GIT

Fast version control

Gregor Schwab |
gregor.schwab@voestalpine.com

Git presentation @ Voestalpine | 30.11.2017

ZU MEINER PERSON

- Studium an der Universität Innsbruck
- 1994+ versch. Softwareentwicklung mit C++, Java (JBoss) und Perl (Philips)
- 1994+ Erstkontakt mit Unix Workstations und RCS
- 2004+ techn. Infrastruktur Standortleitung Sysadmins
- 2011+ Leitung der Webentwicklung (EA Startup Turbulenz)

- 2013+ Teamleitung Virtualisierungsinfrastruktur am ZID (Universität Innsbruck) für 4000 Mitarbeiter und ca. 40.000 Studierende Planung, Betreuung und Weiterentwicklung des Gitlab Community Edition Servers der Universität Innsbruck (800 Benutzer über 1000 Projekte) Entwicklung der Puppet Deployment Infrastruktur mit Tests/CI/CD
- 2015+ SCRUM Master Ausbildung
- 2015+ Projektmanagement div. DevOPS Projekte

VERSIONSKONTROLLSYSTEME

- einfache Versionskontrolle
  - final, final2, really_final, ...
  - v1, v1.2.5,....
  - RCS, lokal
- Client - Server
  - CVS, SVN
- Distributed
  - GIT, Mercurial, Bazaar

KURZE ENTSTEHUNGSGESCHICHTE VON GIT

- 2006 von Linus Torvalds entwickelt
- Schwerpunkt Softwareentwicklung (Linux Kernel)
- Unzufriedenheit mit bestehenden Systemen

VORTEILE VON GIT

- schneller
- kein Server nötig (Committen, Mergen, branchen wann immer man will)
- einfachere Kollaboration durch Pull Requests, Forks, Soziale Netzwerke (Github,...)

DVCS VS. VCS

- Snapshots (keine Deltas)
- Lightweight Forks, wie DNA
- Keine Network Latency
- Sauberes Merge Tracking
- Datenintegrität (SHA1 Summen)
- Meritocracy anstelle Comitter Status
- Lokale Branches (ermöglicht privates Arbeiten und neue Ideen)

INSTALLATION VON GIT

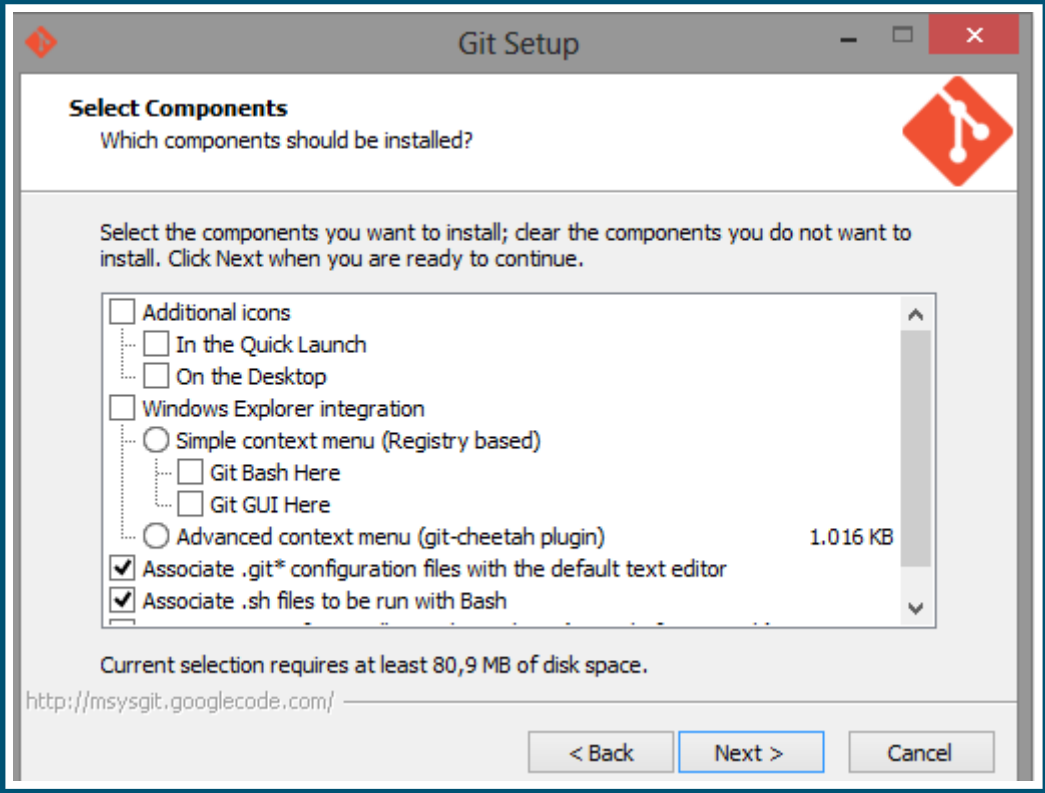
https://git-scm.com/downloads

- Unter Windows: https://git-scm.com/download/win
- Unter Linux (z.B. über Packagemanager):

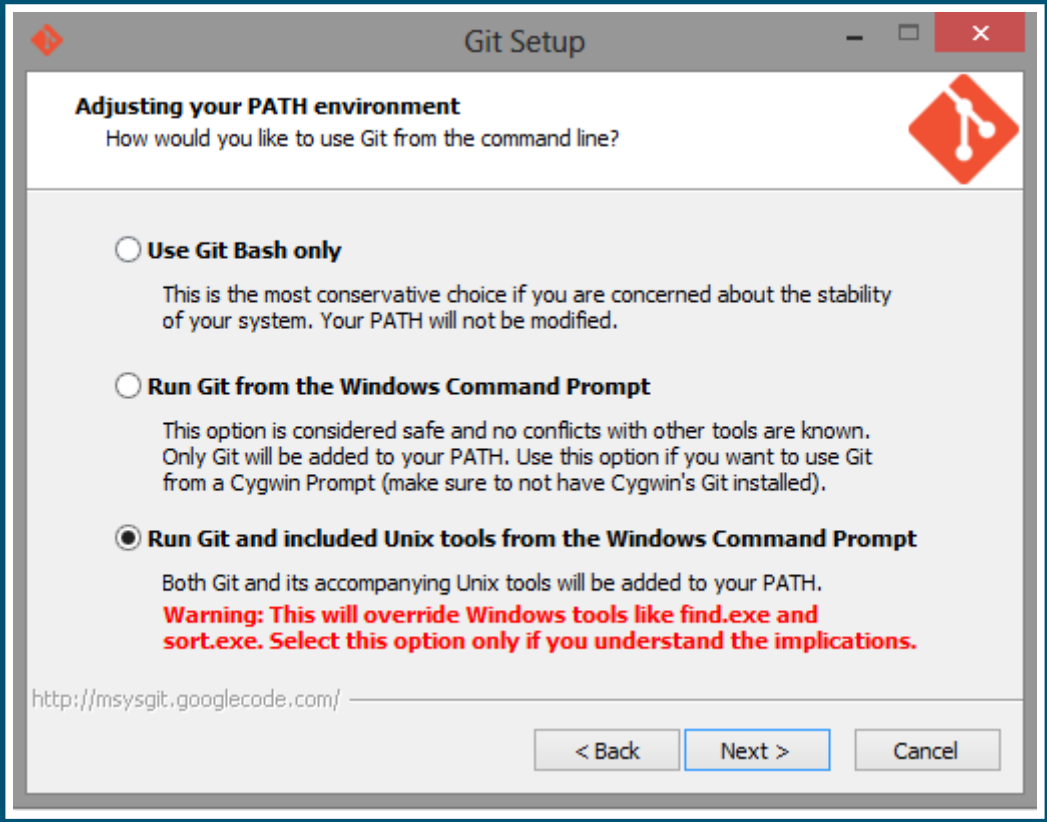
```
# Installation von git auf Ubuntu
$ apt-get install git
```

- Selbst kompilieren (nicht empfohlen)

