

Verlässliche Echtzeitsysteme - Übungen

Git

Wintersemester 2024

Eva Dengler, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl Informatik 4 (Systemsoftware)

<https://sys.cs.fau.de>

1 Versionsverwaltung mit git

Typische Aufgaben eines Versionsverwaltungssystems sind:

- *Sichern* alter Zustände
- *Zusammenführung* paralleler Entwicklung
- *Transportmedium*

Idealerweise zusätzlich:

- *Unabhängige Entwicklung* ohne zentrale Infrastruktur

- in VEZS: git
- 2005 von Linus Torvalds für den Linux-Kernel geschrieben
- Konsequenz der Erfahrungen mit *bitkeeper*
- Eigenschaften:
 - dezentrale, parallele Entwicklung
 - Koordinierung hunderter Entwickler
 - Visualisierung von Entwicklungszweigen

● master	origin/master	no ascii art needs formatting	Tobias Klaus	May 3 2016, 12:26
●		Every repository should contain a README file	Peter Waegemann	Apr 22 2016, 18:26
●		fix two warnings caused by ignored return values of system	Florian Franzmann	Apr 8 2014, 14:01
●		change colors to black and white theme	Florian Franzmann	May 27 2013, 18:46
●		fix warnings	Florian Franzmann	May 27 2013, 18:46
●		switch to cmake	Florian Franzmann	May 27 2013, 18:45
●		Anpassung fuer neuen gcc	Florian Franzmann	Jun 2 2008, 20:21
●		- fixed compiler warnings	Florian Franzmann	Sep 13 2009, 22:35
●		Sonnenauf/-untergang korrigiert	Florian Franzmann	Jul 18 2007, 22:45
●		gestrige Aenderung wieder rueckgaengig	Florian Franzmann	Jul 12 2007, 12:14
●		Monat fuer Sonnenaufgang/Untergang korrigiert	Florian Franzmann	Jul 12 2007, 00:23
●		Test fuer sunriset	Florian Franzmann	Feb 22 2007, 23:05
●		logarithmische Skalierung fuer tics	siflfran	May 5 2006, 12:56
●		Outlineblamodi	siflfran	May 5 2006, 12:47
●		*** empty log message ***	siflfran	May 4 2006, 21:50
●		Kreiskrams fertig	siflfran	May 4 2006, 21:48
●		Kreisbla angefangen	siflfran	May 4 2006, 20:43
●		Neon-Krams	siflfran	May 2 2006, 17:22



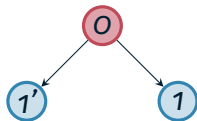
- Was speichert ein Commit?

- Was speichert ein Commit?
 - Wer? \rightsquigarrow Autor
 - Warum? \rightsquigarrow Commit-Nachricht
 - Was? \rightsquigarrow Vorher/Nachher *Zustände*



- Was speichert ein Commit?
 - Wer? \rightsquigarrow Autor
 - Warum? \rightsquigarrow Commit-Nachricht
 - Was? \rightsquigarrow Vorher/Nachher *Zustände*
 - Vorgänger Commits, auch *mehrere!*
 - *Keine* Nachfolger

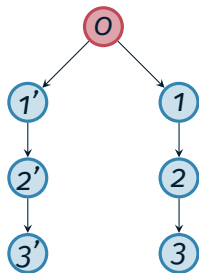
\rightsquigarrow Commit-Id: SHA-1 Hash über Inhalt



git-Commits

- Was speichert ein Commit?
 - Wer? \rightsquigarrow Autor
 - Warum? \rightsquigarrow Commit-Nachricht
 - Was? \rightsquigarrow Vorher/Nachher *Zustände*
 - Vorgänger Commits, auch *mehrere!*
 - *Keine* Nachfolger

- \rightsquigarrow Commit-Id: SHA-1 Hash über Inhalt
- \rightsquigarrow Gerichteter Azyklischer Graph (DAG)
 - \rightsquigarrow Sprünge zurück *möglich*
 - \rightsquigarrow Sprünge vorwärts *nicht möglich*

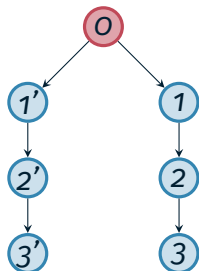


- Was speichert ein Commit?
 - Wer? \rightsquigarrow Autor
 - Warum? \rightsquigarrow Commit-Nachricht
 - Was? \rightsquigarrow Vorher/Nachher *Zustände*
 - Vorgänger Commits, auch *mehrere!*
 - *Keine* Nachfolger

- \rightsquigarrow Commit-Id: SHA-1 Hash über Inhalt
- \rightsquigarrow Gerichteter Azyklischer Graph (DAG)
 - \rightsquigarrow Sprünge zurück *möglich*
 - \rightsquigarrow Sprünge vorwärts *nicht möglich*

- Woher “obere“ Commits?

- \rightsquigarrow Symbolische Namen (Zeiger)
 - HEAD: Aktueller Commit
 - Branch: Zeiger auf Commit



