelle: https://git-scm.com/book/en/v2



Source: https://git-scm.com/book/de/v2/Los-geht%E2%80%99s-Was-ist-Versionsverwaltung%3F



Source: https://git-scm.com/book/de/v2/Los-geht%E2%80%99s-Git-Grundlagen

Git Basics

* Pro Version (=Commit) wird eine SHA1-Checksum aller Dateien erstellt
* Sobald ein Commit erstellt ist, können die Änderungen praktisch nicht mehr verloren gehen
* Es ist möglich Commits zu löschen, dies sollte jedoch **nie** mit bereits veröffentlichten (mit anderen geteilten) Commits geschehen

--> *"Never change public history!"*

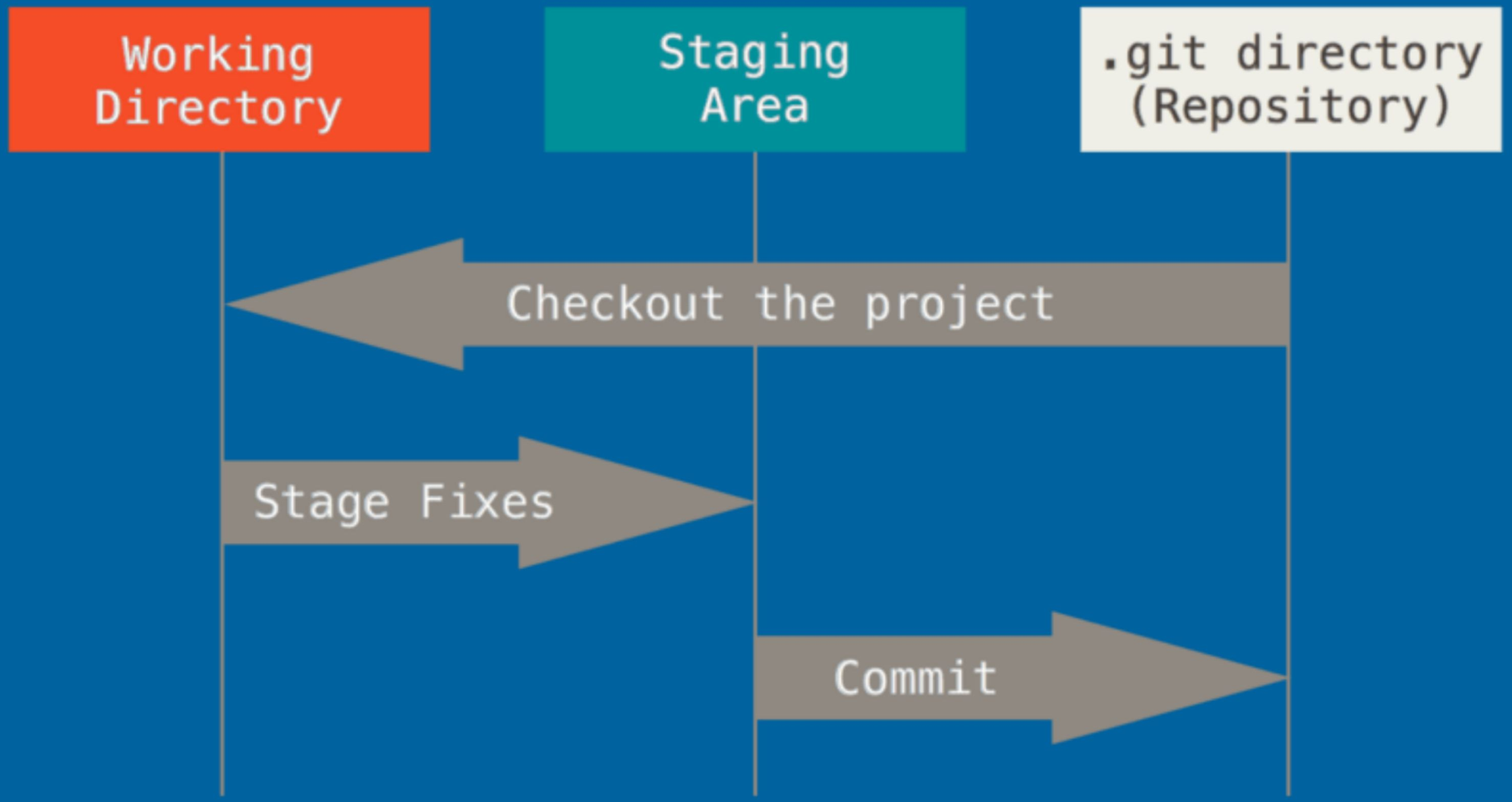
Git Basics - "Three Trees"

Arbeitsablauf mit Git:

1. Änderungen im Dateisystem ("Working Directory") machen
2. Bestimmte Änderungen für den nächsten Commit markieren ("Staging Area")
3. Änderungen committen, d.h. in die Git-History aufnehmen. ("Git Repository")

Ein Commit sollte funktional und logisch zusammengehörige Änderungen enthalten.