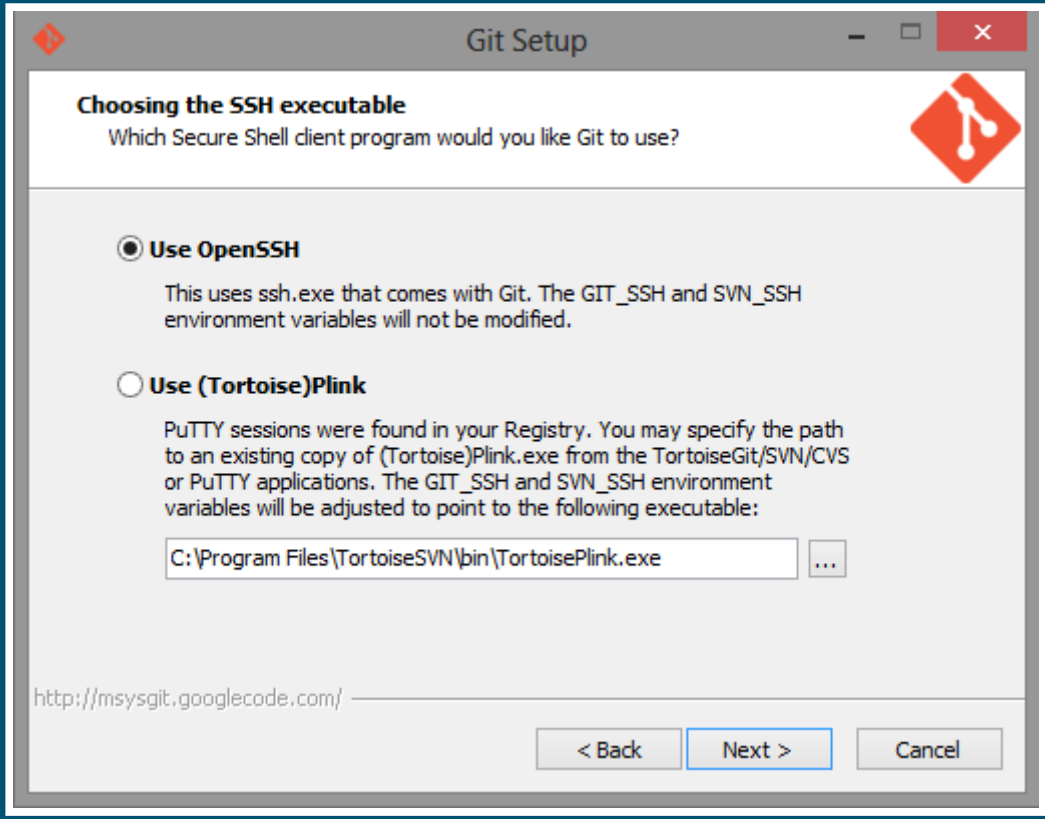
INSTALLATION UNTER WINDOWS IM DETAIL



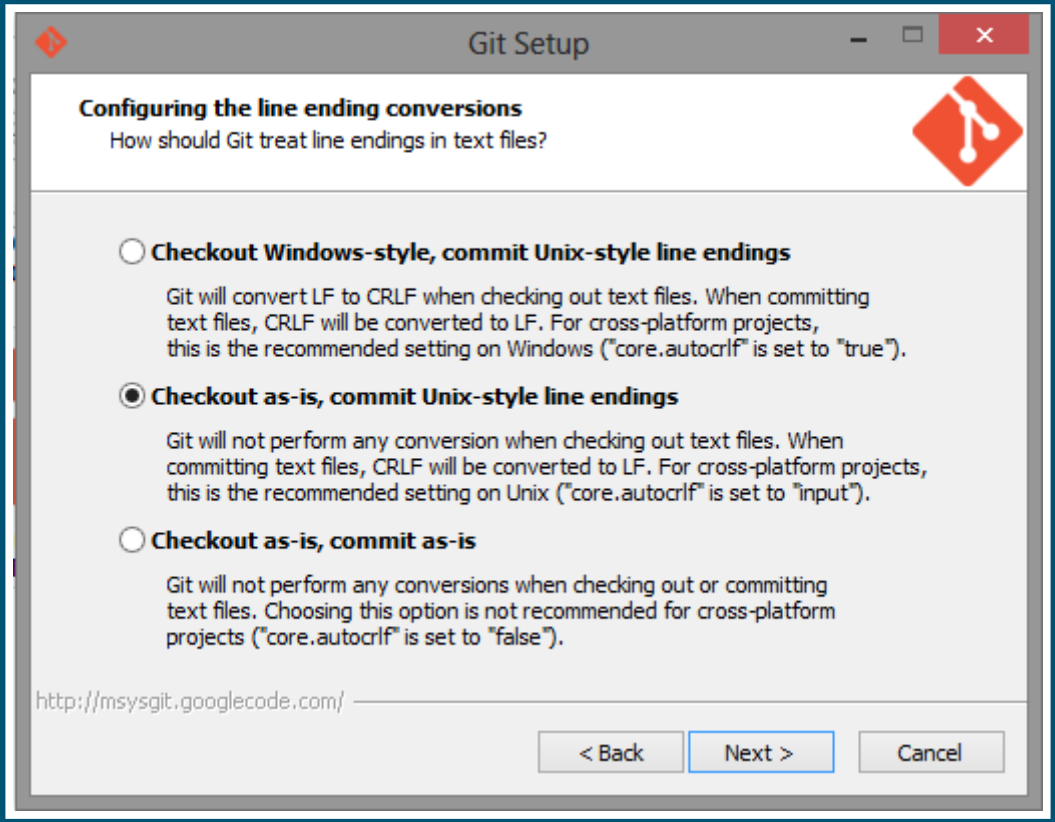
INSTALLATION UNTER WINDOWS IM DETAIL



INSTALLATION UNTER WINDOWS IM DETAIL

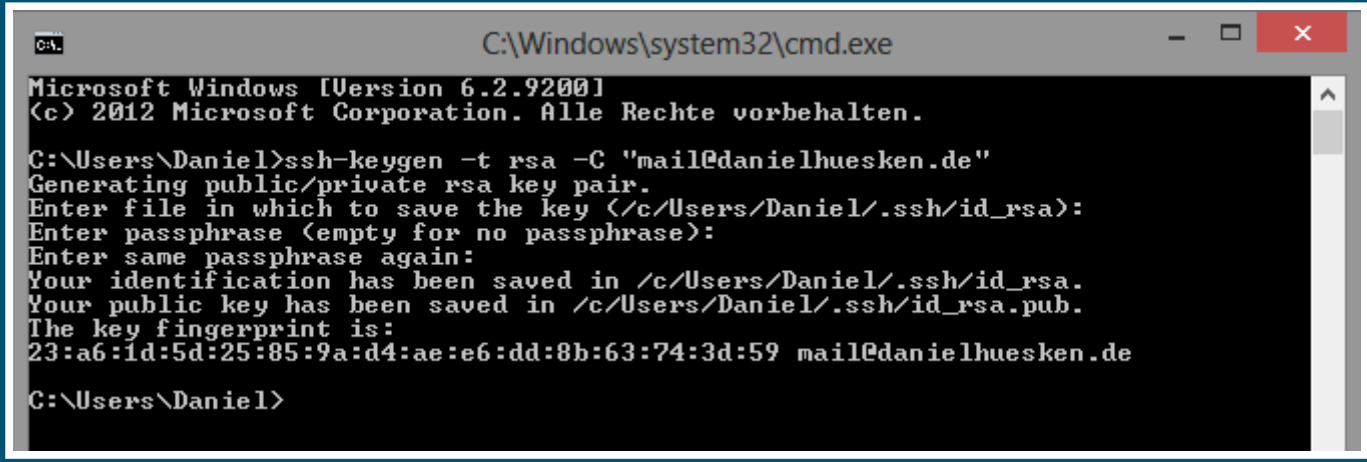


INSTALLATION UNTER WINDOWS IM DETAIL



INSTALLATION DER SSH KEYS (OPTIONAL)

```
ssh-keygen -t rsa -C "your_email@example.com"
```



GIT SETUP

Erstmalig:

```
# Benutzerparameter für GIT
  git config --global user.name "Your Name"
  git config --global user.email "your_email@whatever.com"
```

- /etc/gitconfig (global)
- ~/.gitconfig oder %USERPROFILE%/.gitconfig (benutzerspezifisch)
- PROJEKTORDNER/.git/config (projektspezifisch)

```
# GIT Einstellungen abfragen
git config --list
```

DAS GIT REPOSITORY

- Das .git Verzeichnis
- Die Working Area (checkout)
- Die Staging Area

EVERYDAY GIT

```
git init
```

GIT STATUS

```
#Status abfragen
git status
```

```
git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committe

    hello.js

nothing added to commit but untracked files present (use "git
</file>
```

DATEIEN IN DER STAGING AREA IGNORIEREN

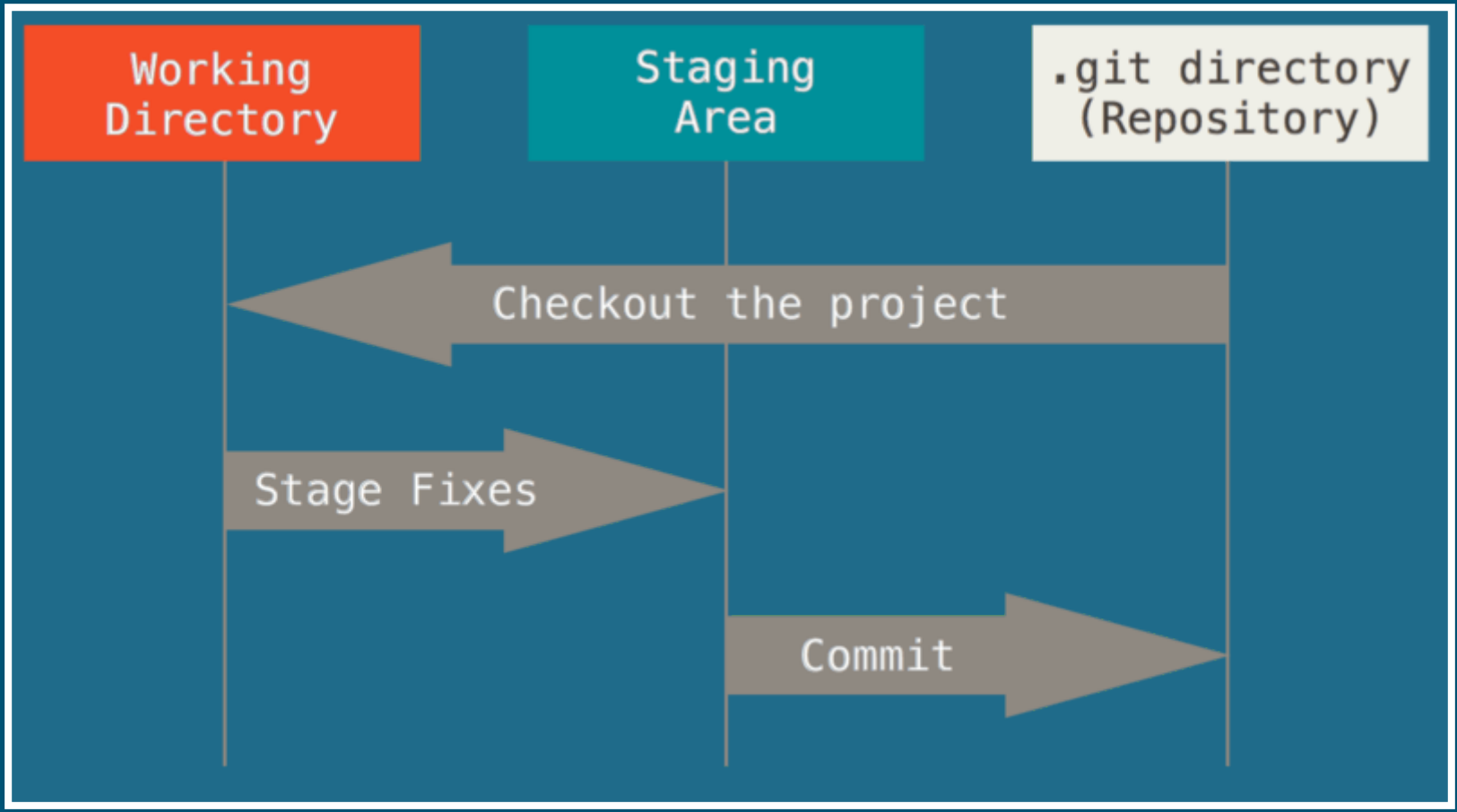
.gitignore

GIT ADD (STAGING)