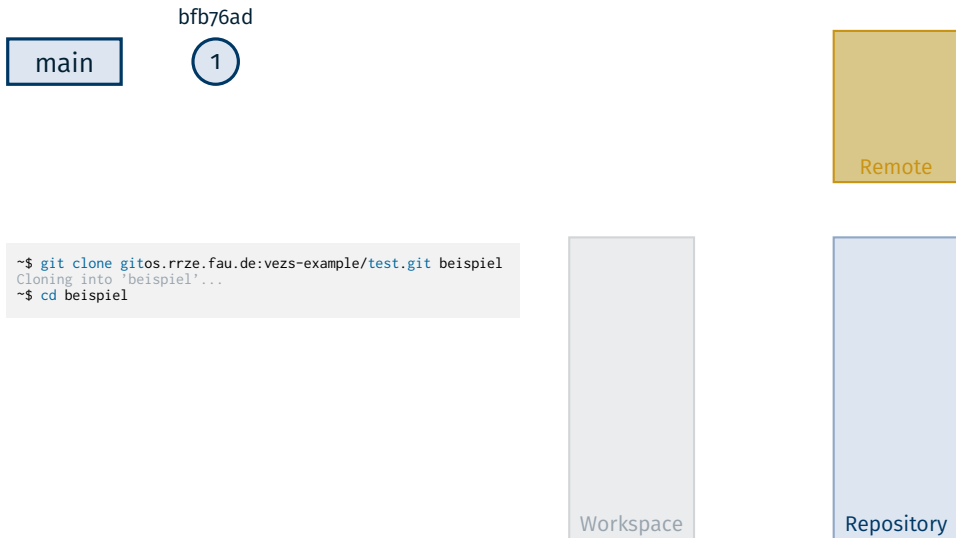
Git-Repository einrichten



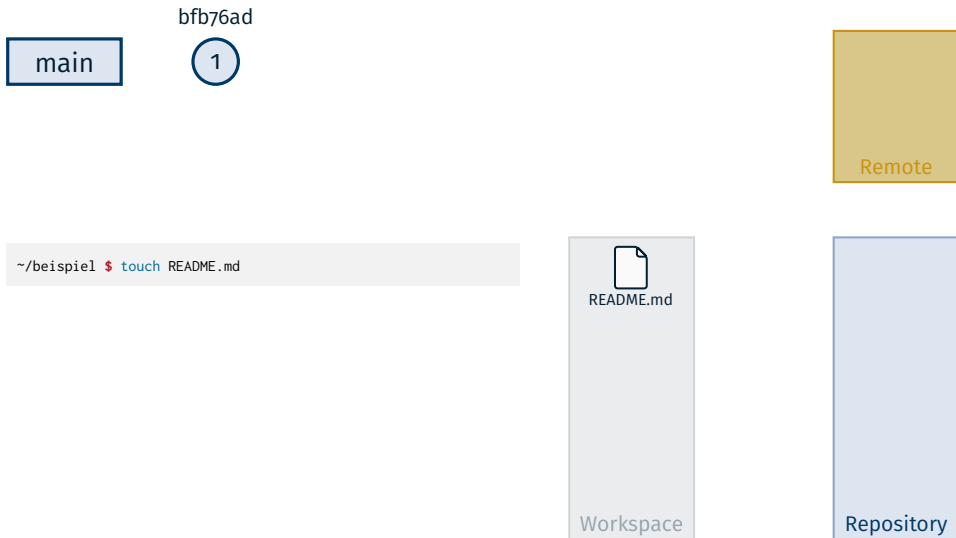
```
~$ git clone gitos.rrze.fau.de:vezs-example/test.git beispiel  
Cloning into 'beispiel'...
```



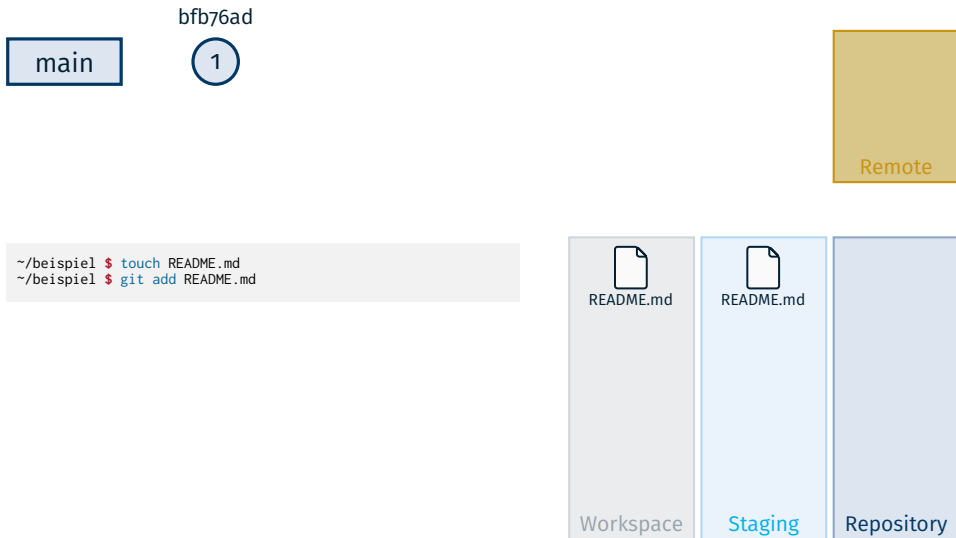
Git-Repository einrichten



Dateien mit GIT verwalten



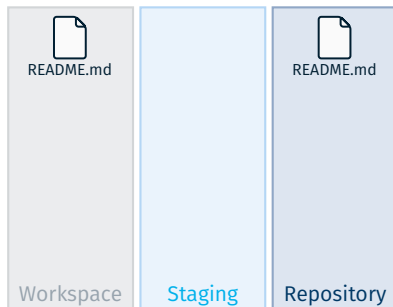
Dateien mit GIT verwalten



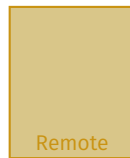
Dateien mit GIT verwalten



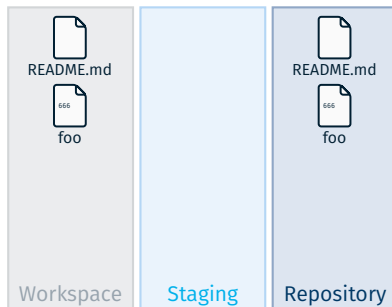
```
~/beispiel $ touch README.md
~/beispiel $ git add README.md
~/beispiel $ git commit -m "Liesmich hinzugefügt"
[main bd2de5c] Liesmich hinzugefügt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
```



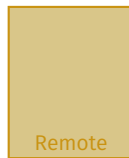
Dateien mit GIT verwalten



```
~/beispiel $ echo "666" > foo
~/beispiel $ git add foo
~/beispiel $ git commit -m "Datei foo erstellt"
[main df7aa5a] Datei foo erstellt
1 file changed, 1 insertion(+)
 create mode 100644 foo
```



Dateien mit GIT verwalten

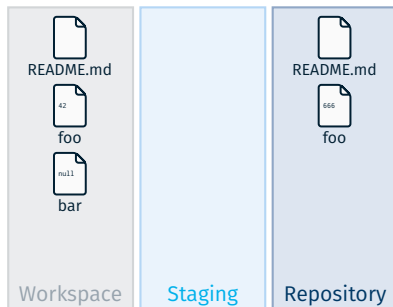


```
~/beispiel $ echo "42" > foo
~/beispiel $ echo "null" > bar
~/beispiel $ git status
On branch main
Your branch is up to date with 'origin/main'.

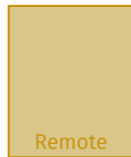
Changes not staged for commit:
  modified: foo

Untracked files:
  bar

no changes added to commit
```



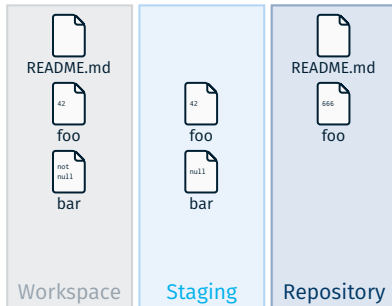
Dateien mit GIT verwalten



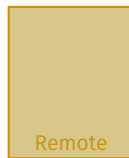
```
~/beispiel $ git add foo bar
~/beispiel $ echo "not null" > bar
~/beispiel $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  new file:   bar
  modified:   foo

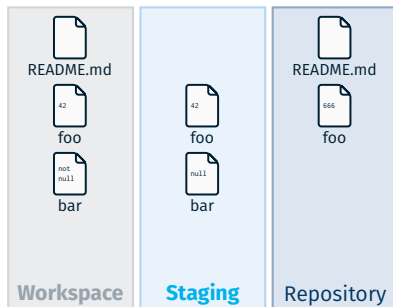
Changes not staged for commit:
  modified:   bar
```