Git Basics - "Three Trees"



Source: https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

Git Basics - Verwendung

* Nutzung von git - normalerweise über das CLI
* Einfache Installation: https://desktop.github.com/
* Mac-only: z.B. `brew install git`
* Der erste Schritt nach der Installation: eigene Identität festlegen

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

Git Basics - GUIs

* Mac OS: GitX-dev https://github.com/rowanj/gitx
  * http://builds.phere.net/GitX/development/GitX-dev.dmg

* Windows & Mac OS: GitHub-Desktop
  * https://desktop.github.com/

Git Basics - Repository anlegen

```
cd /your/project/directory
git init
```

Dann beliebige Dateien in das Verzeichnis legen.

Es ist natürlich auch möglich, in einem bereits gefüllten Verzeichnis ein git-Repository anzulegen.

```
$ git init
Initialized empty Git repository in /Users/bastian/src/vl/gitdemo/.git/
```

Git Basics - Dateien in die Staging Area aufnehmen

Datei oder alle Dateien im aktuellen Verzeichnis in die Staging Area schieben - damit werden sie in den nächsten Commit aufgenommen.

```
git add .
```

oder:

```
git add file1.txt
```

Git Basics - Dateien aus Staging Area entfernen

Dateien "unstagen", d.h. aus der Staging Area entfernen.

Änderungen bleiben erhalten, aber der nächste Commit wird diese Dateien nicht enthalten.

```
git reset .
```

oder:

```
git reset file1.txt
```

Git Basics - Status überprüfen

Zustand des Working Directory und der Staging Area anzeigen:

`git status`

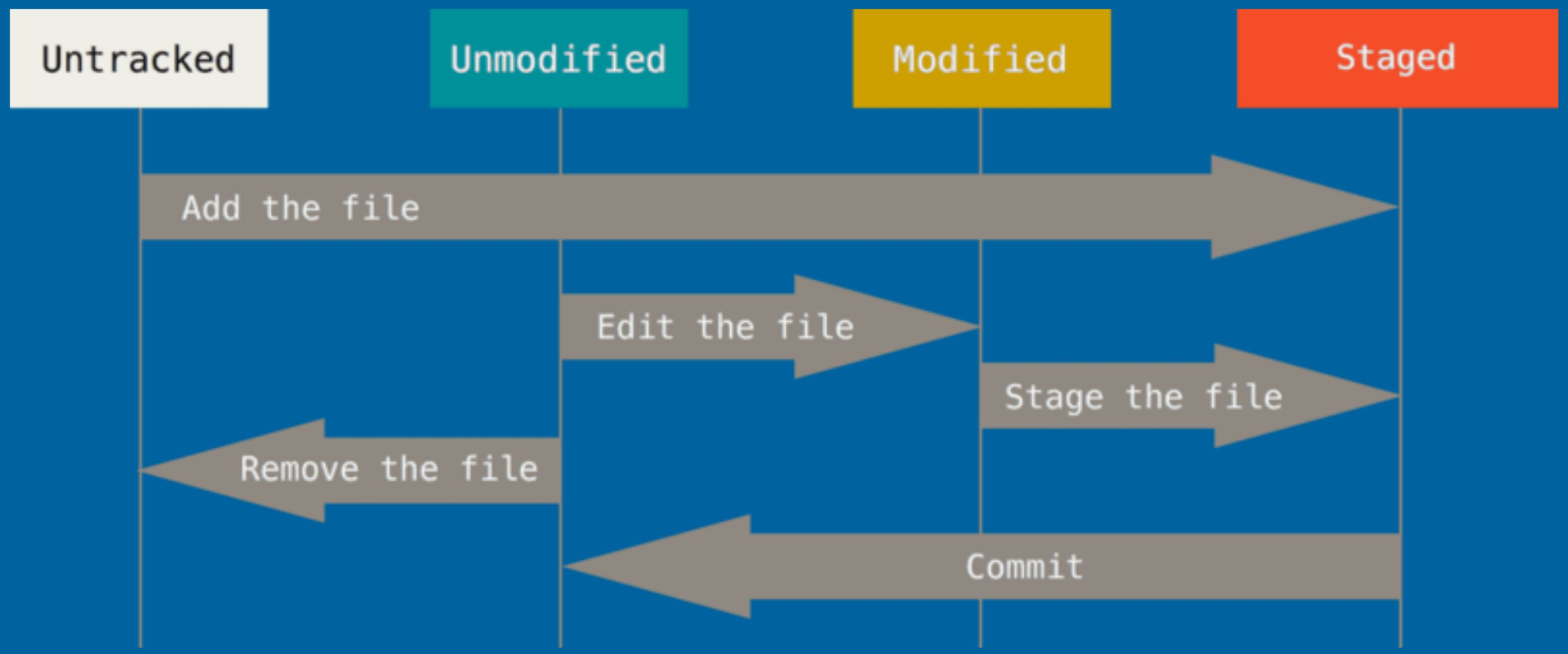
```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   file1.txt
```