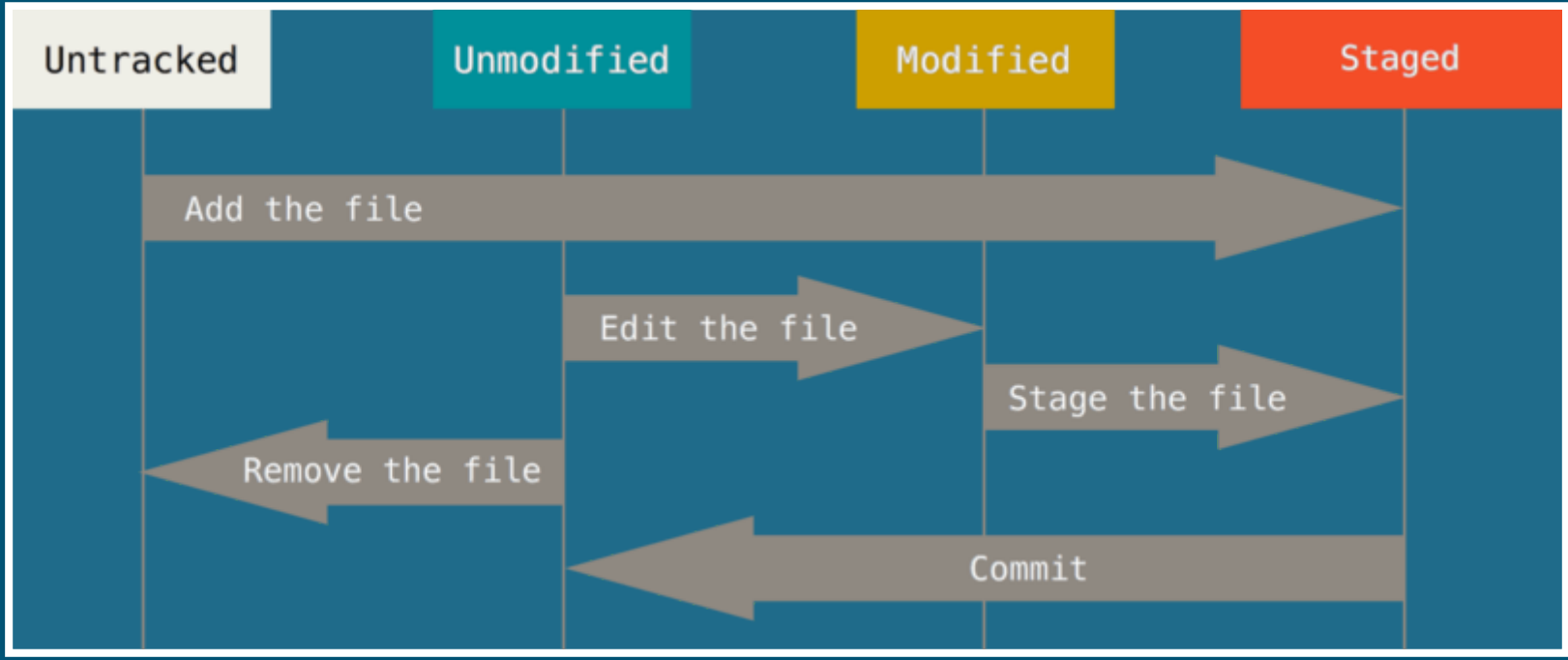
```
git add file
git add \*.c
git add .
```

```
 git add hello.js
gregor@ip-172-31-16-42:~/test$ git status
On branch master

Initial commit

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

    new file:   hello.js
</file>
```

DIE STAGING AREA (DER INDEX)



GIT WORKFLOW



GIT COMMIT

```
git commit -m "Commit Message"
git commit -m "Inital Commit"
```

```
[master d4b7ff2] Inital Commit
 1 file changed, 1 insertion(+), 1 deletion(-)
```

DEN LETZTEN COMMIT RÜCKGÄNGIG MACHEN

```
git commit --amend
```

```
git commit --amend
Inital Commit

# Please enter the commit message for your changes. Lines star
# with '#' will be ignored, and an empty message aborts the co
#
# Author:    Gregor <gregor@ip-172-31-16-42.ap-southeast-1.com
# Date:      Wed Nov 29 22:30:55 2017 +0000
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   hello.js
```

DIE WORKING AREA UND STAGING AREA ZURÜCKSETZEN

```
git reset --hard
#nur das Ändern einer Datei in der Working Area rückgängig mac
git checkout -- CONTRIBUTING.md
```

NUR DIE STAGING AREA ZURÜCKSETZEN (UNSTAGE)

```
git reset
git reset DATEINAME
```

DATEIEN LÖSCHEN

```
#Datei entfernen
rm hello.js
#Die Datei aus der Staging Area löschen
git rm hello.js
git rm --cached hello.js #Datei behalten
```

```
git rm hello.js
HEAD detached at v1
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    hello.js
</file>
```

DATEIEN UMBENENNEN

```
#Datei umbennenen
git mv hello.js hallo.js

mv hello.js hallo.js
git add hallo.js
git rm hello.js
```

```
git status
HEAD detached at v1
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed
  (use "git checkout -- <file>..." to discard changes in worki

    deleted:    hello.js

Untracked files:
  (use "git add <file>..." to include in what will be committe

    hallo.js

no changes added to commit (use "git add" and/or "git commit -
</file></file></file>
```

GIT DIFF

```
#Die Differenz der Working Area zum letzten Commit
git diff
#Die Differenz der Staging Area zum letzten Commit
git diff --staged
```

```
git diff
diff --git a/hello.js b/hello.js
index 8a00f7c..481e5ca 100644
--- a/hello.js
+++ b/hello.js
@@ -1 +1 @@
-console.log ("Hello World!")
+console.log ("Hallo Welt!")
```

TAGGING

```
#dem momentanen HEAD den TAG v1 geben
git tag v1
git tag v1 CommitNo
git checkout v1^
git tag v1-beta
```

```
#Tags ansehen
git tag
```

HISTORY

```
#die letzten Commits anzeigen
git log
git log --since=yesterday
#Übersichtlichere Darstellung
git log --pretty=format:"%h %ad | %s%d [%an]"
--graph --date=short
```

```
#ein Alias in .gitconfig definieren
[alias]
  hist = log --pretty=format:\"%h %ad | %s%d [%an]\" --graph -
```

```
git hist
d4b7ff2 Wed Nov 29 23:24:14 2017 +0000 | Inital Commit (HEAD -
6d3c67a Wed Nov 29 22:30:55 2017 +0000 | Second Commit [Gregor
```

BRANCHES

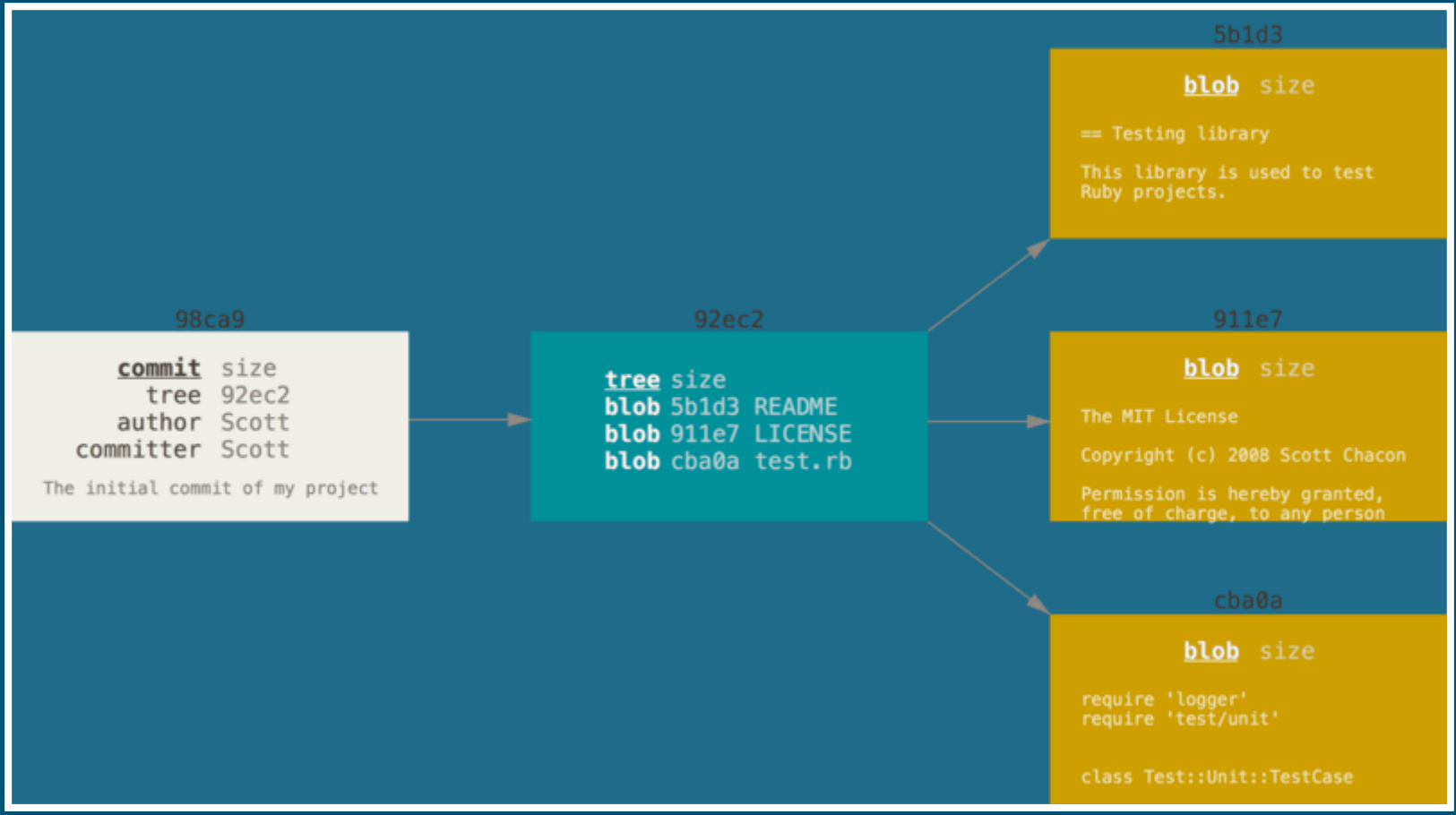
```
#einen neuen lokalen Branch anlegen
git branch name
git branch
git branch -a #zeigt auch remote branches
```

```
git branch
  feature
* master
```