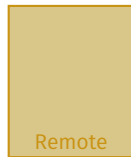
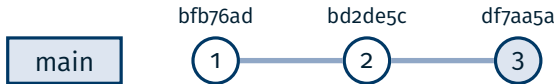
Dateien mit GIT verwalten



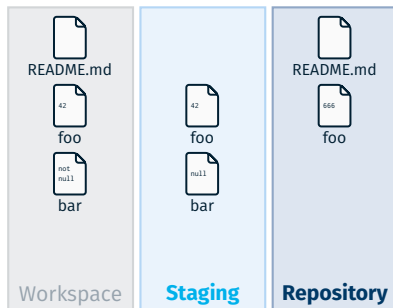
```
~/beispiel $ git diff
diff -git a/bar b/bar
index 19765bd..b263a85 100644
- a/bar
+++ b/bar
@@ -1 +1 @@
-null
+not null
```



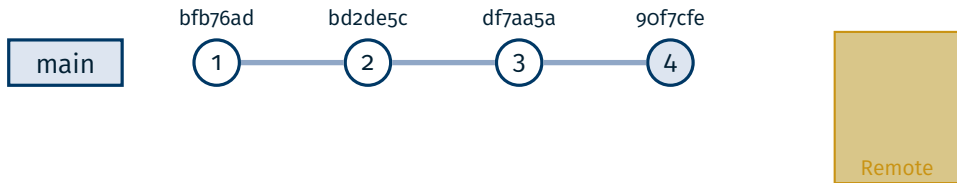
Dateien mit GIT verwalten



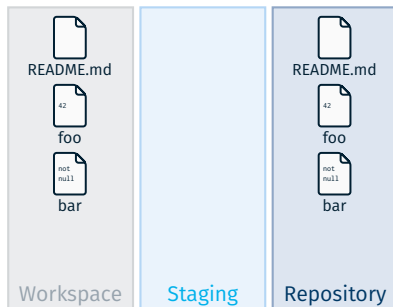
```
~/beispiel $ git diff --staged
diff -git a/bar b/bar
new file mode 100644
index 0000000..19765bd
- a/bar
+++ b/bar
@@ -0,0 +1 @@
+null
diff -git a/foo b/foo
index 7cc86ad..d81cc07 100644
[...]
```



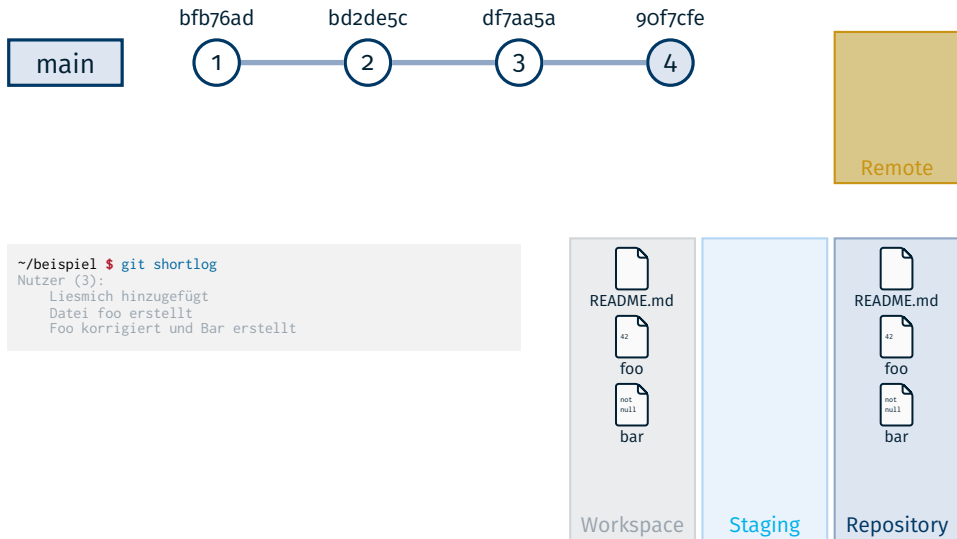
Dateien mit GIT verwalten



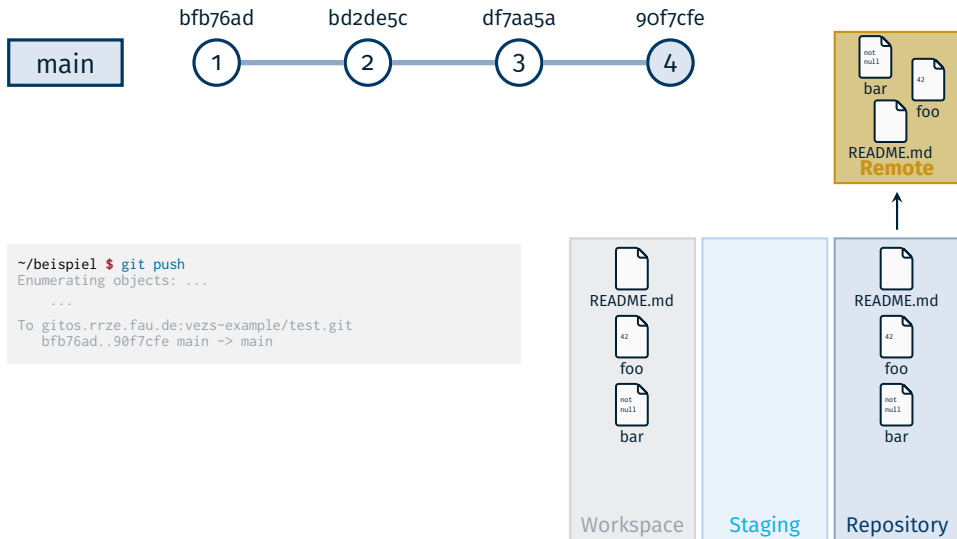
```
~/beispiel $ git add bar
~/beispiel $ git commit -m \
    "Foo korrigiert und Bar erstellt"
[main 90f7cfe] Foo korrigiert und Bar erstellt
2 files changed, 2 insertions(+), 1 deletion(-)
create mode 100644 bar
```



Dateien mit GIT verwalten



Dateien mit GIT verwalten



git push [<remote> [<branch>]]

- schiebt Commits nach <remote> in den ausgewählten <branch>
- dies geht nur, wenn lokales Repo auf dem aktuellen Stand ist!
- sonst beschwert sich git:

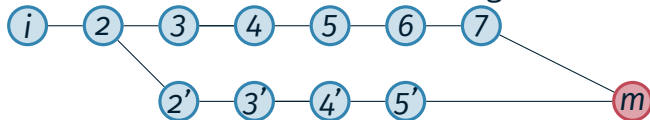
```
% git push origin main
```

```
To /tmp/test.git
! [rejected]          main -> main (non-fast-forward)
error: failed to push some refs to '/tmp/test.git'
To prevent you from losing history, non-fast-forward updates were rejected
Merge the remote changes (e.g. 'git pull') before pushing again.  See the
'Note about fast-forwards' section of 'git push --help' for details.
```

~> wir müssen das Repository erst auf den aktuellen Stand bringen

git pull [<remote> [<branch>]]

- holt Änderungen aus remote in den aktuellen Branch
- verschmilzt aktuellen Branch mit geholten Änderungen



% git pull origin

```
remote: Counting objects: 5, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From /tmp/test
 38b95cb..8ec6e93  main      -> origin/main
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
```

- Änderungen an gleicher Stelle in der Zwischenzeit
~> Konflikte müssen von Hand behoben werden

Konflikt beheben (1)

```
% cat test.txt
```

```
hallo  
<<<<<<< HEAD  
welt!     meine Version  
=====  
Welt!     Version in origin/main  
>>>>>>> 8ec6e9309fa37677e2e7ffc9553a6bebf8827d6
```

■ Konflikt auflösen:

- Manuelles Bearbeiten
- Übernahme des eigenen, Verwerfen des fremden Standes:
git checkout --ours test.txt
- Übernahme des fremden, Verwerfen des eigenen Standes:
git checkout --theirs test.txt

Konflikt beheben (2)

- Konfliktlösung an git signalisieren:

```
% git add test.txt && git commit
```

```
[main 4d21871] Merge branch 'main' of /tmp/test
```

```
% git push origin main
```

```
Counting objects: 5, done.  
Writing objects: 100% (3/3), 265 bytes, done.  
Total 3 (delta 0), reused 0 (delta 0)  
Unpacking objects: 100% (3/3), done.  
To /tmp/test.git  
8ec6e93..278c740 main -> main
```

- *Juhu!*

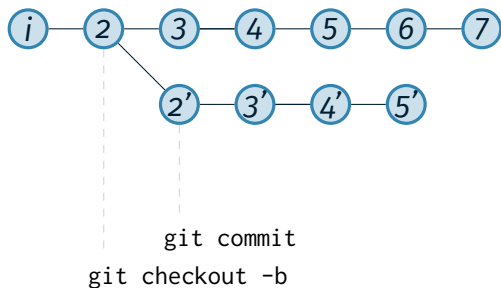
Verzweigungen und Zusammenführungen

Beispiel für parallele Entwicklung:



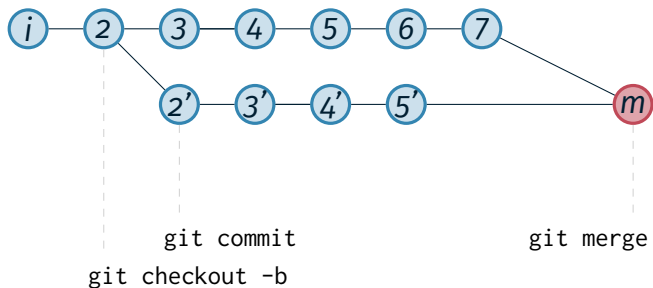
Verzweigungen und Zusammenführungen

Beispiel für parallele Entwicklung:



Verzweigungen und Zusammenführungen

Beispiel für parallele Entwicklung:



In den meisten Versionsverwaltungssystemen

1. Featurebranch anlegen
2. Feature im Branch implementieren, testen
3. Featurebranch mit main verschmelzen
4. ggf. Featurebranch löschen

In den meisten Versionsverwaltungssystemen

1. Featurebranch anlegen
2. Feature im Branch implementieren, testen
3. Featurebranch mit main verschmelzen
4. ggf. Featurebranch löschen

Naiver Ansatz

~> skaliert nicht!

Warum branch/edit/merge nicht skaliert

Aufgaben von Versionsverwaltung

1. Codeschreiben unterstützen
2. Konfigurationsmanagement/Branches
~> z. B. Release-Version, HEAD-Version ...

Warum branch/edit/merge nicht skaliert

Aufgaben von Versionsverwaltung

1. Codeschreiben unterstützen
2. Konfigurationsmanagement/Branches
~> z. B. Release-Version, HEAD-Version ...

~> **Konflikt**

1. braucht Checkpoint-Commits
 - möglichst oft einchecken
 - ~> skaliert nicht
2. braucht Stable-Commits
 - nur einchecken, wenn Commit perfekt
 - ~> nicht praktikabel

Lösung mit git: öffentlicher vs. privater Branch

Öffentlicher Branch \leadsto verbindliche Geschichte

Commits sollen $\left. \begin{array}{l} \text{atomar} \\ \text{gut dokumentiert} \\ \text{linear} \\ \text{unveränderlich} \end{array} \right\}$ sein

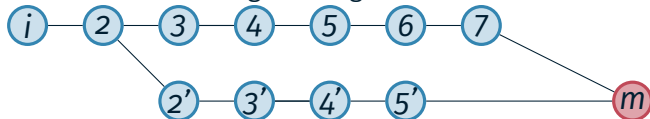
Privater Branch \leadsto Schmierpapier

- für einzelnen Entwickler
- möglichst lokal
- wenn im zentralen Repo \leadsto auf Privatheit einigen

Aufräumen

- verschmelze nie direkt privaten mit öffentlichem Branch
 - Historie wird sonst unübersichtlich

↪ nicht einfach `git merge` im main machen



- vorher immer erst `git`
 - `rebase` ↪ Commits auf Branch anwenden
 - `merge --squash` ↪ einzelnen Commit aus Branch-Commits
oder: `rebase --interactive` ↪ Commits umgruppieren
 - `commit --amend` ↪ letzten Commit überarbeiten
- Ziel: öffentlicher Commit \equiv Kapitel eines Buches

Michael Crichton

Great books aren't written – they're rewritten.