Git Basics - Status überprüfen



Source: https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

Git Basics - Commit erstellen

Alle Dateien in der Staging Area committen:

`git commit -m "Initial commit"`

```
$ git commit -m "Initial commit"
[master (root-commit) 54470f0] Initial commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
```

Git Basics - Commits anzeigen

`git log`

```
$ git log
commit 54470f019fbd79f85e150fbedaa0320c5528b17a (HEAD -> master)
Author: Bastian Heist <bastian.heist@sandstorm.de>
Date:   Fri May 25 01:34:51 2018 +0200
    Initial commit
```

Mit Anzeige der Änderungen:

`git log -p`

Besser hierfür: Git-GUI wie gitk, GitX oder GitHub Desktop

Git Basics - Repository clonen

"Clonen" eines bereits bestehenden Repositories.

Beispiel: Neos-Projekt

```
git clone git@github.com:neos/neos-base-distribution.git
```

Git Basics - Remotes

Durch clonen wird automatisch die URL, von der gecloned wurde, als remote "origin" angelegt.

```
git remote -v
```

Git Basics - Remotes

Remotes können auch manuell hinzugefügt werden:

```
git remote add <remotename> https://github.com/some/repo
```

Git Basics - Remotes

Änderungen aus einem Remote-Repository namens `origin` holen:

```
git fetch origin
```

Achtung: Änderungen sind damit noch nicht mit dem eigenen Arbeitsstand zusammengeführt ("gemerged"). Es werden nur die remote existierenden Commits abgeholt und lokal bereitgestellt.

Git Basics - Remotes

Änderungen holen und direkt in den eigenen Stand mergen:

```
git pull
```

Hierdurch wird ein merge der Remote-Änderungen in den lokalen Branch (siehe nächste Folien) ausgeführt.

Für eine sauberere Commit-History ohne Merge-Commits:

```
git pull --rebase
```

Git Basics - Remotes

Eigene Änderungen in den Branch "master" der Remote "pushen":

```
git push origin master
```

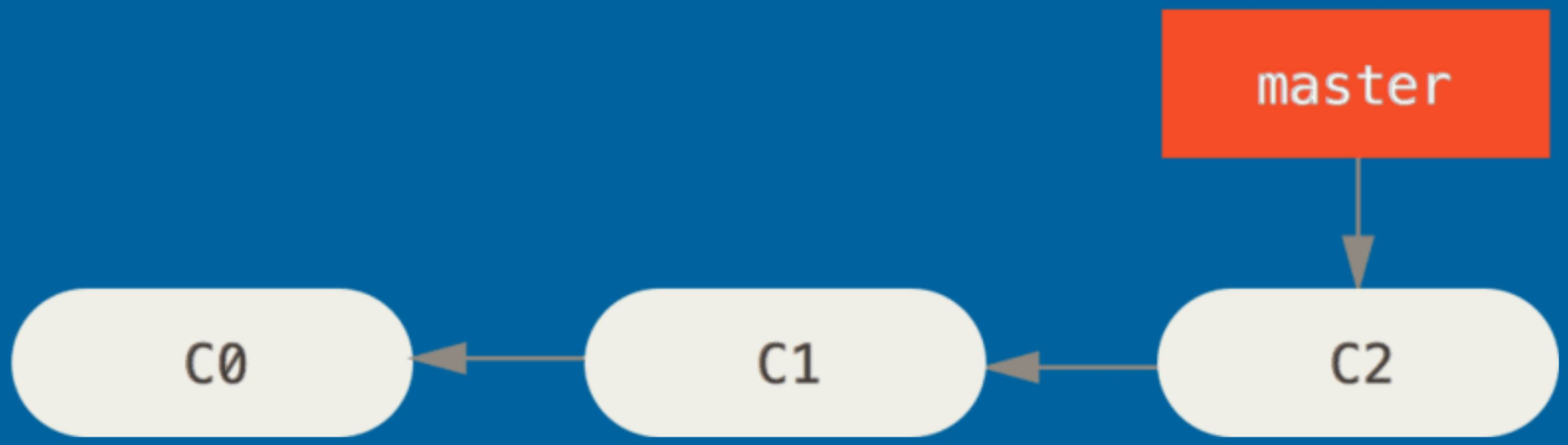
Falls der aktuelle Branch noch keinen "Upstream"-Branch hat, kann dieser zuvor festgelegt werden:

```
git push --set-upstream origin master
```

Danach reicht dann:

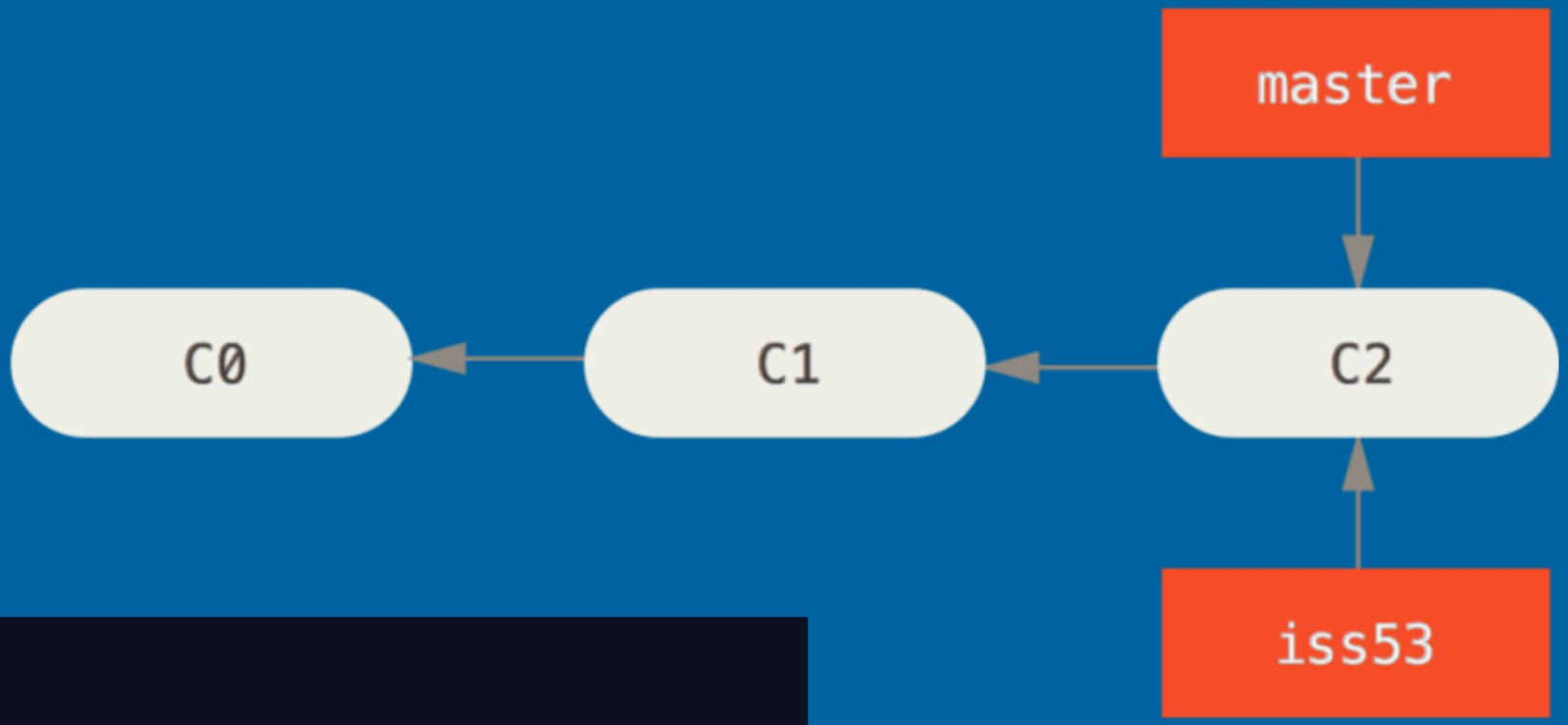
```
git push
```

Git Basics - Branching



Source: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

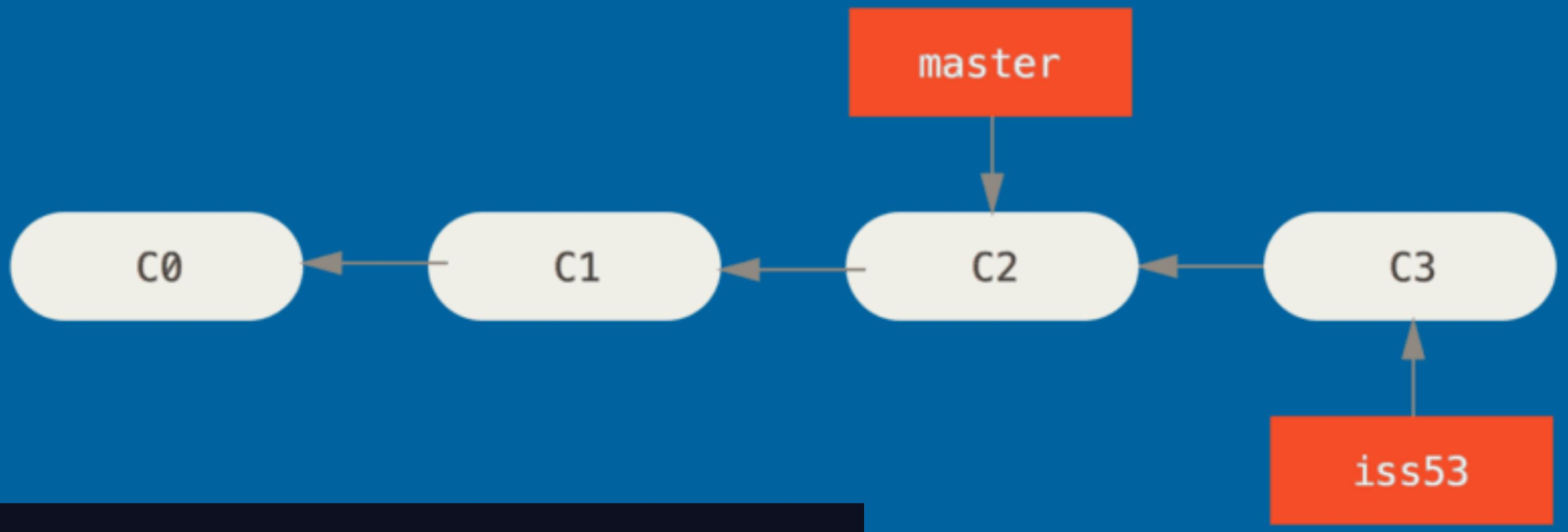
Git Basics - Branching




```
$ git branch iss53
$ git checkout iss53
```