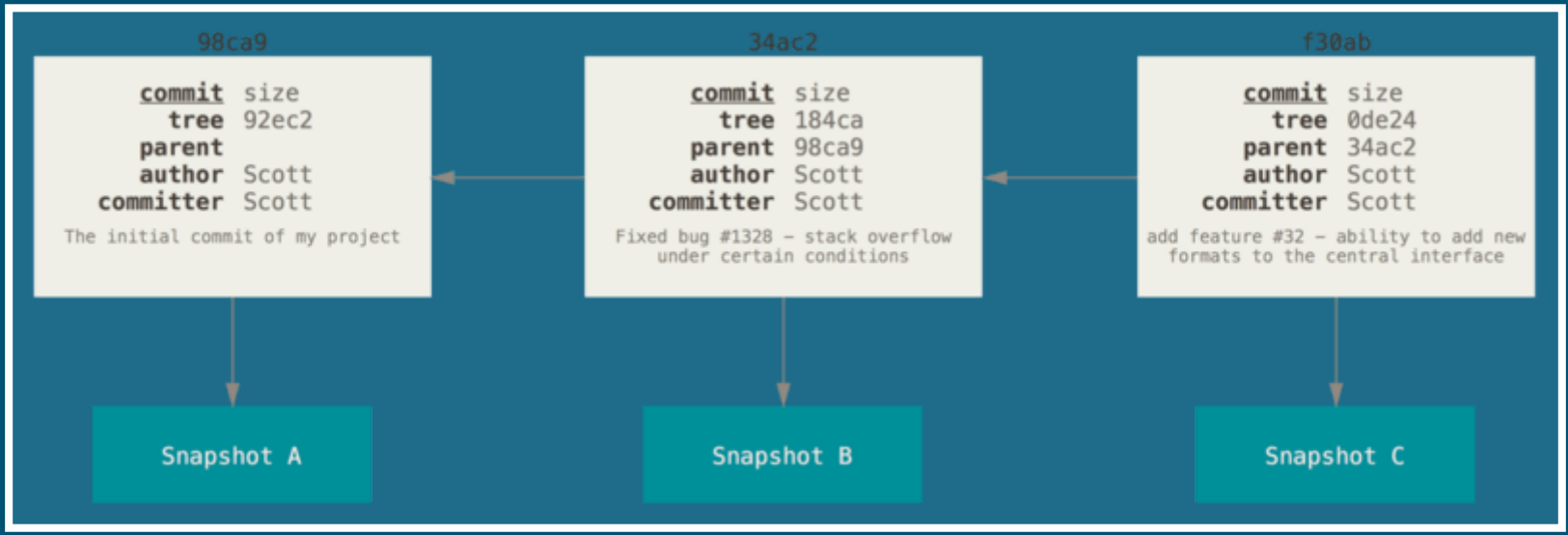
```
#einen neuen lokalen Branch anlegen
git checkout branch #(-b) legt neuen branch an
```

```
git checkout feature
Switched to branch 'feature'
```