Arbeitsablauf für kleinere Änderungen

- `git merge --squash` \rightsquigarrow Änderungen aus Branch in aktuellen Index

Branch

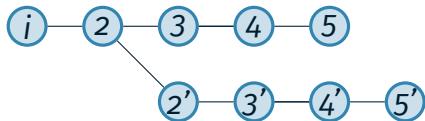
```
% git checkout -b private_feature_branch (Branch anlegen)
% touch file1.txt file2.txt
% git add file1.txt; git commit -am "WIP1" (file1.txt einchecken)
% git add file2.txt; git commit -am "WIP2" (file2.txt einchecken)
```

Merge

```
% git checkout main (nach main wechseln)
% git merge --squash private_feature_branch (Änderungen auf main)
% git commit -v (Änderungen einchecken)
```

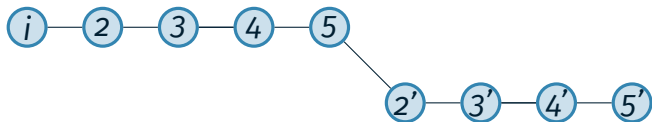
git rebase <branch>

- Aufsetzen auf bestehenden <branch>



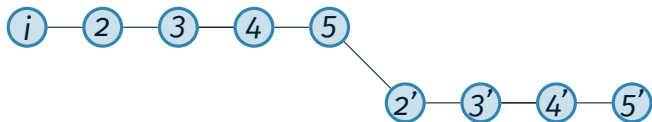
git rebase <branch>

- Aufsetzen auf bestehenden <branch>



git rebase <branch>

- Aufsetzen auf bestehenden <branch>



- Patches aus „unterem“ Zweig werden auf „oberen“ aufgespielt
- Die Historie ist nun linear
- Linearisierte Änderungen lassen sich häufig einfacher bewerten
- **Vorsicht!**
 - Verzweigungen von alten Zweig können nicht mehr zusammengeführt werden
 - Keine gemeinsamen Vorgänger mehr
 - Visualisierung der Historie ist nun bestenfalls verwirrend

git rebase -interactive <commit>

- schreibt Geschichte um

```
git rebase -interactive ccd6e62^
```

pick ~> übernimmt Commit

```
pick ccd6e62 Work on back button
pick 1c83feb Bug fixes
pick f9d0c33 Start work on toolbar
```

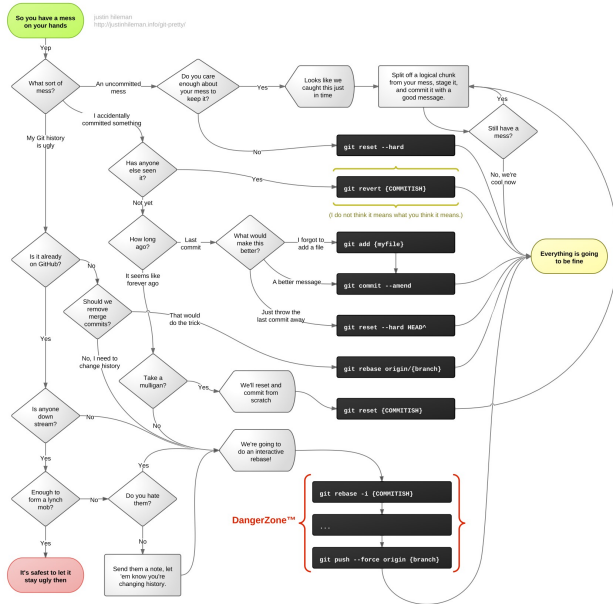
fixup ~> verschmilzt Commit mit Vorgänger

```
pick ccd6e62 Work on back button
fixup 1c83feb Bug fixes # mit Vorgaenger verschmelzen
pick f9d0c33 Start work on toolbar
```

reword ~> Beschreibung editieren

edit ~> kompletten Commit editieren

Geschichte neuschreiben



`git push -force`



`git push -force`



~> Wenn wirklich erforderlich: `git push --force-with-lease`

git stash [pop]

- Vorübergehendes Sichern von Änderungen
 - `git stash` und `git stash pop`
- ↪ sichert Änderungen an der Working Copy auf Stapel
- rebase braucht saubere Working Copy
⇒ vorher `git stash`

Im Feature-Branch

```
% git stash
```

```
Saved working directory and index state WIP on main: 81c0895 cmake  
HEAD is now at 81c0895 cmake, git ...
```

```
% ...
```

```
% git stash pop
```

Wenn der Feature-Branch im Chaos versinkt?

~> aufgeräumten Branch anlegen

1. auf Branch main wechseln

```
% git checkout main
```

2. Branch aus main erzeugen

```
% git checkout -b cleaned_up_branch
```

3. Branch-Änderungen in den Index und die Working Copy ziehen

```
% git merge --squash private_feature_branch
```

4. Index zurücksetzen

```
% git reset
```

■ danach Commits neu zusammenbauen

~> Auf der Konsole: `git add -p`

~> Graphisch: `git cola`

- Historie des HEAD-Zeigers
- Historie aller Befehle, die HEAD verändern

git reflog

```
8afd010 HEAD@{0}: rebase -i (finish): returning to refs/heads/main
8afd010 HEAD@{1}: checkout: moving from main to 8afd010ae2ab48246d5
7f97fab HEAD@{2}: commit: Pentax K20D fw version 1.04.0.11 wb presets
8c37332 HEAD@{3}: rebase -i (finish): returning to refs/heads/main
8c37332 HEAD@{4}: checkout: moving from main to 8c373324ca196c337dd
9d66ec9 HEAD@{5}: clone: from git://github.com/darktable-org/darkt...
```

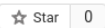
- `git reset --hard HEAD@{2}` stellt alten Zustand wieder her

- Selbstverwaltet auf <https://gitos.rrze.fau.de/>
- Login über Single-Sign-On mit IDM-Kennung
- **Regeln auf der Hauptseite beachten!**
- Abgaberepository pro Gruppe:
<https://gitos.rrze.fau.de/i4/teaching/vezs/WS24/group<groupid>>
- Arbeitskopie als Fork erstellen

[ezs](#) > [DemoSemester](#) > [group42](#) > **Details**



group42 
Project ID: 11440



- Abgabe per Merge-Request gegen
<https://gitos.rrze.fau.de/i4/teaching/vezs/WS24/group<groupid>>

SSH-Schlüssel konfigurieren

- SSH Schlüssel erzeugen:

~> % ssh-keygen -t rsa -f ~/.ssh/gitlab

- SSH Schlüssel für Authentifizierung hinterlegen:

~> Kopieren: % xclip -selection clipboard .ssh/gitlab.pub

~> Einfügen im Gitlab unter: Benutzer → Einstellungen → SSH-Keys

- <https://gitos.rrze.fau.de/help/user/ssh>

- SSH-Config anpassen, damit der neue Schlüssel auch verwendet wird:

\$HOME/.ssh/config

```
Host gitos.rrze.fau.de
  IdentityFile ~/.ssh/gitlab
```

Dateien ignorieren mit git

- Dateien erscheinen nicht in der Ausgabe von bspw. `git status`
- Änderungen an Dateien werden ignoriert

`.gitignore`

```
# Ignore LaTeX temporary files
*.aux
*.log

# except this one
!important.log

# everything under directory
solutions/
```

- Auch global möglich: `$HOME/.gitconfig`

git-Konfiguration des Repositories

Per Befehlszeile

```
% git remote set-url origin git@gitos.rrze.fau.de:<user>/<projekt>.git  
% git remote add vorgabe git@gitos.rrze.fau.de:ezs/vezs24-vorgabe.git
```

