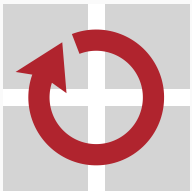
Übungen zu Systemprogrammierung 1

Üo – Einführung

Sommersemester 2023

Jonas Rabenstein, Eva Dengler, Luis Gerhorst, Dustin Nguyen, Christian Eichler, Jürgen Kleinöder

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Agenda

0.1 Allgemeines

0.2 Organisatorisches

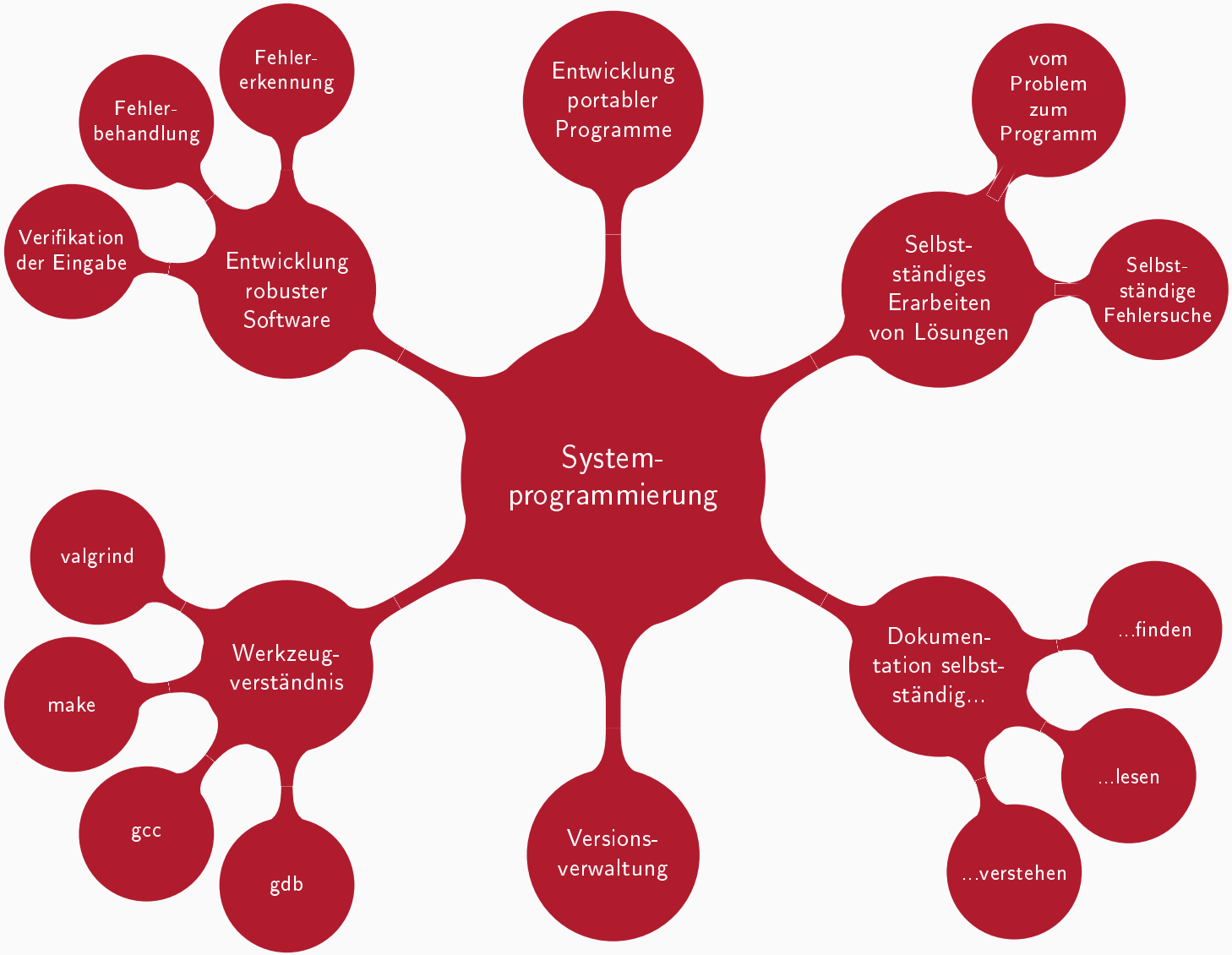
0.3 Linux-Kenntnisse

0.4 Versionsverwaltung mit Git

0.5 SP-Abgabesystem

Agenda

Lernziele Systemprogrammierung



Aufbau der SP-Übungen

Tafelübungen

- Vorstellung von Betriebssystemkonzepten und Werkzeugen
- Einführung in die Verwendung der Schnittstellen
- Erarbeiten eines kleinen Programmes (Demo)
- Besprechung der Abgaben und allgemeiner Fallstricke

Praktischer Teil – Aufgaben

- Arbeiten mit der Betriebssystemschnittstelle
- Fehlersuche und Fehlerbehebung
- Verwenden der vorgestellten Werkzeuge

Rechnerübungen

- Hilfestellung für Aufgaben

Agenda

Praktischer Teil – Aufgaben

- Ausgabe neuer Aufgaben in den Tafelübungen
  - Aufgabenstellung meist recht knapp
    - $\Rightarrow$ nicht alles bis in letzte Detail spezifiziert
  - Gegebene Spezifikationen sind zwingend einzuhalten
- Selbstständiges Bearbeiten der Aufgaben
  - bei Problemen hilft z. B. ein Besuch in den Rechnerübungen
- Korrektur und Bewertung durch die Tutoren
  - Korrekturen werden elektronisch zur Verfügung gestellt
  - eigenes Ergebnis nach Login im *WAFFEL* einsehbar
  - Korrekturrichtlinen auf Webseite dokumentiert
- Übungspunkte können das Klausurergebnis verbessern
  - Abschreibtests
  - Vorstellen der eigenen Lösungen
    - $\rightarrow$ **Anwesenheit in Besprechungsübungen für Bonuspunkte**
  - Notenbonus nur bei bestandener Klausur