Source: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

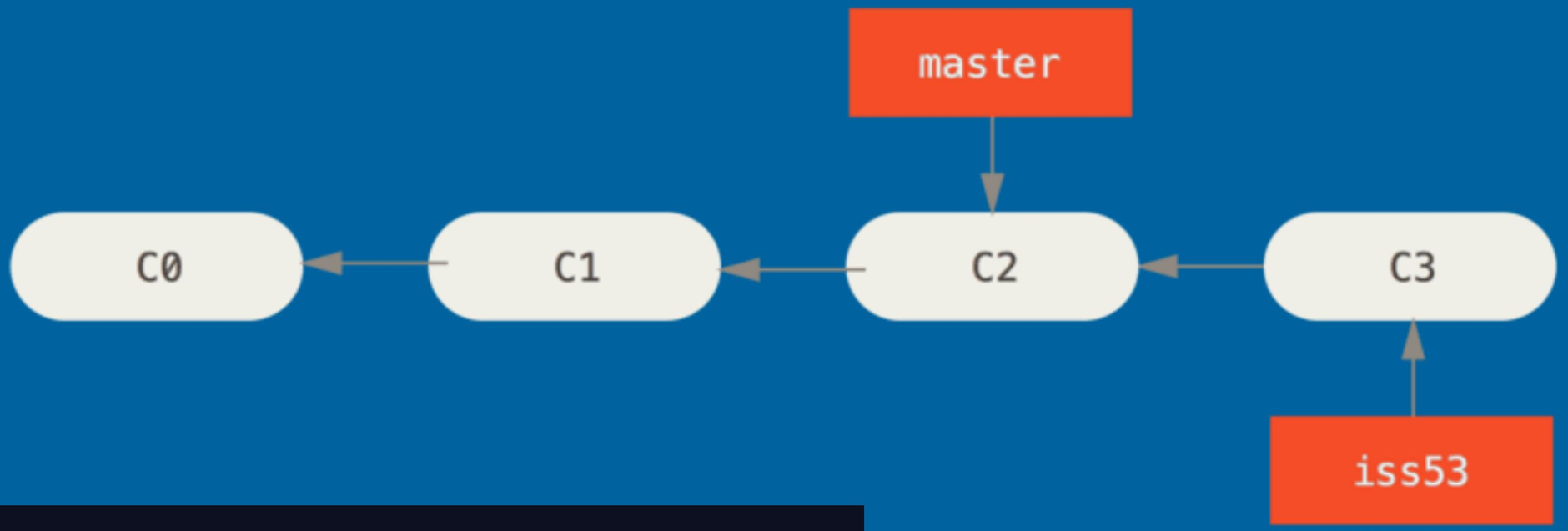
Git Basics - Branching




```
$ vim index.html
$ git commit -a -m 'added a new footer [issue 53]'
```