.git/config

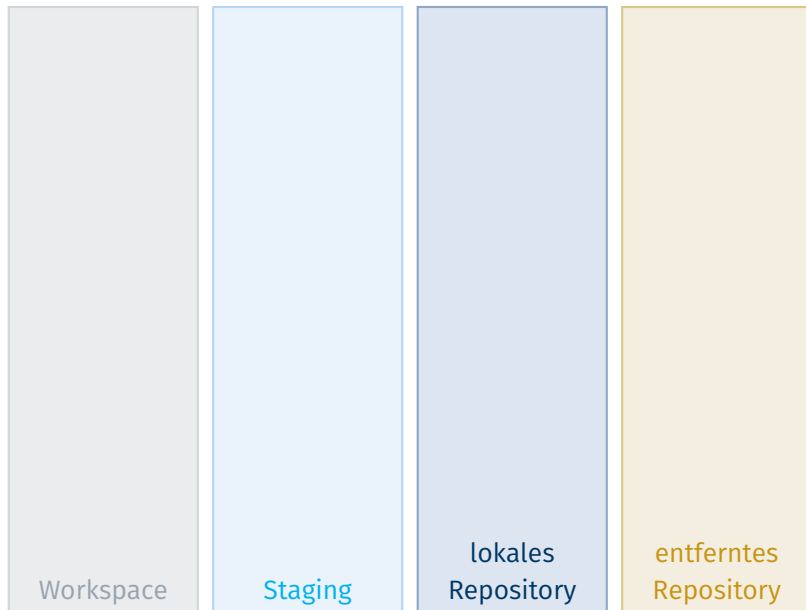
```
[remote "origin"]  
  fetch = +refs/heads/*:refs/remotes/origin/*  
  url = git@gitos.rrze.fau.de:<username>/<projektname>.git  
  
[remote "vorgabe"]  
  fetch = +refs/heads/*:refs/remotes/origin/*  
  url = git@gitos.rrze.fau.de:ezs/vezs24-vorgabe.git
```

Globale git-Konfiguration des Systems

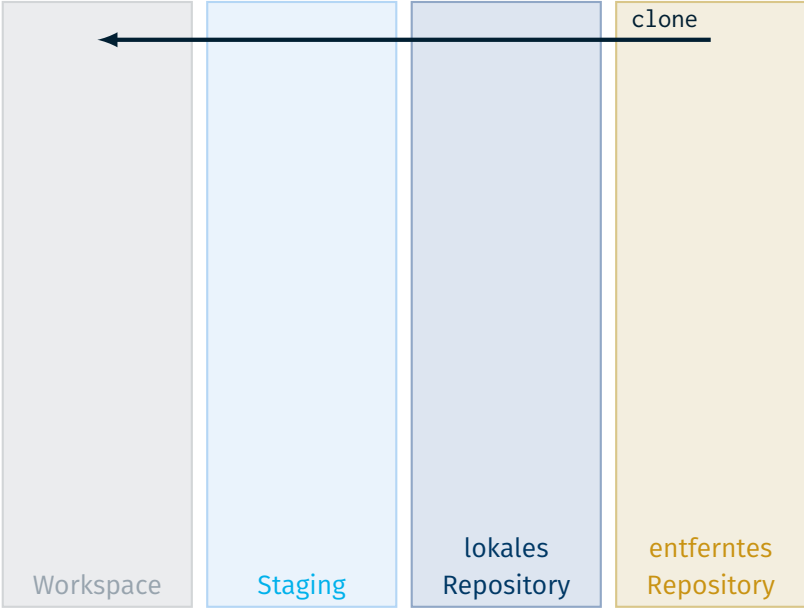
`$HOME/.gitconfig`

```
[user]
  name = Max Mustermann
  email = max.mustermann@fau.de
[core]
  editor = <maxs-lieblingseditor>
[commit]
  verbose = true
[color]
  ui = auto
[alias]
  s      = status
  a      = add
  c      = commit
  co     = checkout
  b      = branch -a -v
  unstage = reset HEAD --
  visual = !gitk
  lg     = log --graph \
          --abbrev-commit \
          --date=relative
```