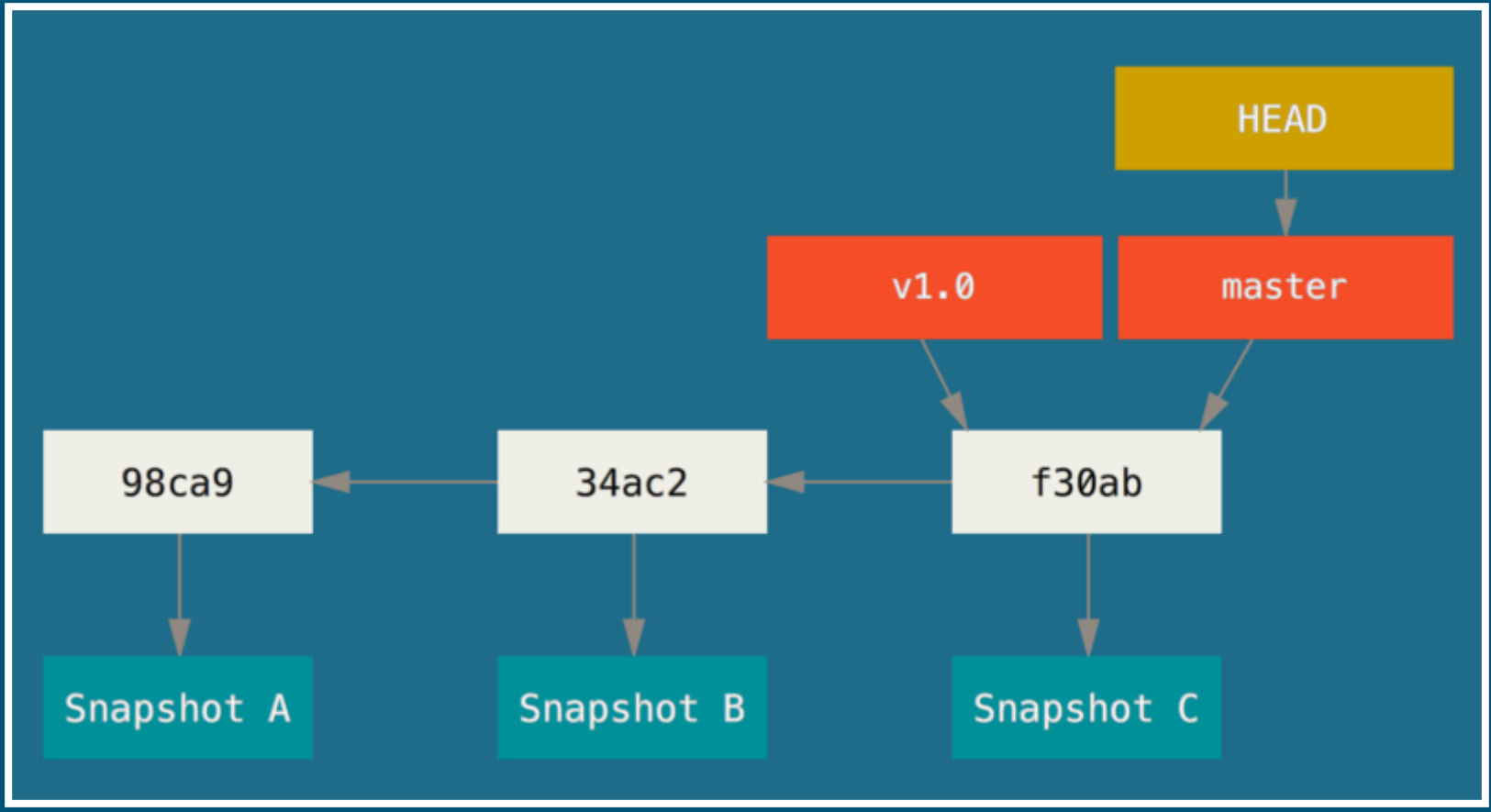
GIT INTERNALS



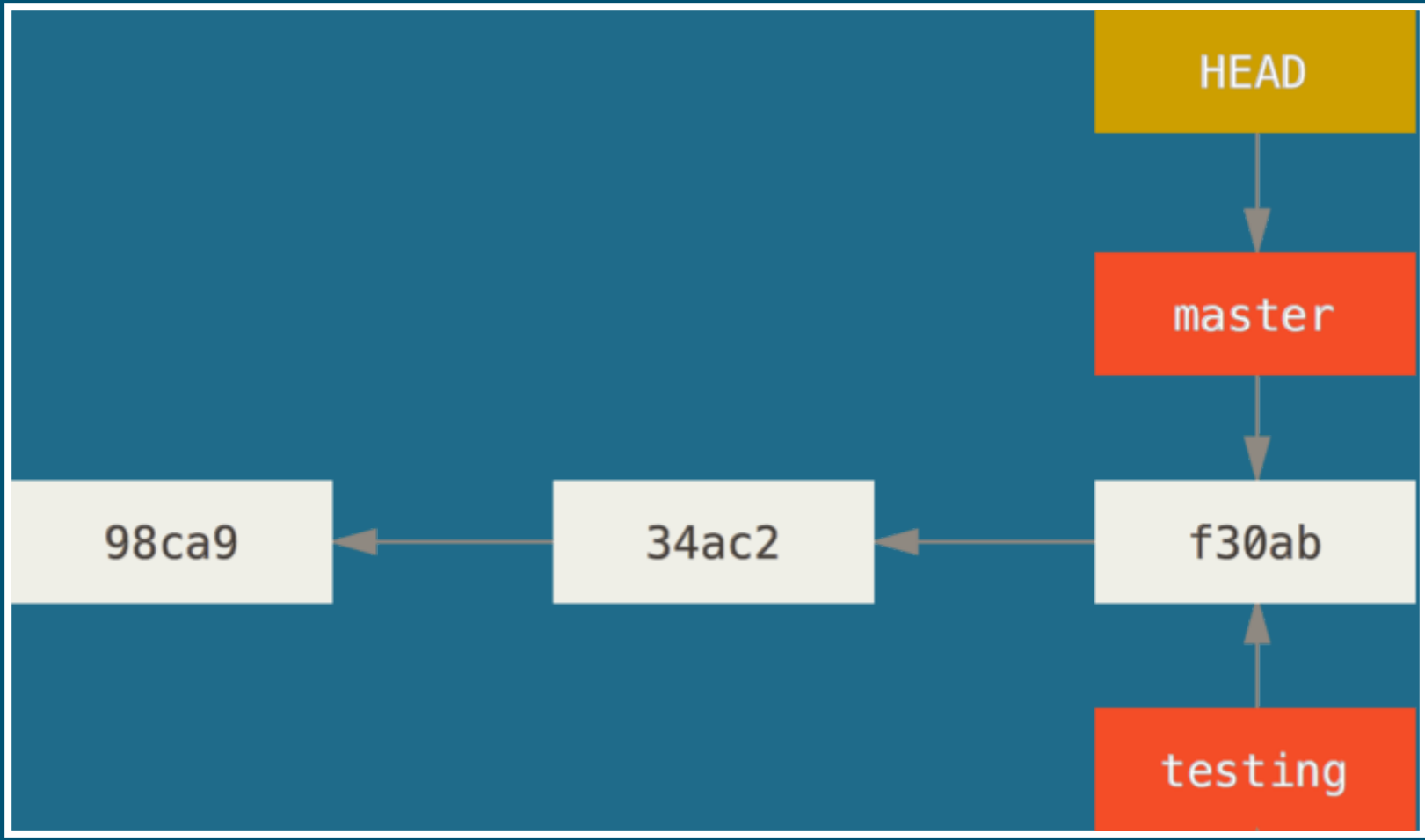
GIT INTERNALS



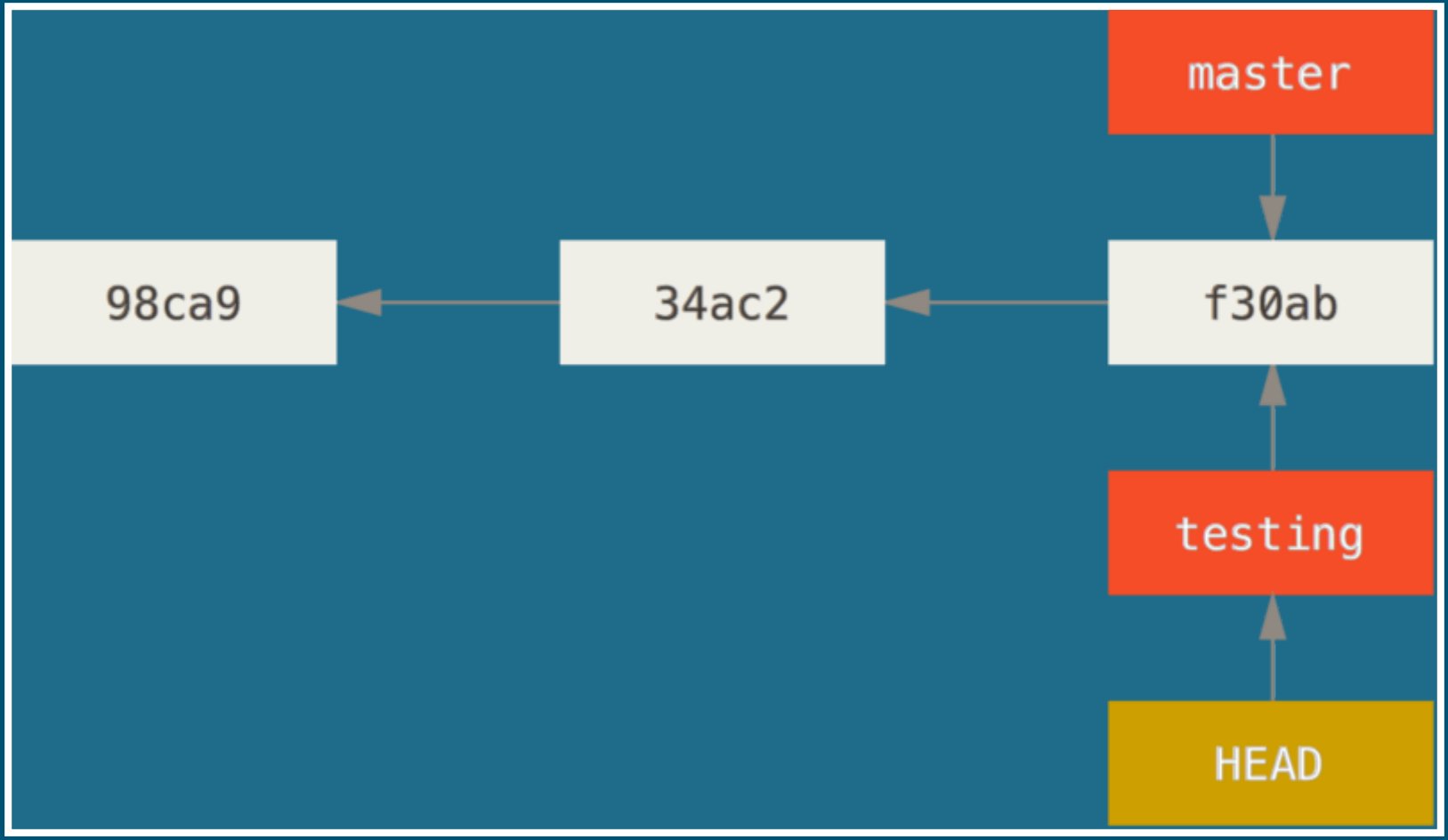
DER MASTER BRANCH



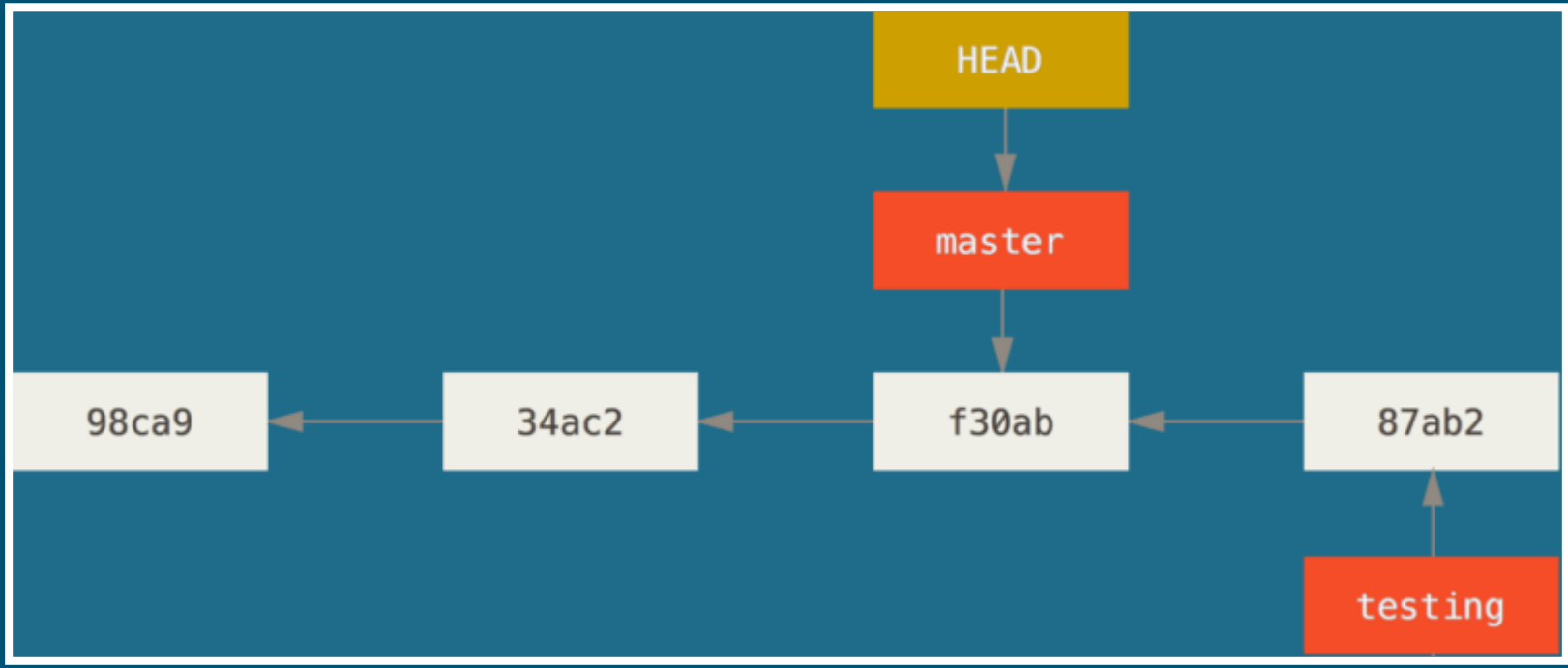
CREATE NEW BRANCH



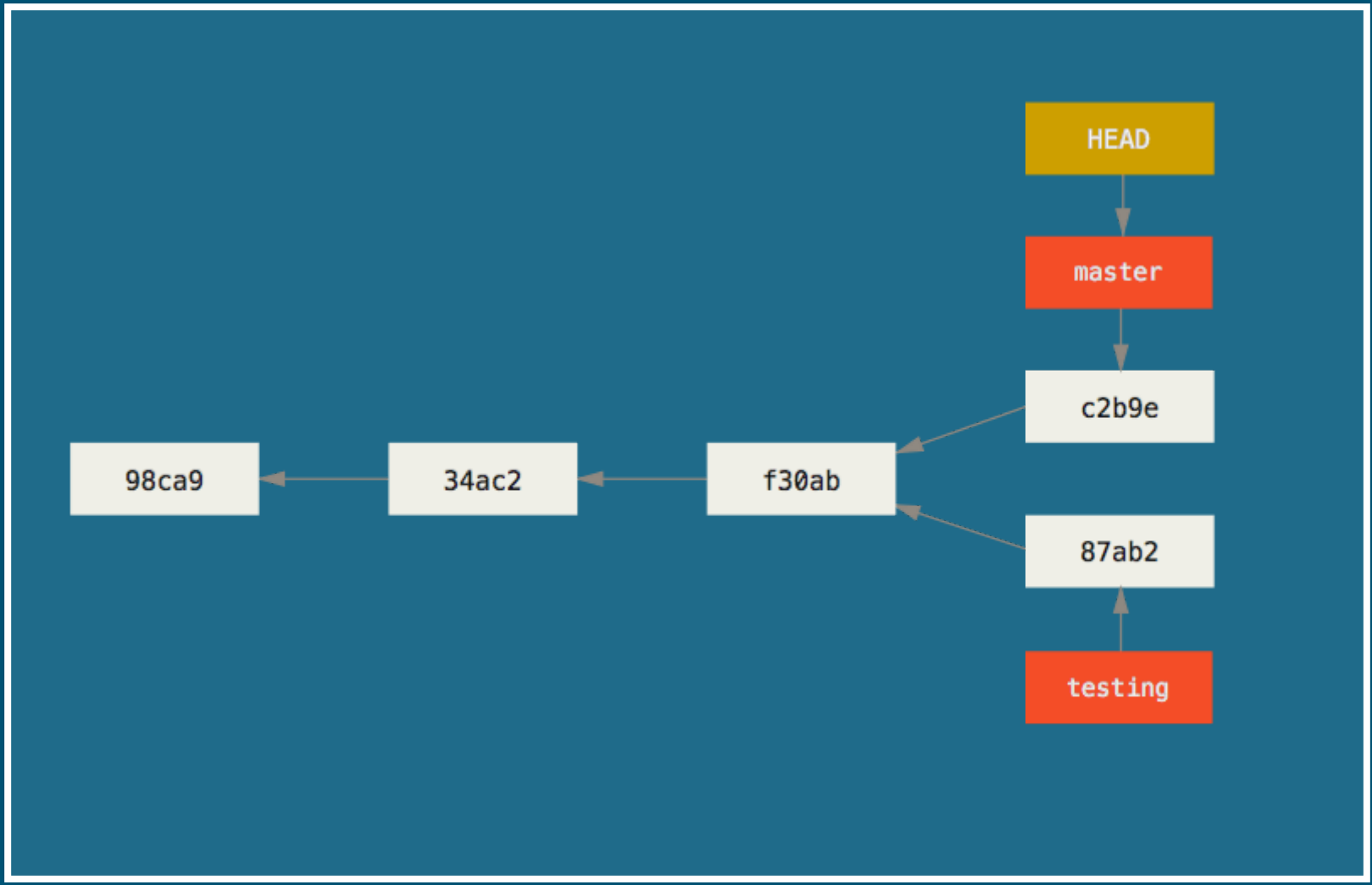
SWITCH TO NEW BRANCH



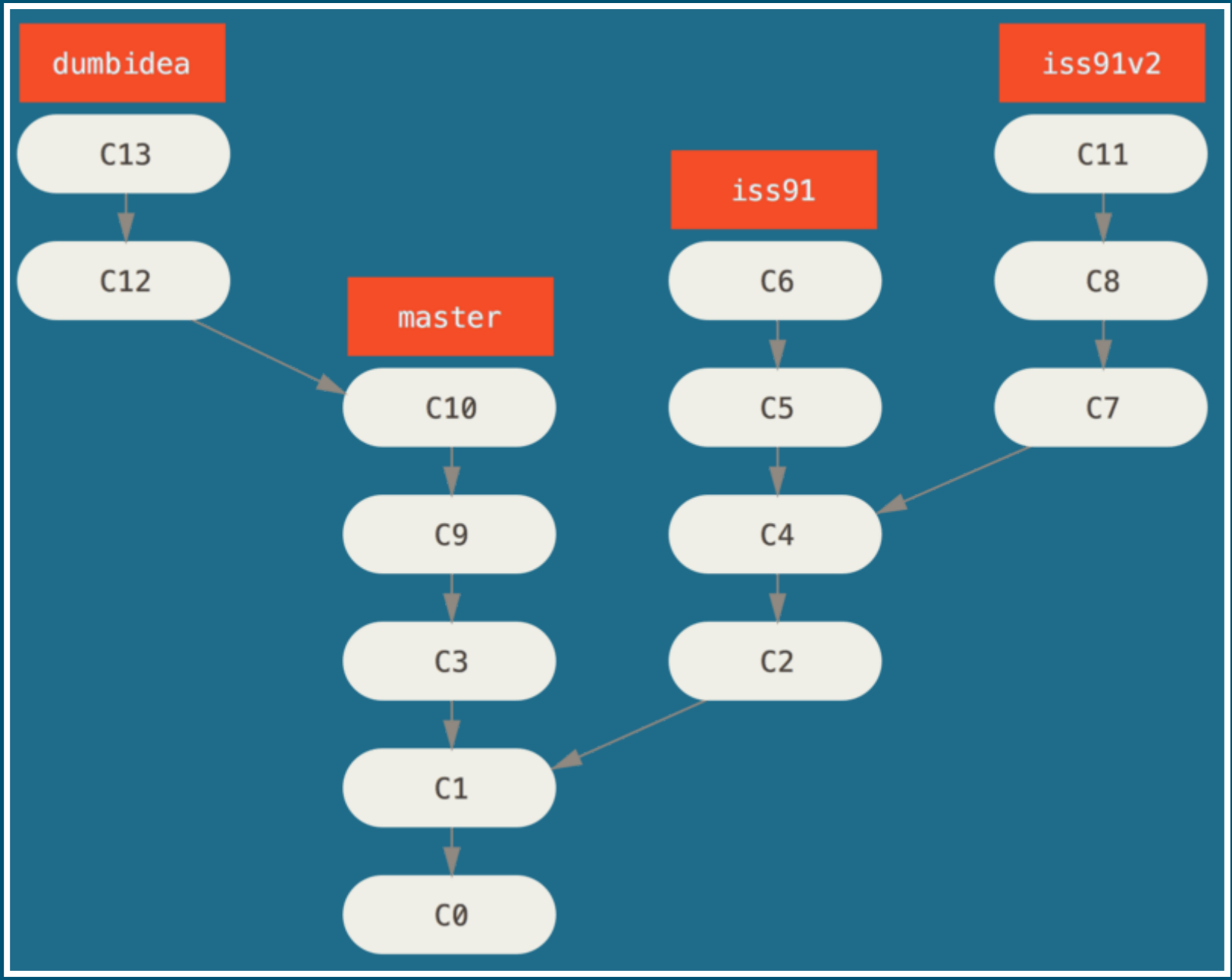
CHECKOUT MASTER AGAIN



ADVANCE MASTER



BRANCHES