Source: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

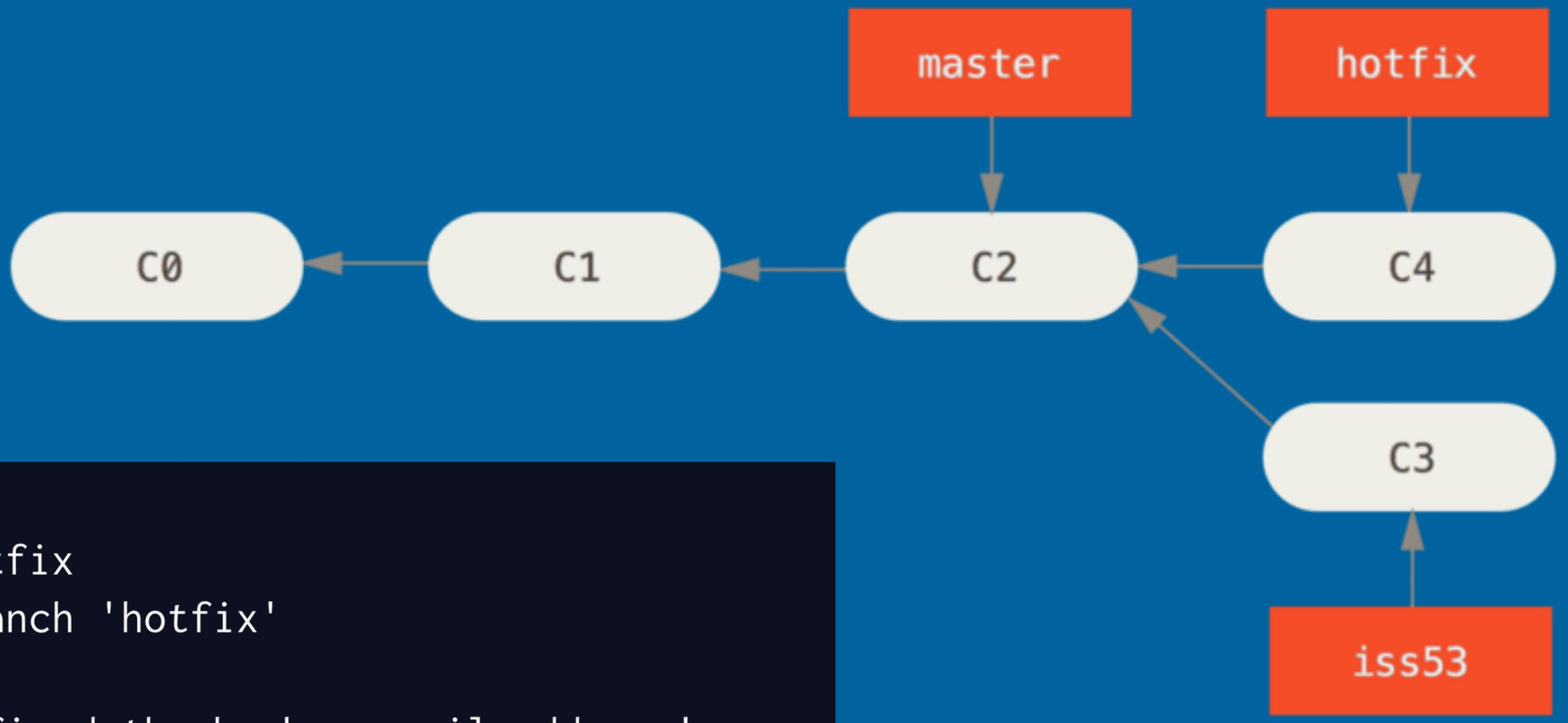
Git Basics - Branching




```
$ git checkout master
Switched to branch 'master'
```