Praktischer Teil – Bearbeitung der Aufgaben

- Bearbeitungszeitraum angegeben in Werktagen (Mo. bis Fr.)
  - Bearbeitungszeitraum beinhaltet Tag der Tafelübung
  - Feiertage und „Berg-Dienstag” (nach Pfingsten) nicht enthalten
  - Abgabetermin kann per Skript erfragt werden
- plant mit **mindestens** 8–16 Stunden (in Worten: ein bis zwei **Tage**) Bearbeitungszeit pro Aufgabe
  - langer Bearbeitungszeitraum bietet Flexibilität bei der Arbeitsverteilung
  - Feedback über wirkliche Bearbeitungszeit erwünscht

Kontakt zum *SP-Team*

- Mailingliste: `i4sp@cs.fau.de`
  - geht an alle Tutoren
  - Angelegenheiten, die nur die eigene Person/Gruppe betreffen
- Mailingliste: `i4sp-orga@cs.fau.de`
  - geht an die SP-Organisatoren
  - Fragen zur Organisation und zum Übungsbetrieb
- Rechnerübungen (siehe Homepage)
  - Hilfe bei konkreten Problemen (z. B. Quellcode kompiliert nicht)
  - **kein** Händchenhalten, während ihr die Tastatur bedient :)
- der korrigierende Tutor
  - Fragen zur Korrektur, vergessener Gruppenbonus
  - fälschlicherweise positiver Abschreibtest
- Forum: `https://studon.fau.de/crs2958245_join.html`
  - inhaltliche Fragen zum Stoff oder den Aufgaben
  - allgemein alles, was auch für andere Teilnehmer interessant sein könnte

Agenda

Linux

- Grundkenntnisse zur Linux- und Shell-Nutzung werden vorausgesetzt
- Bei Bedarf:
  - Linux-Kurs der FSI Informatik (inkl. Aufzeichnung):
    `https://fsi.cs.fau.de/linuxkurs`
  - Bei Problemen:
    Fragen stellen in den ersten Wochen der Rechnerübungen
- CIP als Referenzsystem
  - Einführung zum CIP:
    `https://wwwcip.cs.fau.de/documentation/overview.de.html`
  - Auflistung der Rechnerausstattung:
    `https://wwwcip.cs.fau.de/cipPools/roomIndex.de.html`
  - Benutzung über den Webbrowser:
    `https://remote.cip.cs.fau.de`
  - Liste der SSH Host Keys:
    `https://wwwcip.cs.fau.de/documentation/sshhostkeys.de.html`

Dokumentation aus 1. Hand: Manual-Pages

- Aufgeteilt in verschiedene *Sections*
  1. Kommandos
  2. Systemaufrufe
  3. Bibliotheksfunktionen
  5. Dateiformate (Spezielle Datenstrukturen etc.)
  7. Verschiedenes (z. B. Terminaltreiber, IP)
- Angabe normalerweise mit *Section*: `printf(3)`

```
> man 3 printf                          # man [section] begriff
```

- Suche nach *Sections*:

```
> man -f <begriff>
```

- Suche nach Manual-Pages zu einem Stichwort:

```
> man -k <stichwort>
```

- **Achtung:** Manual-Pages unter Mac OS oft abweichend von Linux
  $\Rightarrow$ CIP ist Referenzsystem!

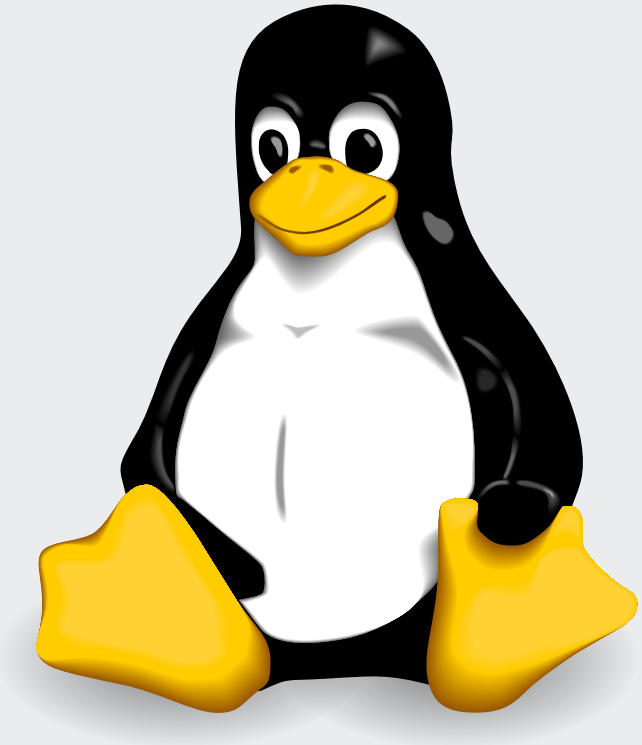
Demo: man-page

Veranstaltungshinweis

Für alle, die noch kein Linux auf dem eigenen Rechner haben, diesen Zustand aber gerne ändern würden:

Linux-Install-Party der FSI

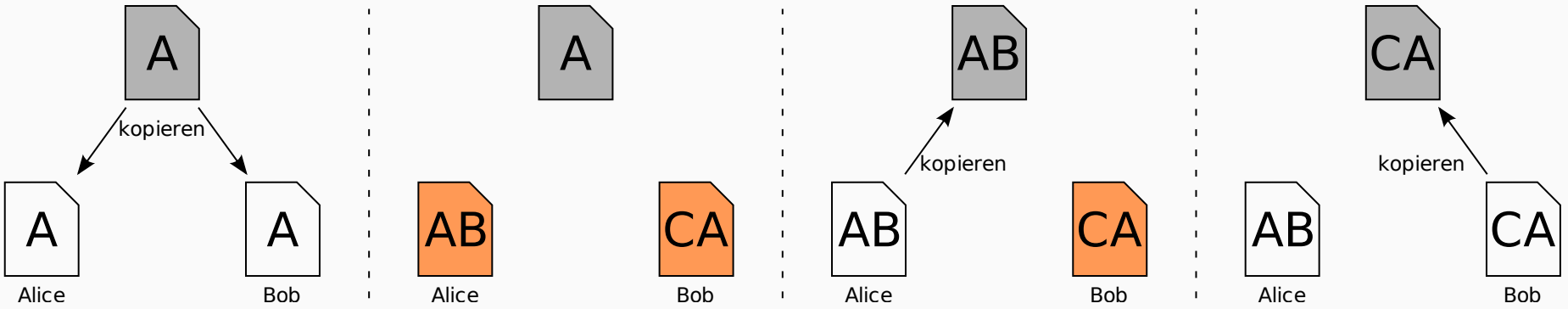
- am Dienstag, den 02. Mai
- von 10:00 bis 18:00
- im 02.152-113 („Vorstandszimmer“, Blaues Hochhaus, 2. Stock)
- weitere Informationen unter `https://fsi.cs.fau.de/linuxinstall`



Agenda

Warum Versionsverwaltung?

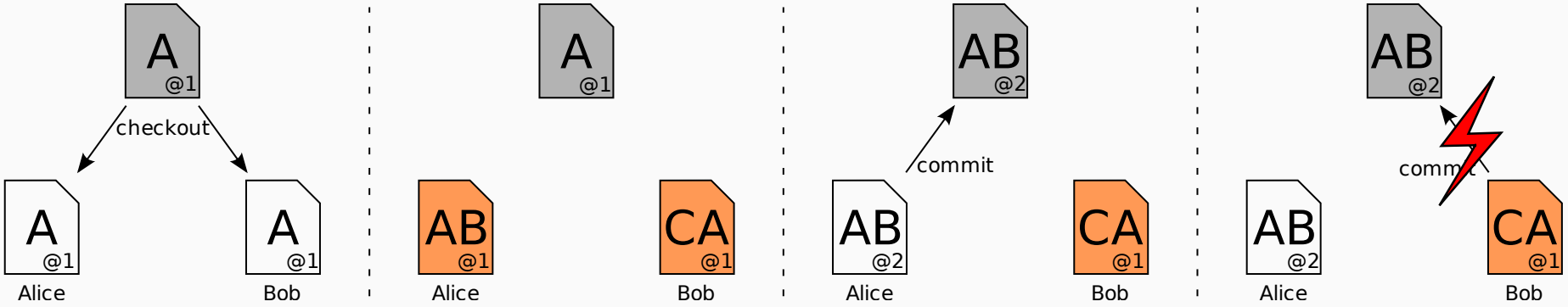
- Gemeinsames Bearbeiten einer Datei kann zu Problemen führen:



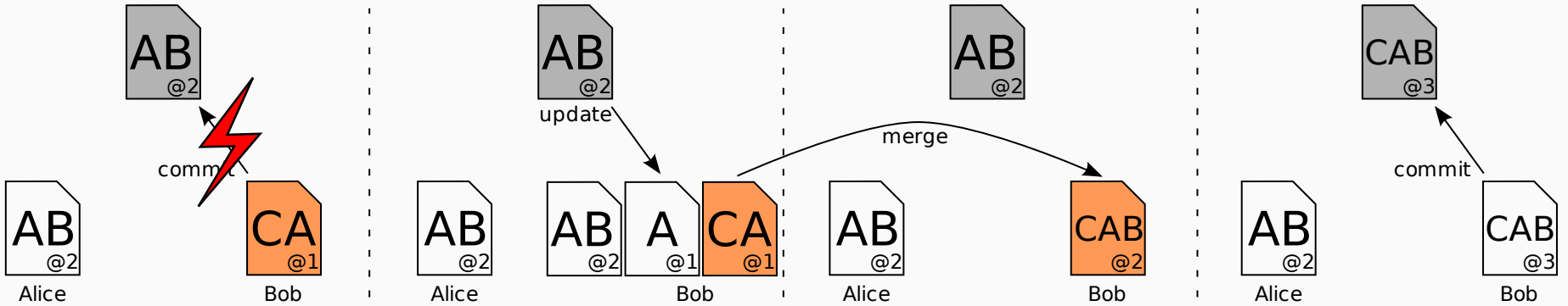
  - Modifikationen werden nicht erkannt
  - Änderungen von Alice gehen unbemerkt verloren

Warum Versionsverwaltung?

- Versionsnummer zur Erkennung von Modifikationen



- Entstandener Konflikt muss lokal gelöst werden



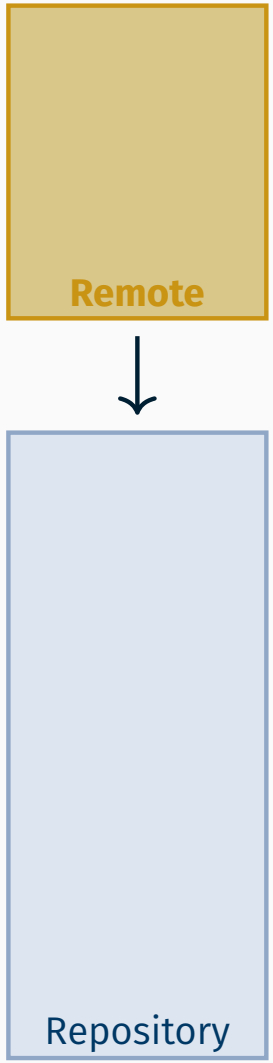
Das Versionsverwaltungssystem Git

- Kommando: `git`
- Speichert Zusatzinformationen zu jeder Änderung
  - Name des Ändernden
  - Zeitpunkt
  - Kommentar
  - ...