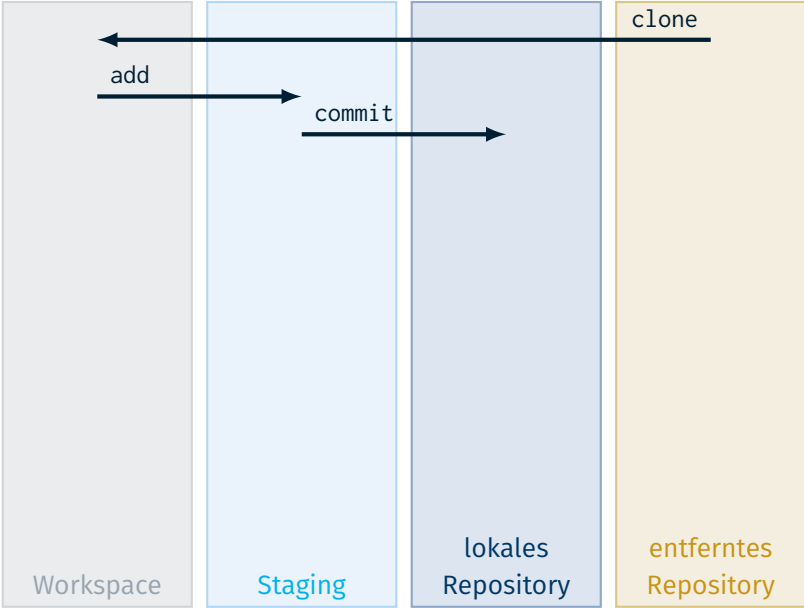
Überblick: Dateien mit GIT verwalten



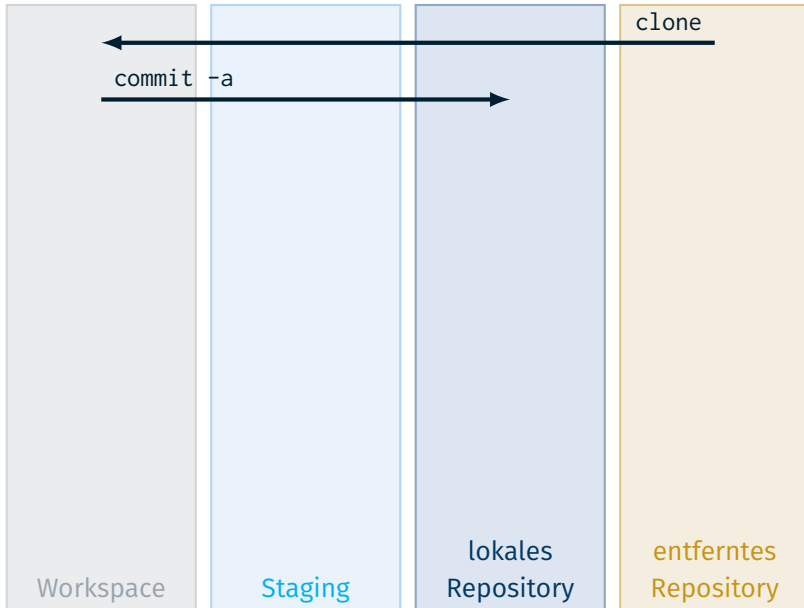
Überblick: Dateien mit GIT verwalten



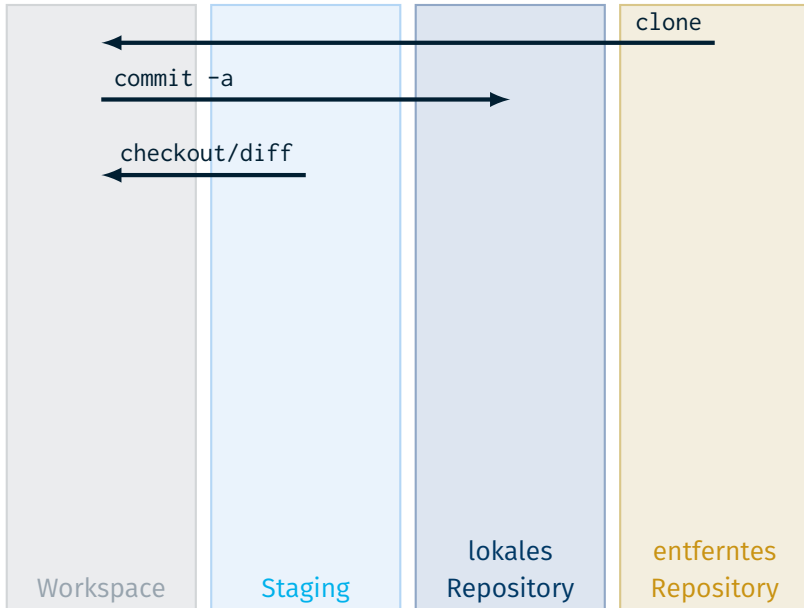
Überblick: Dateien mit GIT verwalten



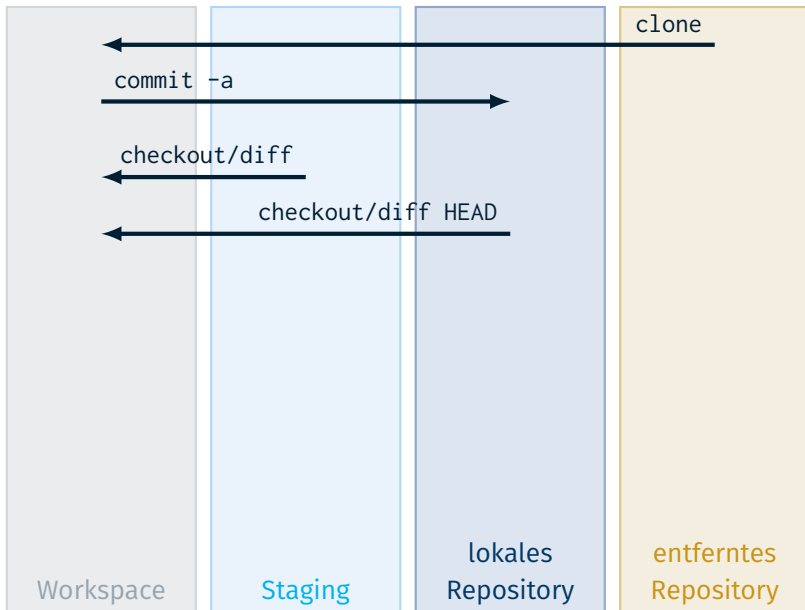
Überblick: Dateien mit GIT verwalten



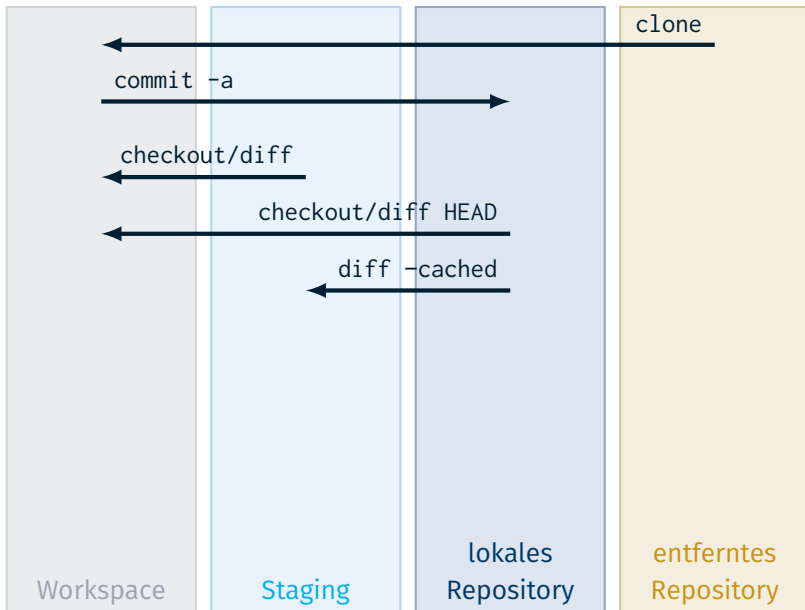
Überblick: Dateien mit GIT verwalten



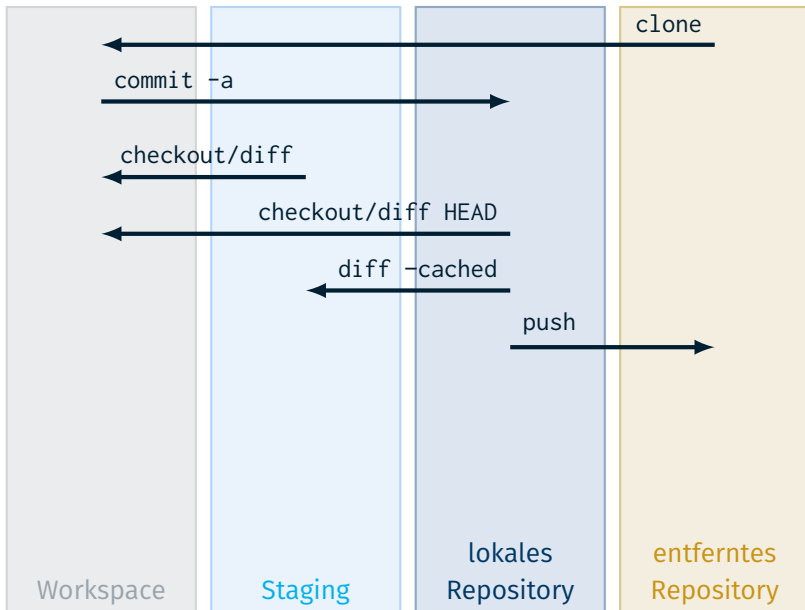
Überblick: Dateien mit GIT verwalten



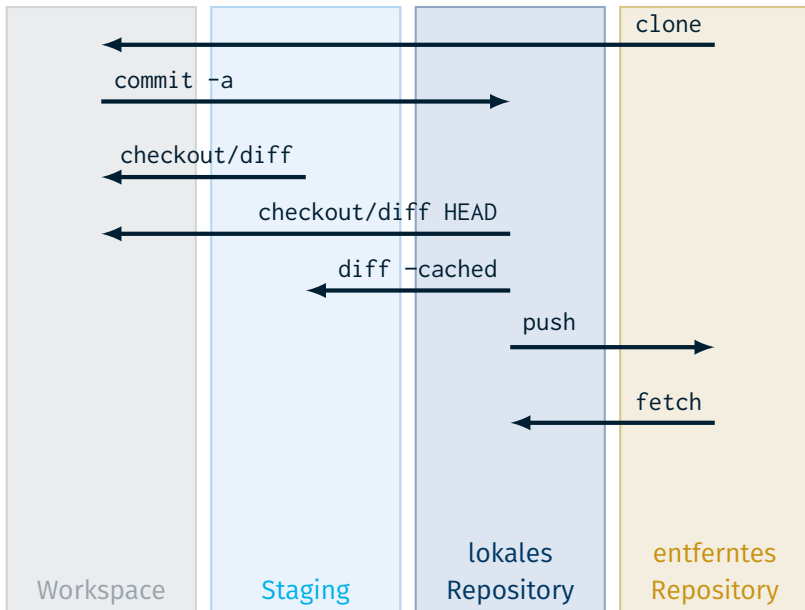
Überblick: Dateien mit GIT verwalten



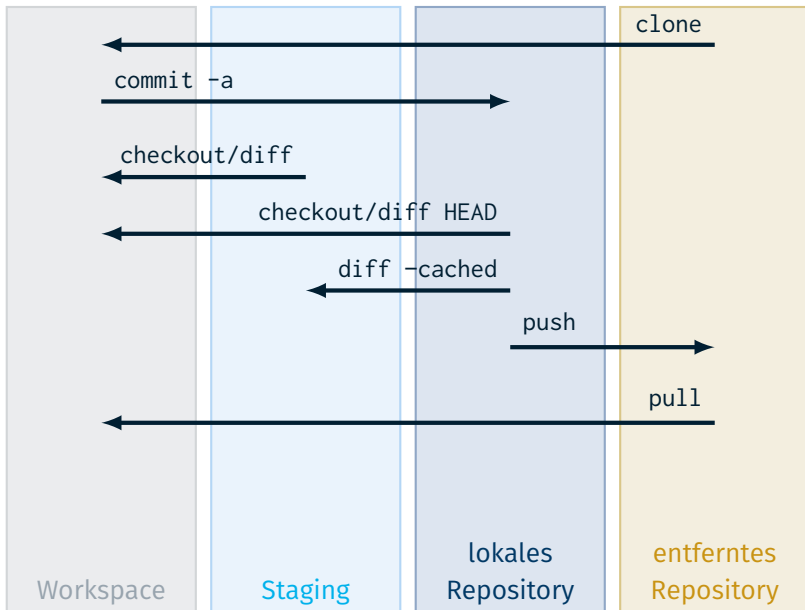
Überblick: Dateien mit GIT verwalten



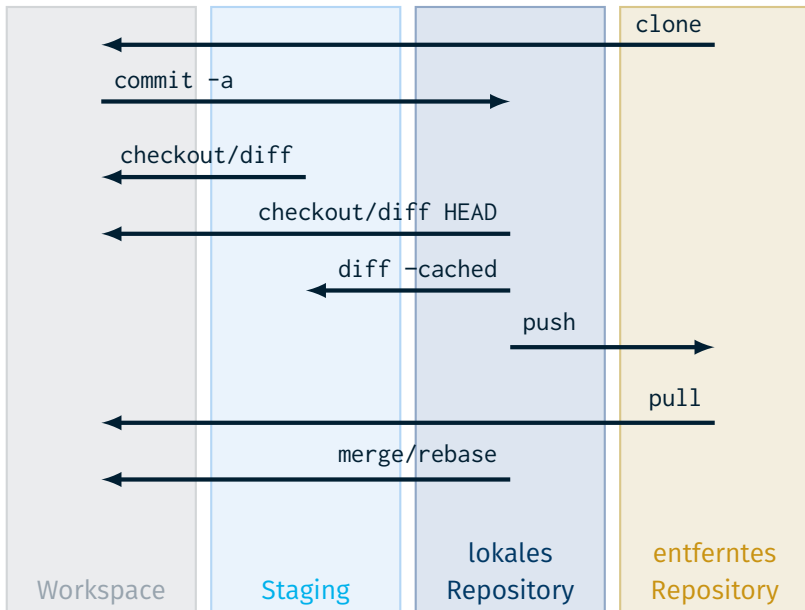
Überblick: Dateien mit GIT verwalten



Überblick: Dateien mit GIT verwalten



Überblick: Dateien mit GIT verwalten



Git Cheatsheet

- git init** initiales Anlegen eines Repositories
- git clone <url>** initiales Kopieren von einer Quelle
- git pull** kurz für Holen und Zusammenfügen
- git push** Übertragen in entfernte Quelle
- git add <file>** Datei als Kandidat für nächsten *commit* markieren
- git add -p** feingranular...
- git add -i** ... oder interaktiv
- git diff** unversionierte Änderungen anzeigen
- git diff -cached** Änderungen des nächsten Commits anzeigen
- git commit** Änderungen versionieren
- git clean -d <path>** alles, was nicht im git ist, löschen
- git clean -n -d <path>** anzeigen, was gelöscht werden würde

Git Cheatsheet (II)

git show neuste (versionierte) Änderungen anzeigen

git status Änderungen zum Vorgänger anzeigen

git log Historie anzeigen

git checkout - <path> geänderte, aber nicht eingetragene Datei zurücksetzen

git checkout <branch> zu einem Branch wechseln

git revert <id> Commit rückgängig machen

git branch <name> Branch anlegen

git branch -a alle Branches anzeigen

man git-<cmd> Hilfe anzeigen, z.B. man git-add

- <http://gitready.com>
- <http://book.git-scm.com/>
- <http://eagain.net/articles/git-for-computer-scientists/>
- <http://sandofsky.com/blog/git-workflow.html>
- <http://365git.tumblr.com/>

- Zum Ausprobieren:
 - <https://learngitbranching.js.org/>
 - <https://github.com/git-game/git-game>
 - `man git-tutorial`, `man git-tutorial2`

42