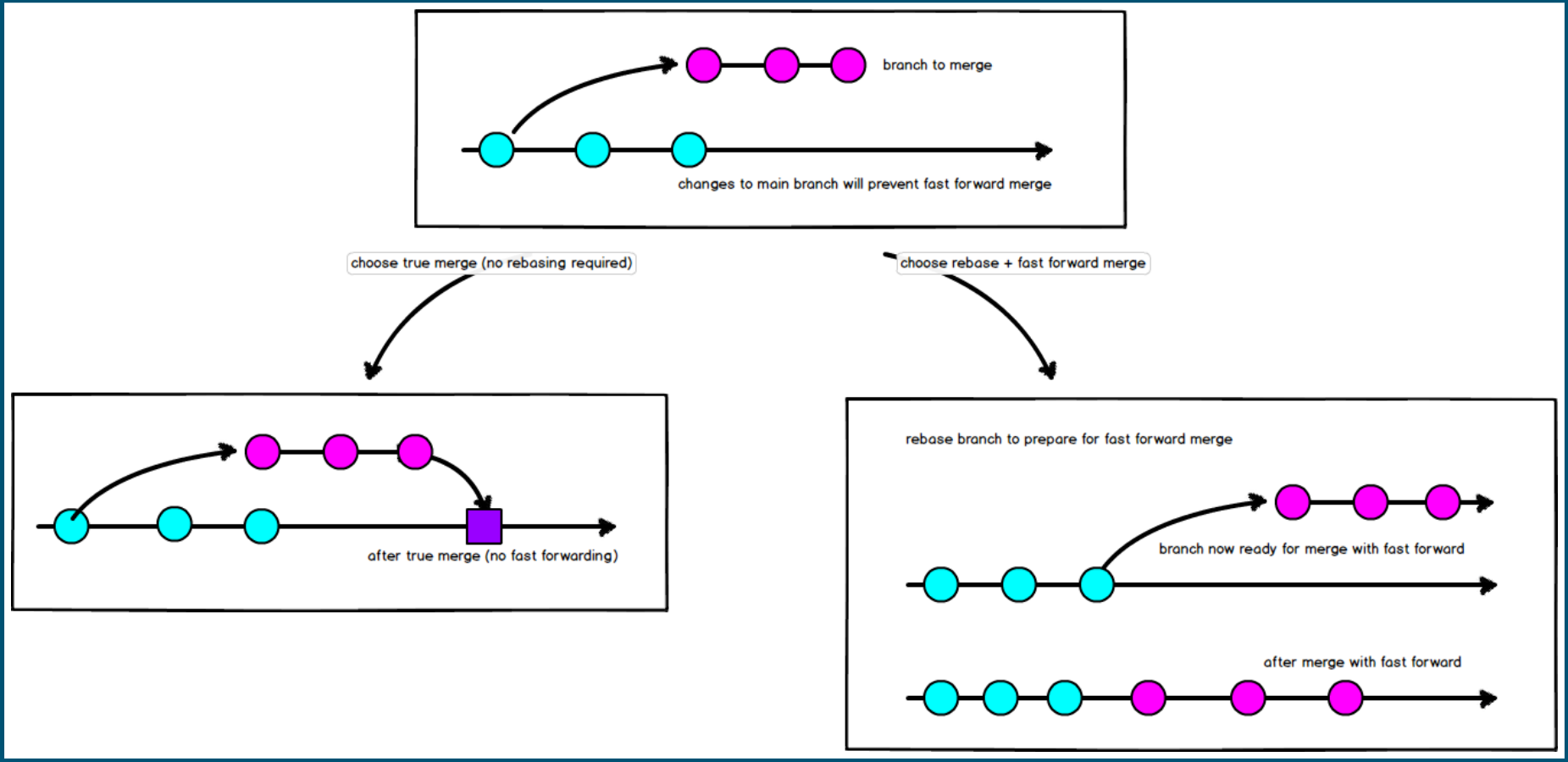

dumbidea
iss91v2
C13
C11
iss91
C12
C6
C8
master
C10
C5
C7
C9
C4
C3
C2
C1
C0

MERGING

```
#einen Branch in einen anderen mergen
git merge branch1 branch2
git merge branch #den branch in den momentanen checkout mergen
```

```
git branch
  feature
* master
```

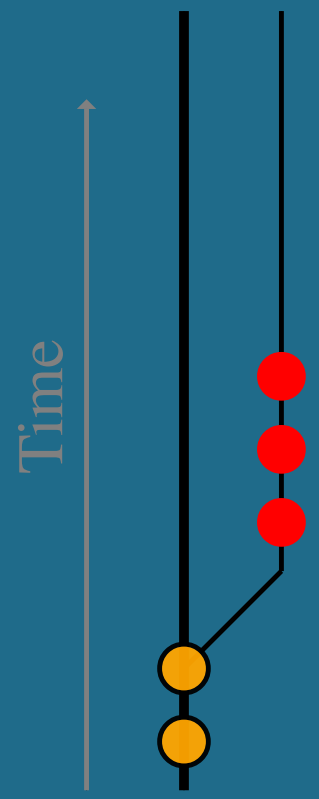
MERGE TYPES


branch to merge
changes to main branch will prevent fast forward merge