  $\Rightarrow$ identifiziert durch Commit-Hash
- Hilfe über Manpages (`man 1 git`) oder `git --help`
- SP-Abgabesystem verwendet Git

Git-Repository einrichten


```
~> git clone gitlab.cs.fau.de:i4sp/ss23/test.git
       beispiel
Cloning into 'beispiel'...
```



Git-Repository einrichten

bfb76ad

main

1

Remote

```
~> git clone gitlab.cs.fau.de:i4sp/ss23/test.git
       beispiel
Cloning into 'beispiel'...
~> cd beispiel
```

Workspace

Repository

Dateien mit GIT verwalten

bfb76ad

main

1

Remote

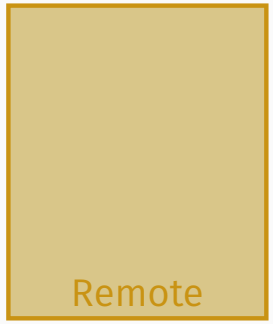
```
~/beispiel> touch README.md
```

README.md

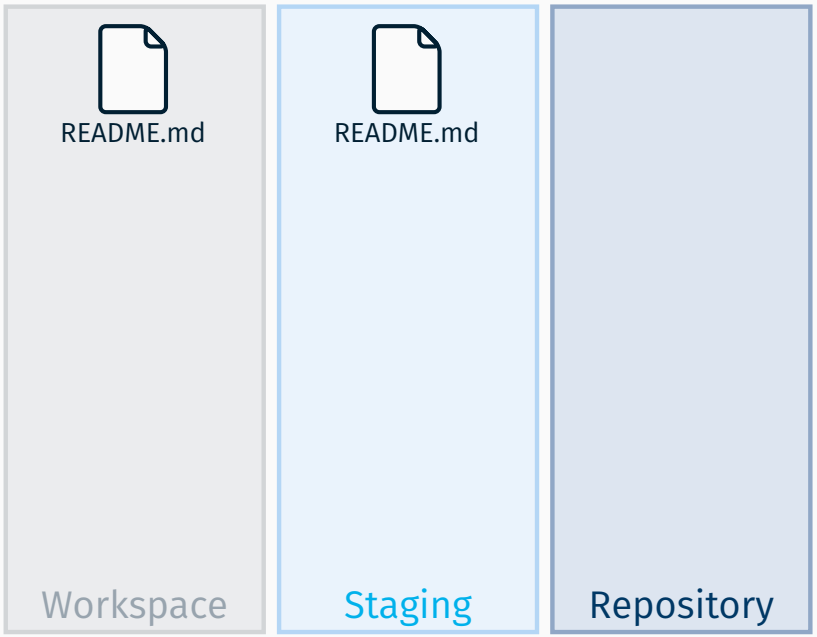
Workspace

Repository

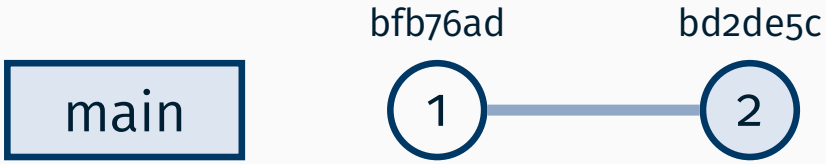
Dateien mit GIT verwalten

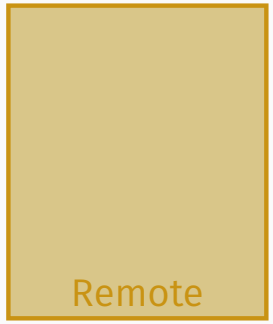



```
~/beispiel> touch README.md
~/beispiel> git add README.md
```

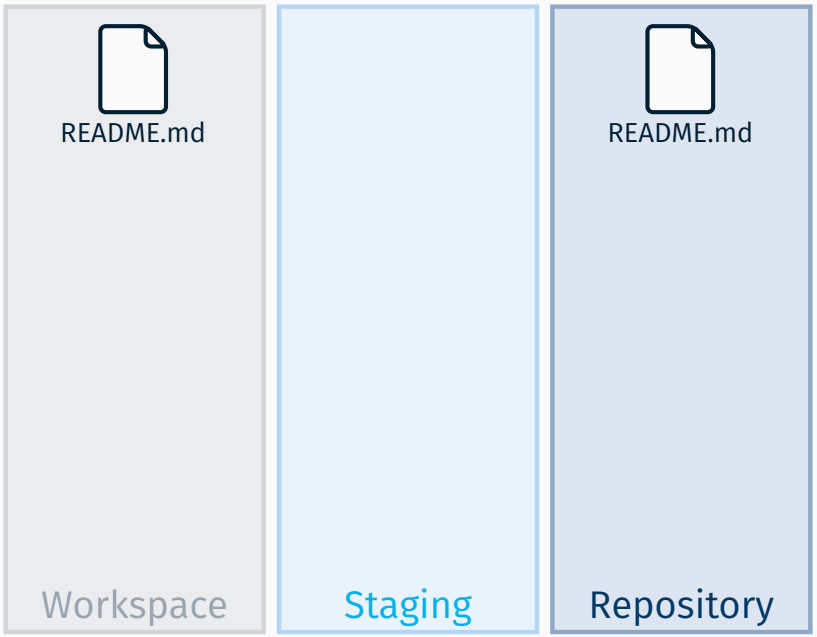


Dateien mit GIT verwalten

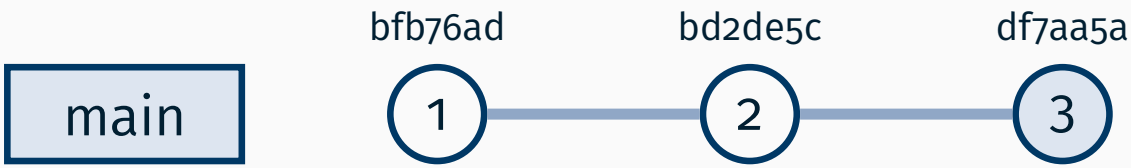


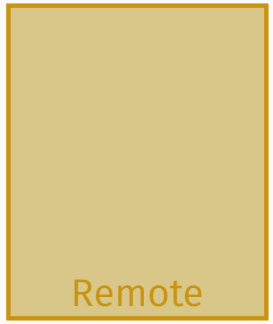



```
~/beispiel> touch README.md
~/beispiel> git add README.md
~/beispiel> git commit -m "Liesmich hinzugefügt"
[main bd2de5c] Liesmich hinzugefügt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
```