Source: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

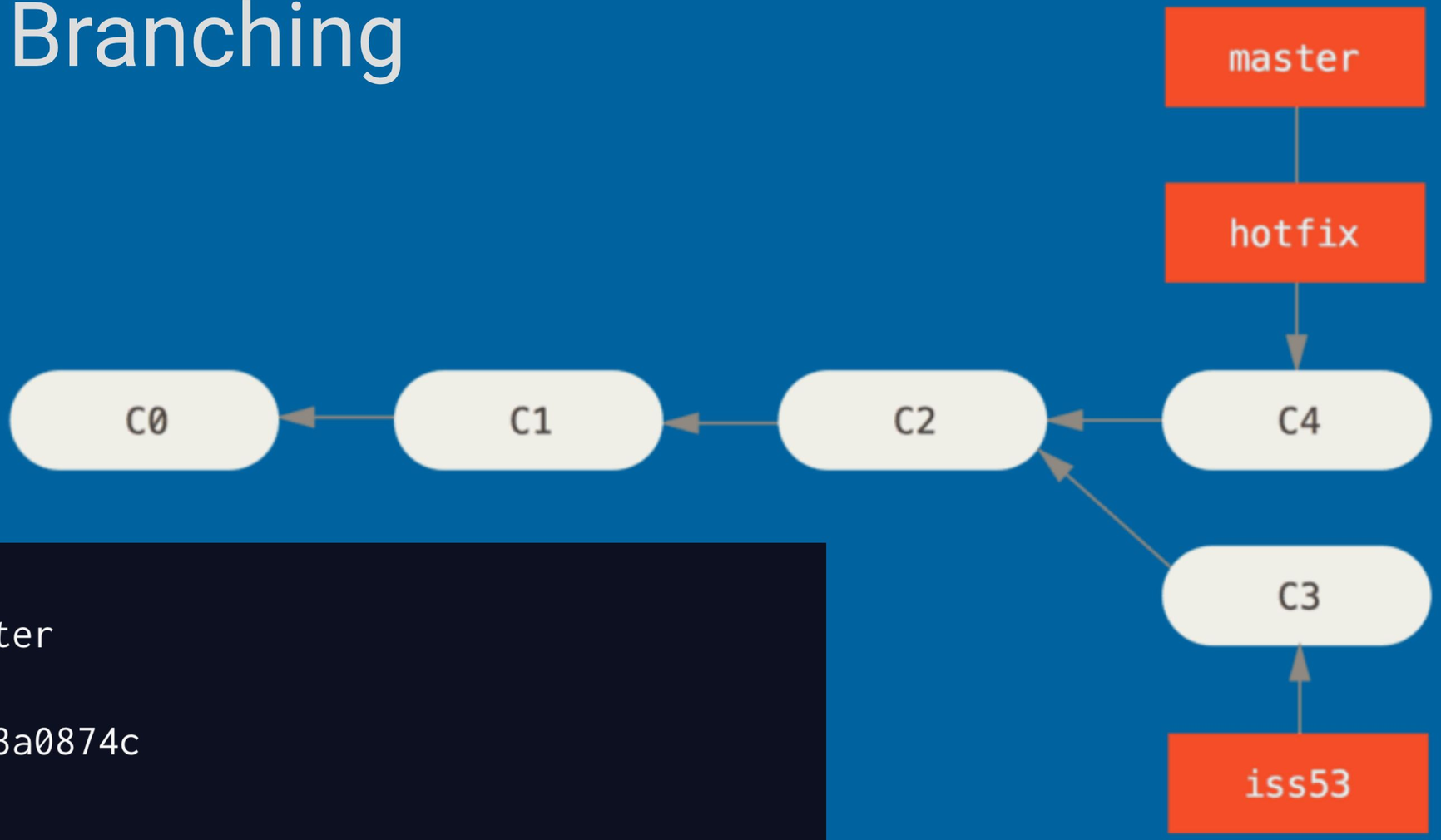
Git Basics - Branching




```
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
$ vim index.html
$ git commit -a -m 'fixed the broken email address'
[hotfix 1fb7853] fixed the broken email address
 1 file changed, 2 insertions(+)
```