choose true merge (no rebasing required)
choose rebase + fast forward merge
after true merge (no fast forwarding)
rebase branch to prepare for fast forward merge
branch now ready for merge with fast forward
after merge with fast forward

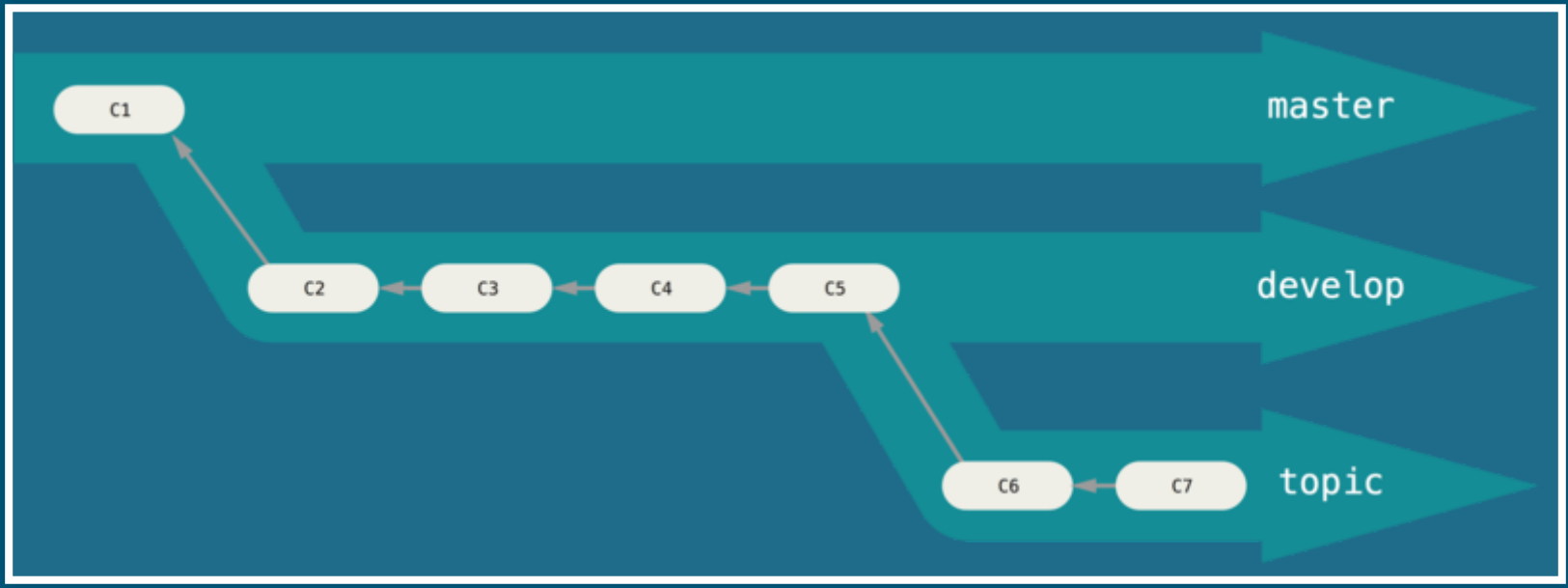
REBASE VS. MERGE

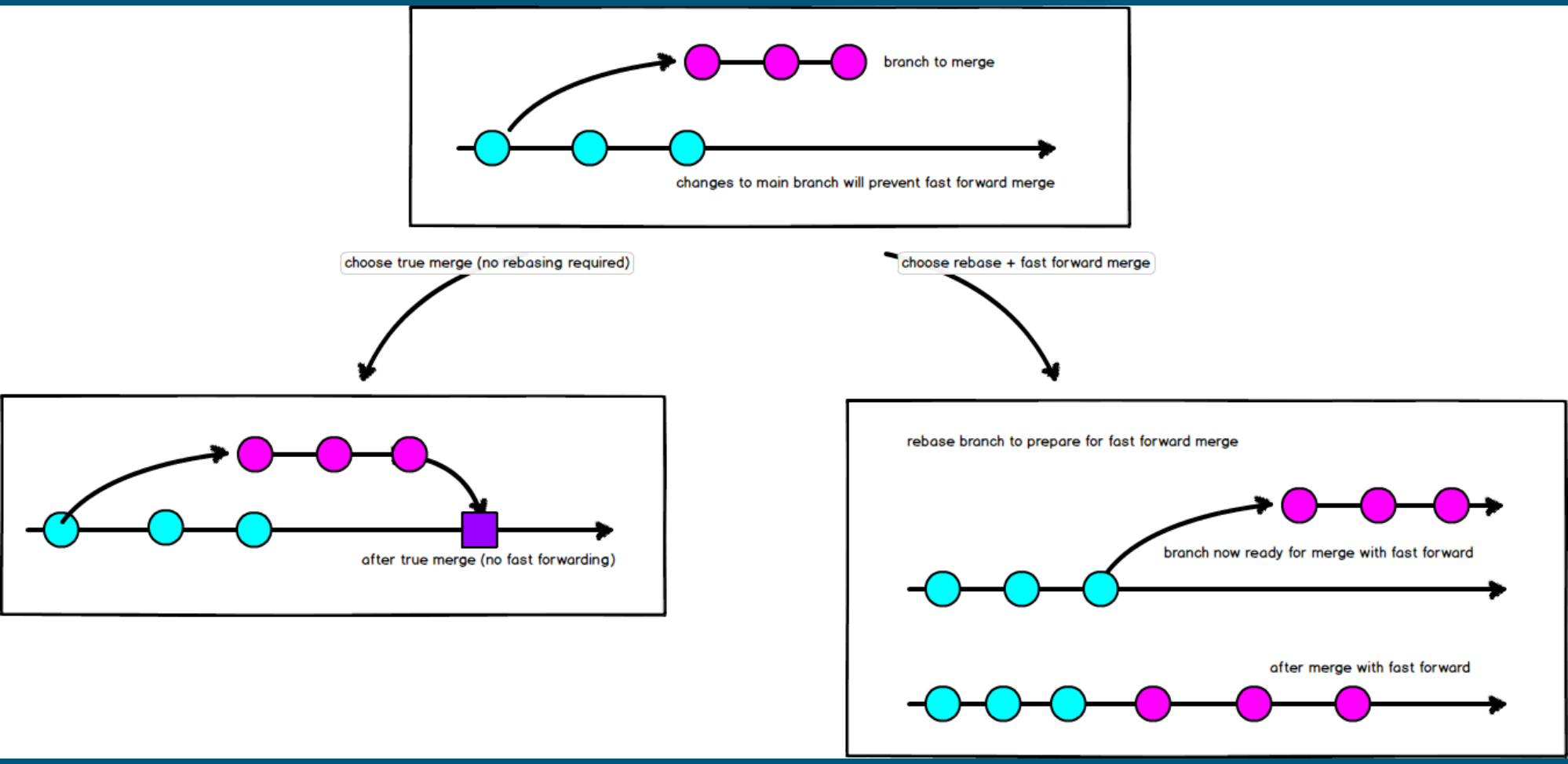
Rebasing:

Forward-port local commits to the updated upstream head.