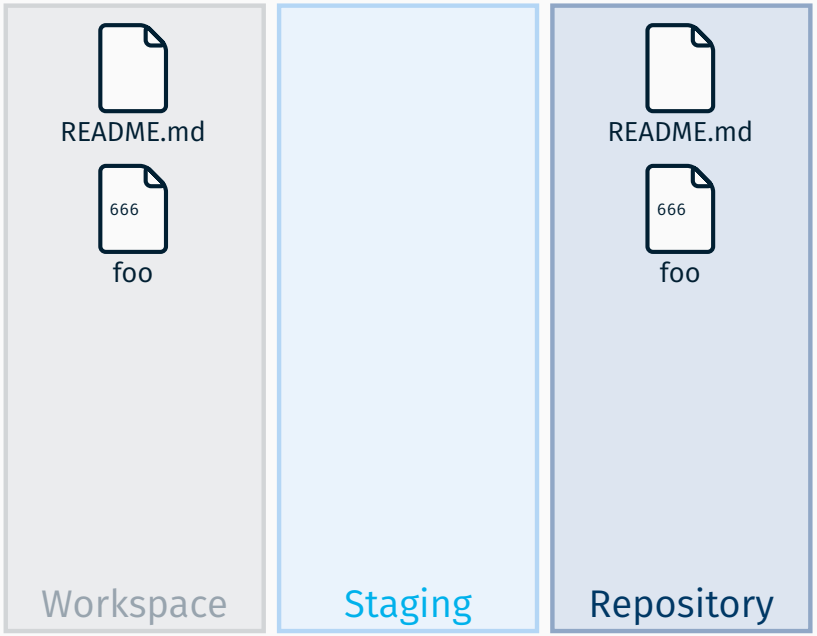


Dateien mit GIT verwalten

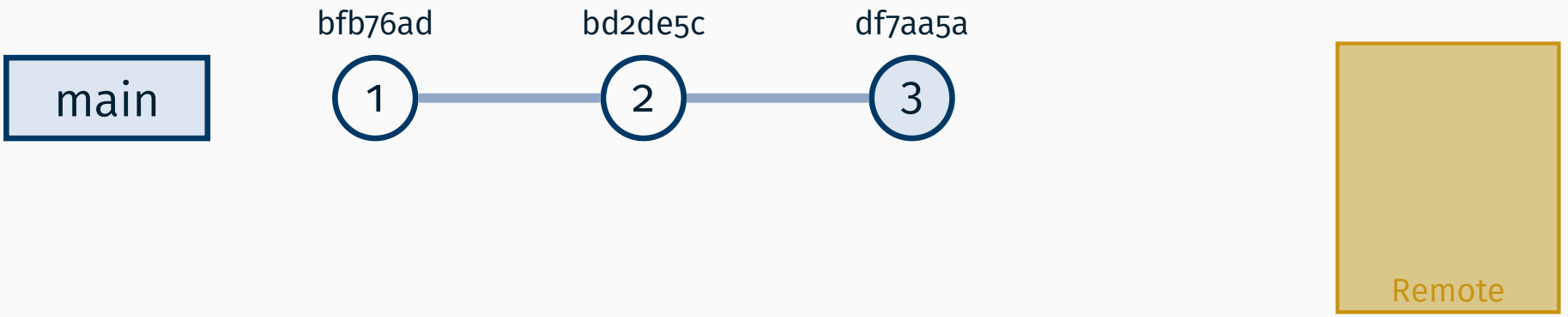




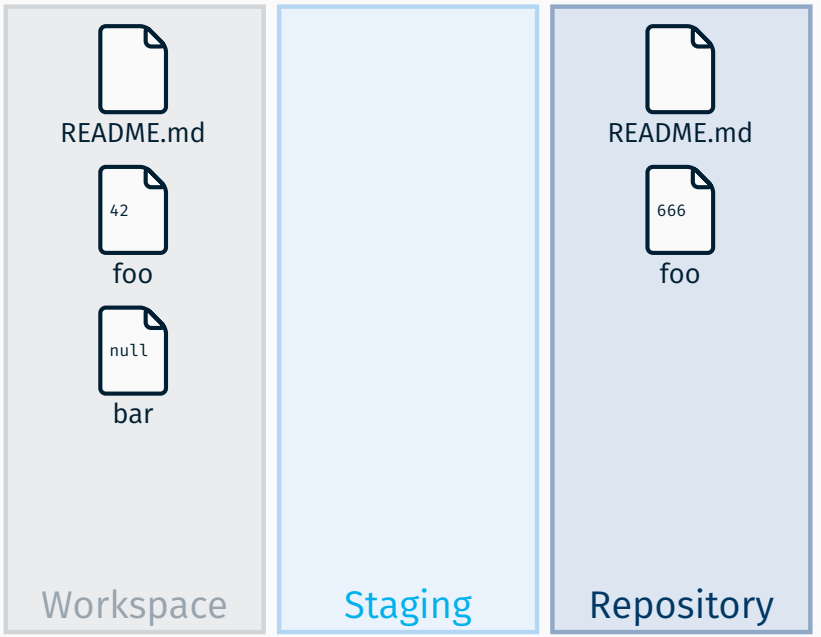

```
~/beispiel> echo "666" > foo
~/beispiel> git add foo
~/beispiel> git commit -m "Datei foo erstellt"
[main df7aa5a] Datei foo erstellt
 1 file changed, 1 insertion(+)
 create mode 100644 foo
```