Source: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

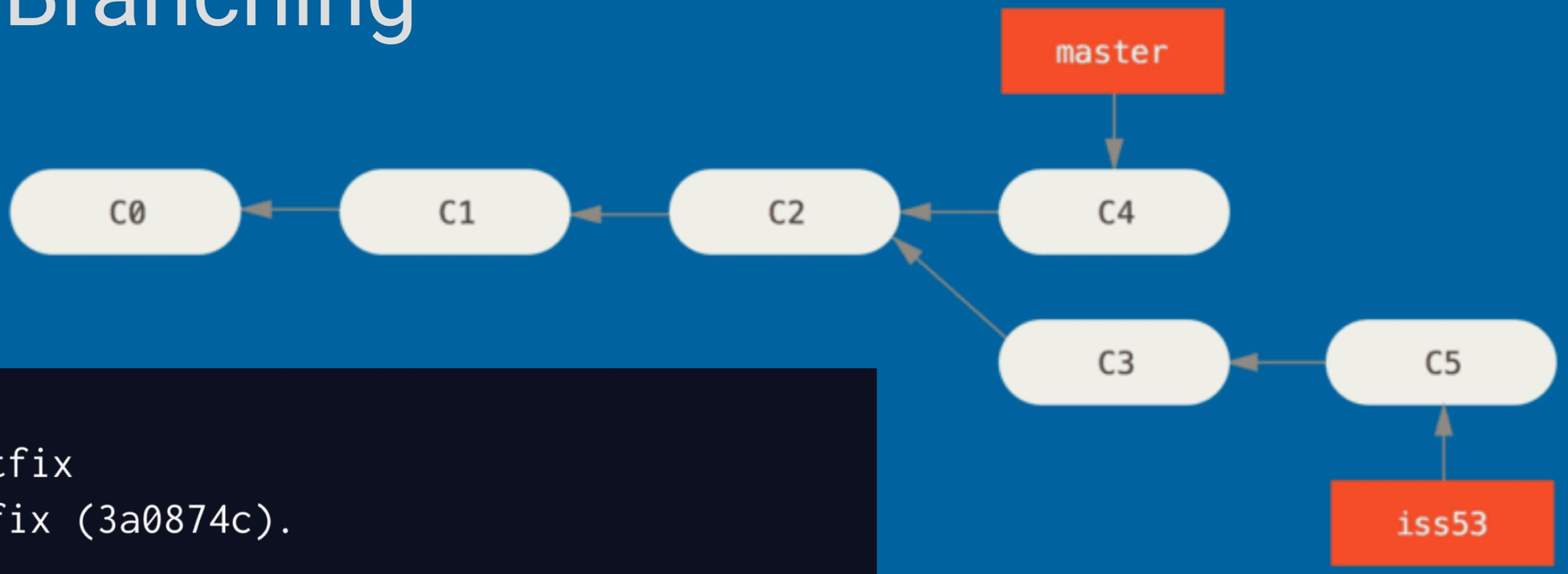
Git Basics - Branching




```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```

Source: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Git Basics - Branching

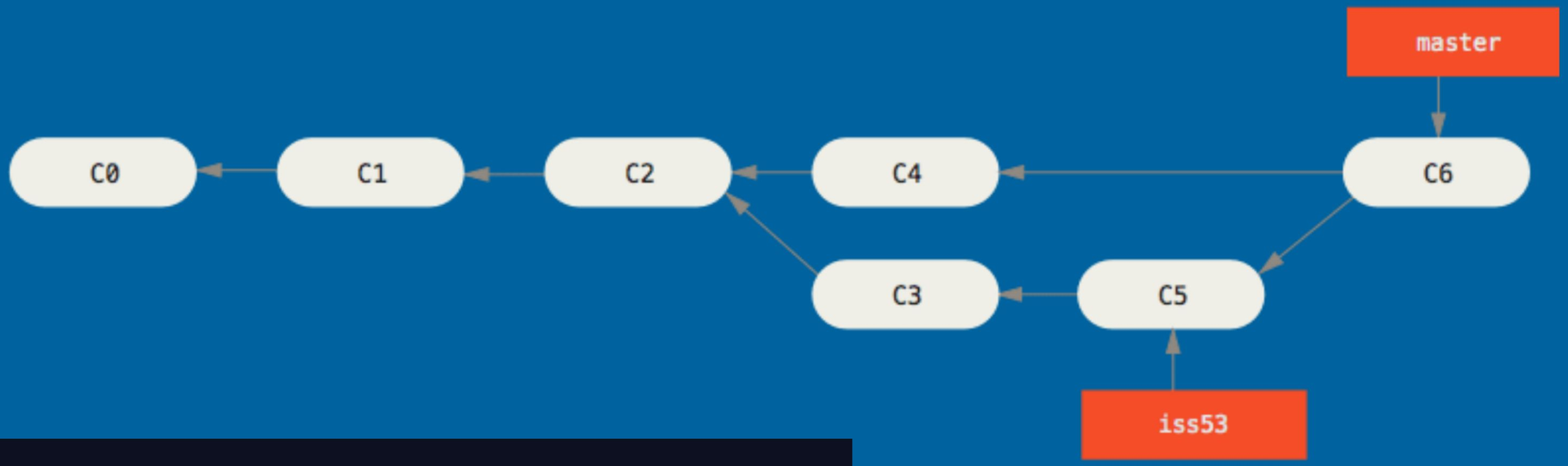



```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).

$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'finished the new footer [issue 53]'
[iss53 ad82d7a] finished the new footer [issue 53]
1 file changed, 1 insertion(+)
```

Source: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Git Basics - Branching




```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html |    1 +
1 file changed, 1 insertion(+)
```

Source: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Git Basics - Summary

* Empfehlung: Lest (und versteht!) die ersten 3 Kapitel von https://git-scm.com/book/en/v2
* Git eignet sich nicht nur für Quellcode, sondern für Dateien jeglicher Art.
* Eine super Idee, um die eigenen Änderungen an Projekt- und Bachelorarbeiten nachzuvollziehen (und ggf. ein Backup zu haben, wenn Word mal wieder eine Datei zerschießt!)
* Kostenlose private Repos gibts auf https://bitbucket.com

Wie arbeite ich mit git?

* Neben der technischen Bedienung ist Versionskontrolle vor allem eine **organisatorische Herausforderung**
* Man einigt sich auf ein Arbeitsmodell - z.B. https://nvie.com/posts/a-successful-git-branching-model/

Wie arbeite ich mit git?

* Beispiel aus dem Neos-Projekt:
* Feature-Entwicklung auf "feature branches"

  Werden nach aus "master" erzeugt, nach Abschluss wieder nach "master" gemerged und gelöscht
* Releases auf "release branches"

  Sind langlebig und erhalten Fixes für das jeweilige Release
* Bugfixes auf "bugfix branches"

  Werden aus dem "lowest supported release branch" erzeugt und werden nach Abschluss auf diesen gemerged. Danach erfolgen "Upmerges", d.h. sukzessive Merges der Release-Branches in "master", damit Bugfixes auch dort landen.

Coding Time!

Git - Codingaufgabe

30m

* Installiert git und die git-GUI eurer Wahl
* Legt ein Repository an
* Fügt Dateien hinzu und fügt sie in die Staging Area ein
* Macht einige Commits
* Initialisiert ein Repository auf GitHub und fügt es als remote hinzu
* Pusht eure Änderungen dorthin
* Legt einige Branches an
* Macht Commits auf den Branches und merged sie in "master"