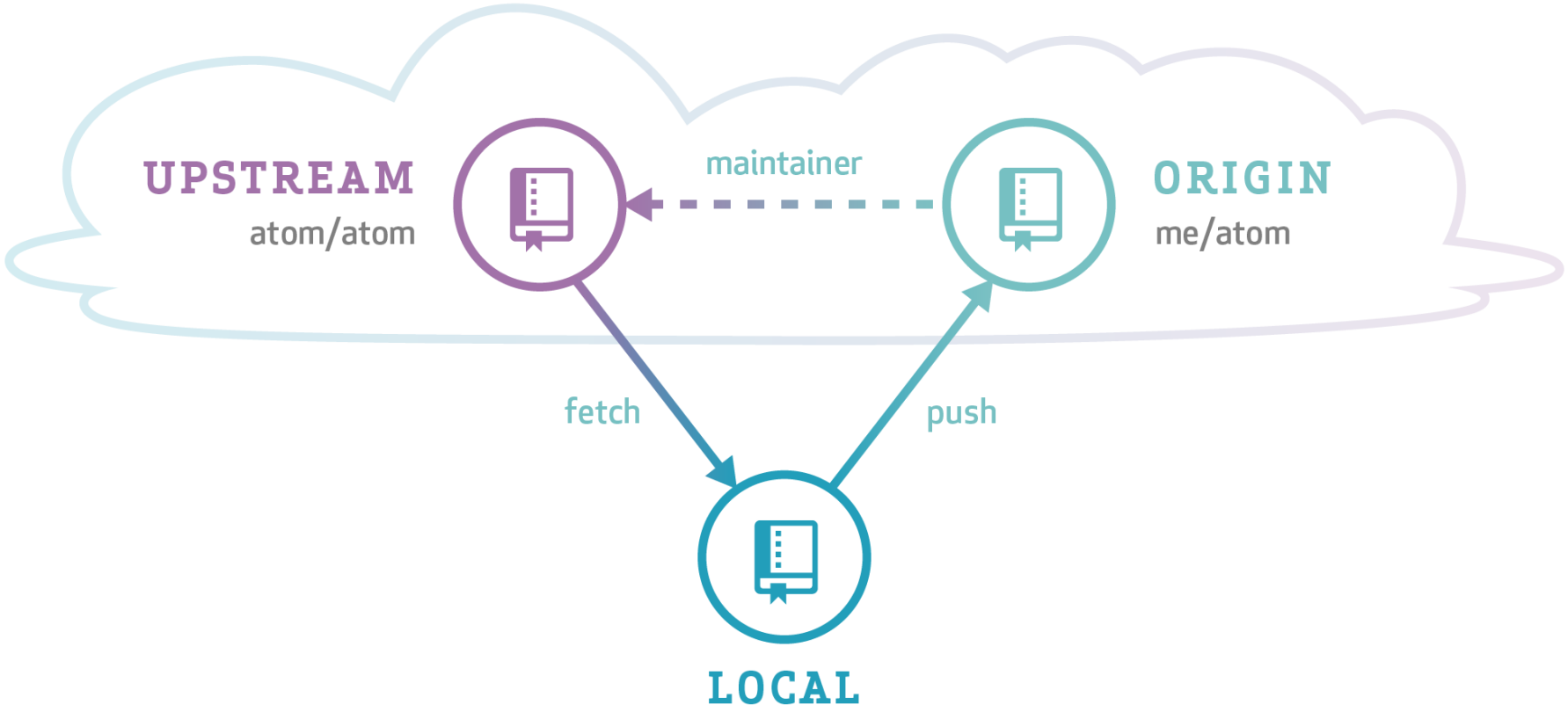
GIT WORKFLOWS




branch to merge
changes to main branch will prevent fast forward merge
choose true merge (no rebasing required)
choose rebase + fast forward merge
after true merge (no fast forwarding)
rebase branch to prepare for fast forward merge
branch now ready for merge with fast forward
after merge with fast forward

DEMO AREA

GIT REMOTES


UPSTREAM
atom/atom
maintainer
ORIGIN
me/atom
fetch
push
LOCAL

REMOTES ANZEIGEN

```
#Liste der remotes
git remote -v
```

```
origin    git@github.com-gregorjs:gregorjs/git.git (fetch)
origin    git@github.com-gregorjs:gregorjs/git.git (push)
```

REMOTE HINZUFÜGEN

```
#Remote hinzufügen
git remote add remote_name remote_adresse
```

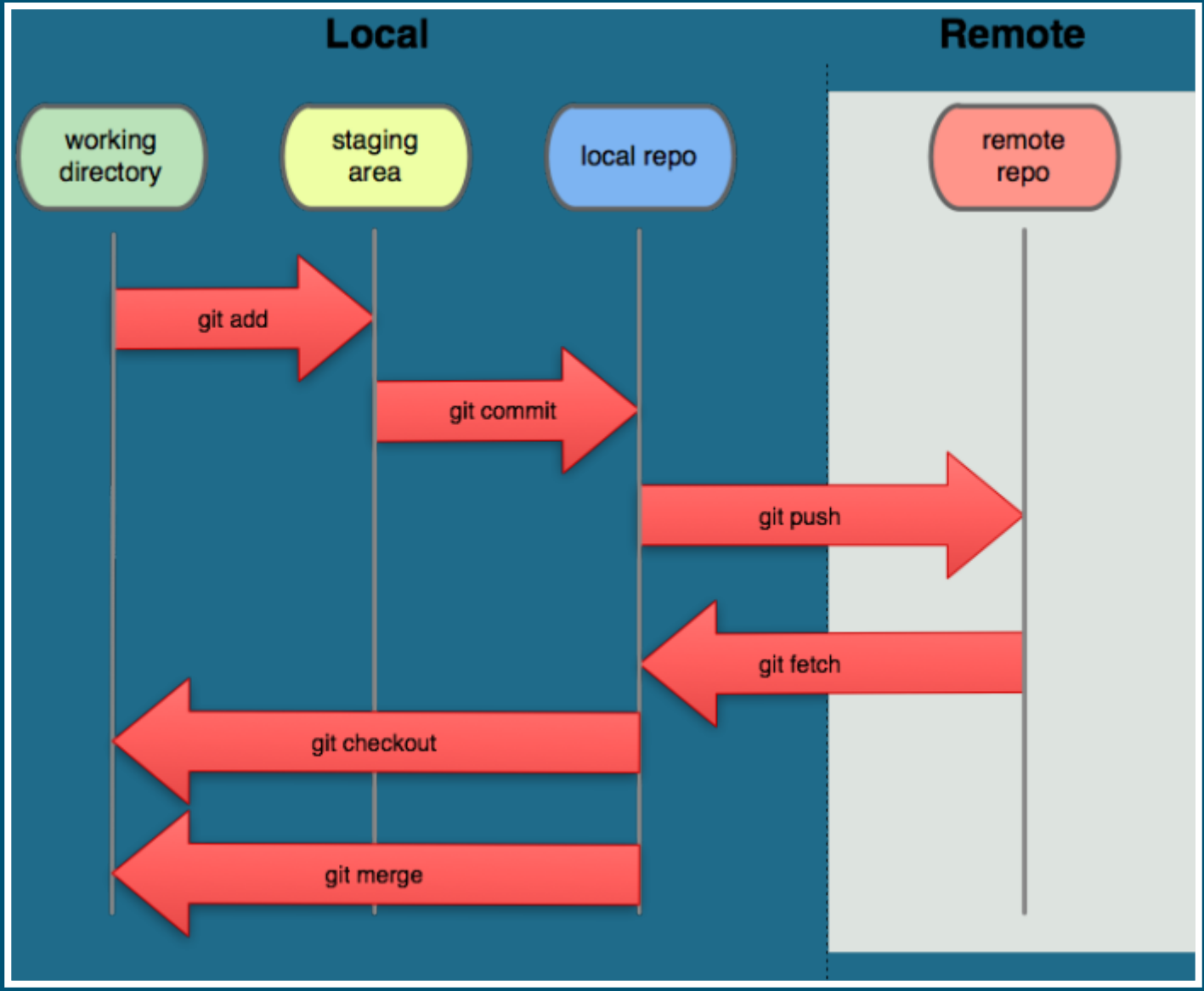
KLONEN (FORKEN)

```
#clone
git clone git://remote_adresse
git clone https://remote_adresse
```

PUSH UND PULL

```
#git push
git push remotename branchname
git push remotename lokaler_branch:remote_branch
#git poll
git pull remotename branchname
git pull remotename remote_branch:lokaler_branch
```

PUSH UND PULL


Local
Remote
working directory
staging area
local repo
remote repo
git add
git commit
git push
git fetch
git checkout
git merge

CODING PLATFORMS UND SOZIALE NETZWERKE

- SCM Manager
- Rhodecode
- Github
- Bitbucket
- Gitlab

CONTINUOUS INTERGATION/DELIVERY

- Merge in eine Hauptlinie mehrmals am Tag
- Tests
  - Unit Tests
  - System Tests
- Automatisiertes QA
- Automatisierte Produktionsbuilds

CI PROJEKTE

- Hudson
- Jenkins
- Travis CI
- Gitlab CI

#Links:

- Git Immersion
- Git Book
- Visual Git

#Thank you!