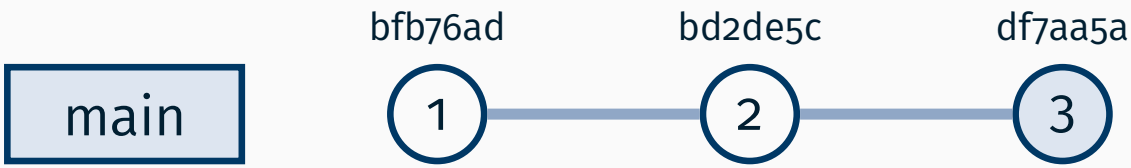
Dateien mit GIT verwalten

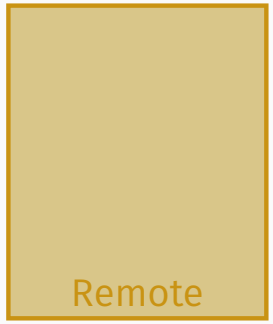



```
~/beispiel> echo "42" > foo
~/beispiel> echo "null" > bar
~/beispiel> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
     modified: foo

Untracked files:
     bar

no changes added to commit
```

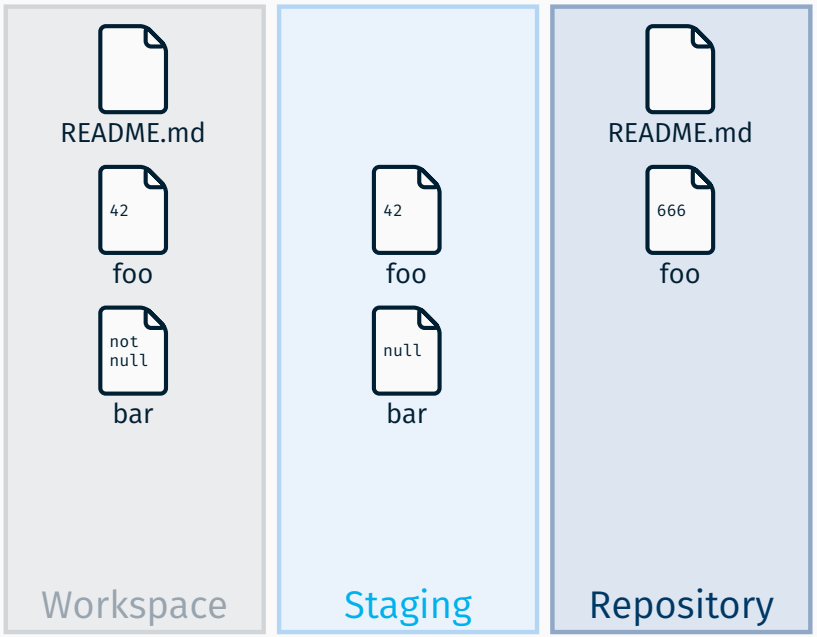


Dateien mit GIT verwalten

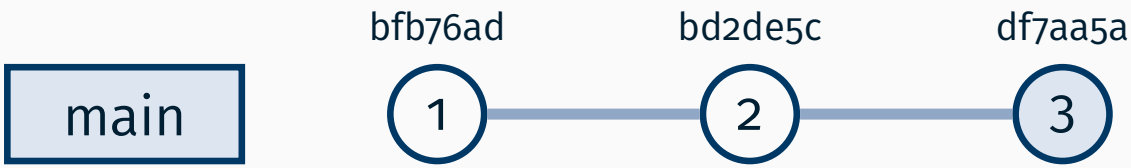


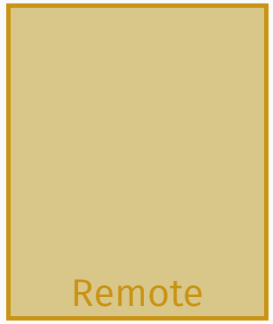



```
~/beispiel> git add foo bar
~/beispiel> echo "not null" > bar
~/beispiel> git status
On branch main
Your branch is up to date with 'origin/main'.

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

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

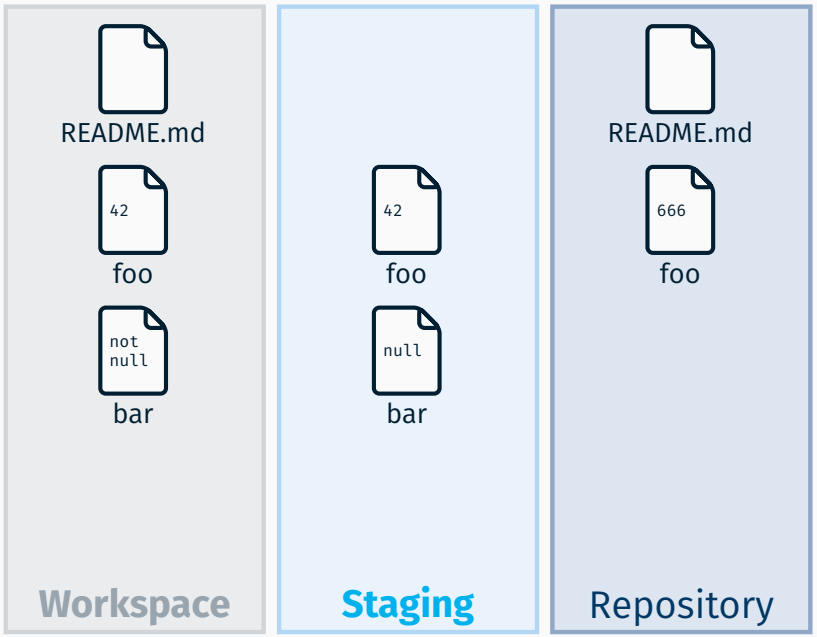


Dateien mit GIT verwalten

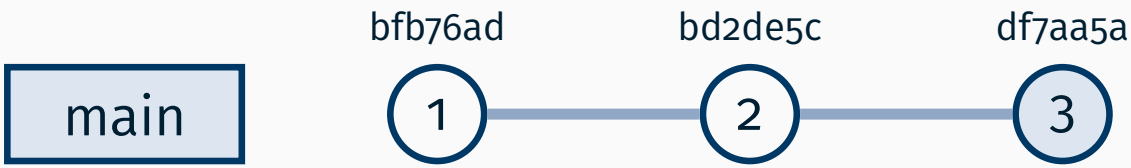


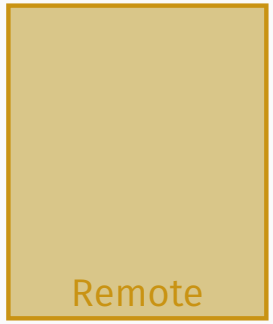



```
~/beispiel> git diff
diff –git a/bar b/bar
index 19765bd..b263a85 100644
– a/bar
+++ b/bar
@@ -1 +1 @@
-null
+not null
```

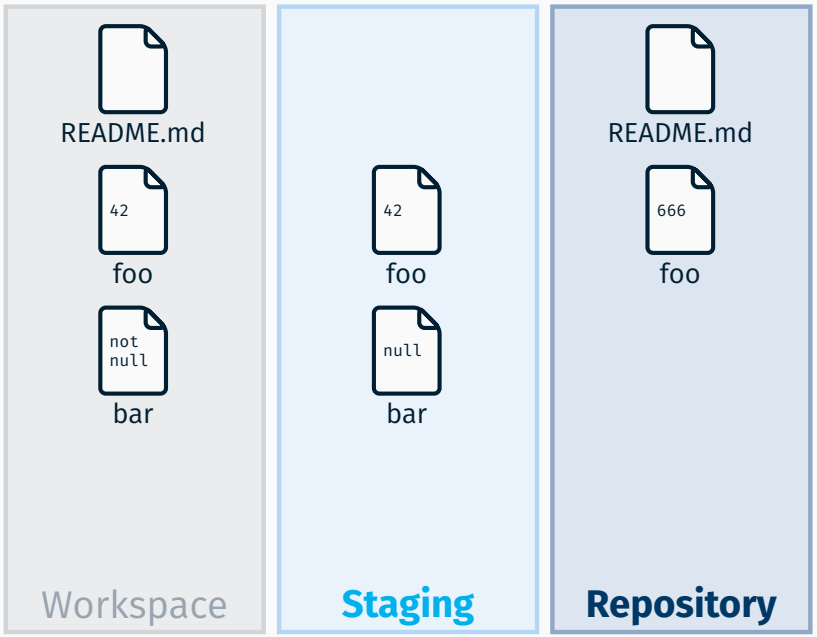


Dateien mit GIT verwalten

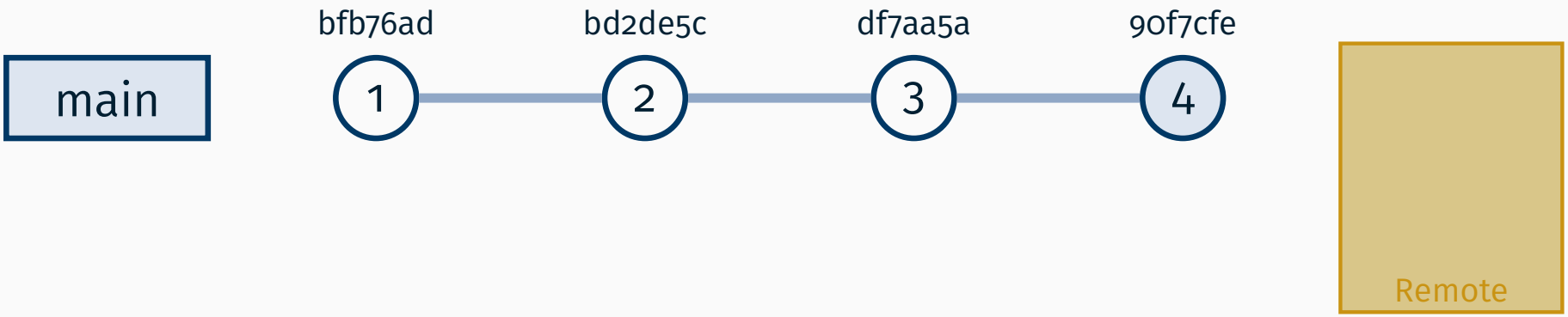




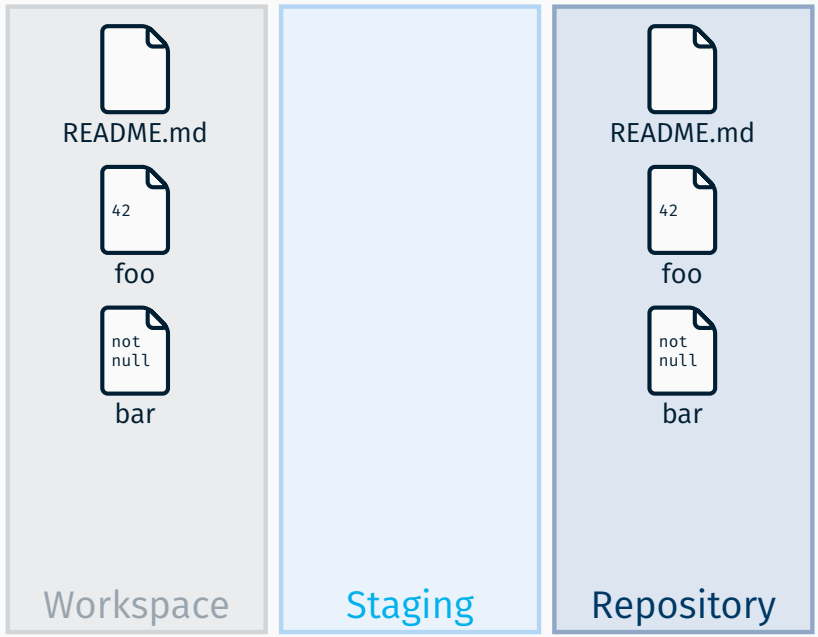

```
~/beispiel> git diff --staged
diff –git a/bar b/bar
new file mode 100644
index 0000000..19765bd
– a/bar
+++ b/bar
@@ -0,0 +1 @@
+null
diff –git a/foo b/foo
index 7cc86ad..d81cc07 100644
[...]
```



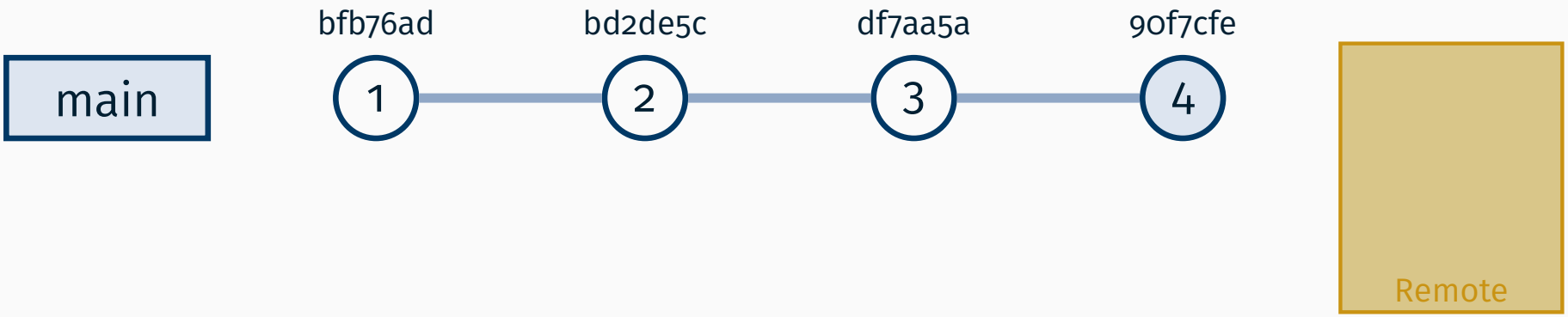
Dateien mit GIT verwalten



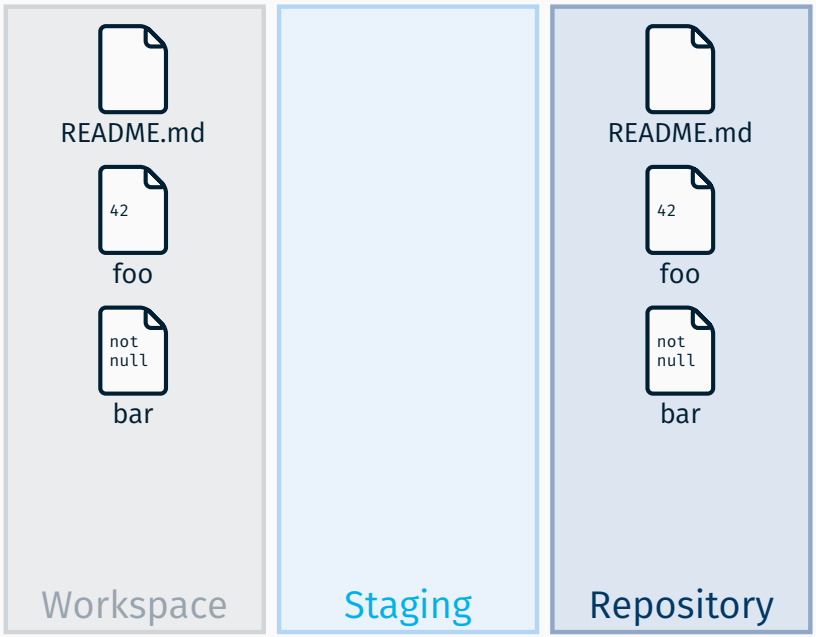

```
~/beispiel> git add bar
~/beispiel> git commit -m \
            "Foo korrigiert und Bar erstellt"
[main 90f7cfe] Foo korrigiert und Bar erstellt
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 bar
```



Dateien mit GIT verwalten

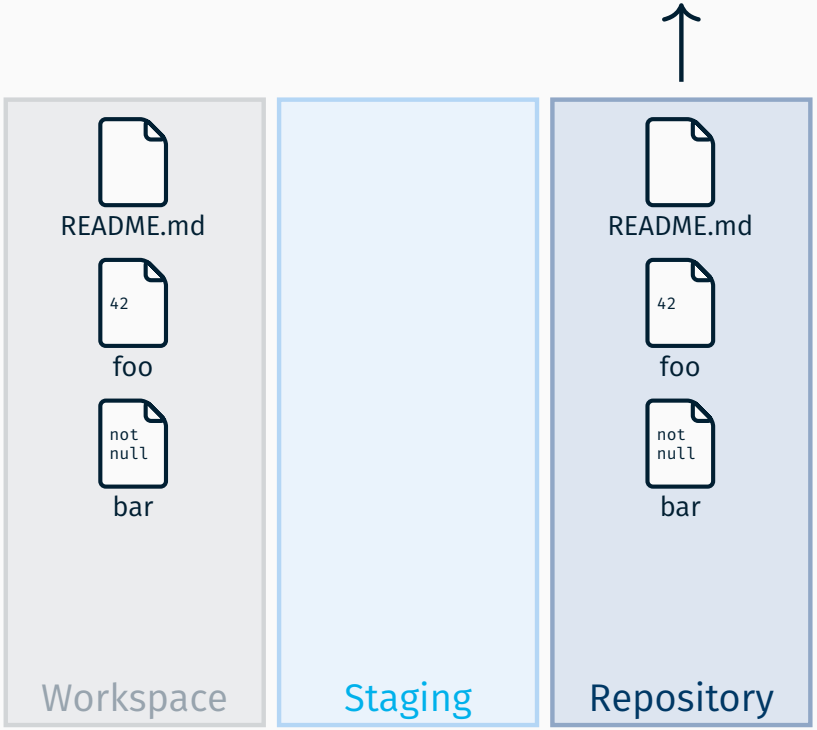



```
~/beispiel> git shortlog
SP-Nutzer (3):
     Liesmich hinzugefügt
     Datei foo erstellt
     Foo korrigiert und Bar erstellt
```

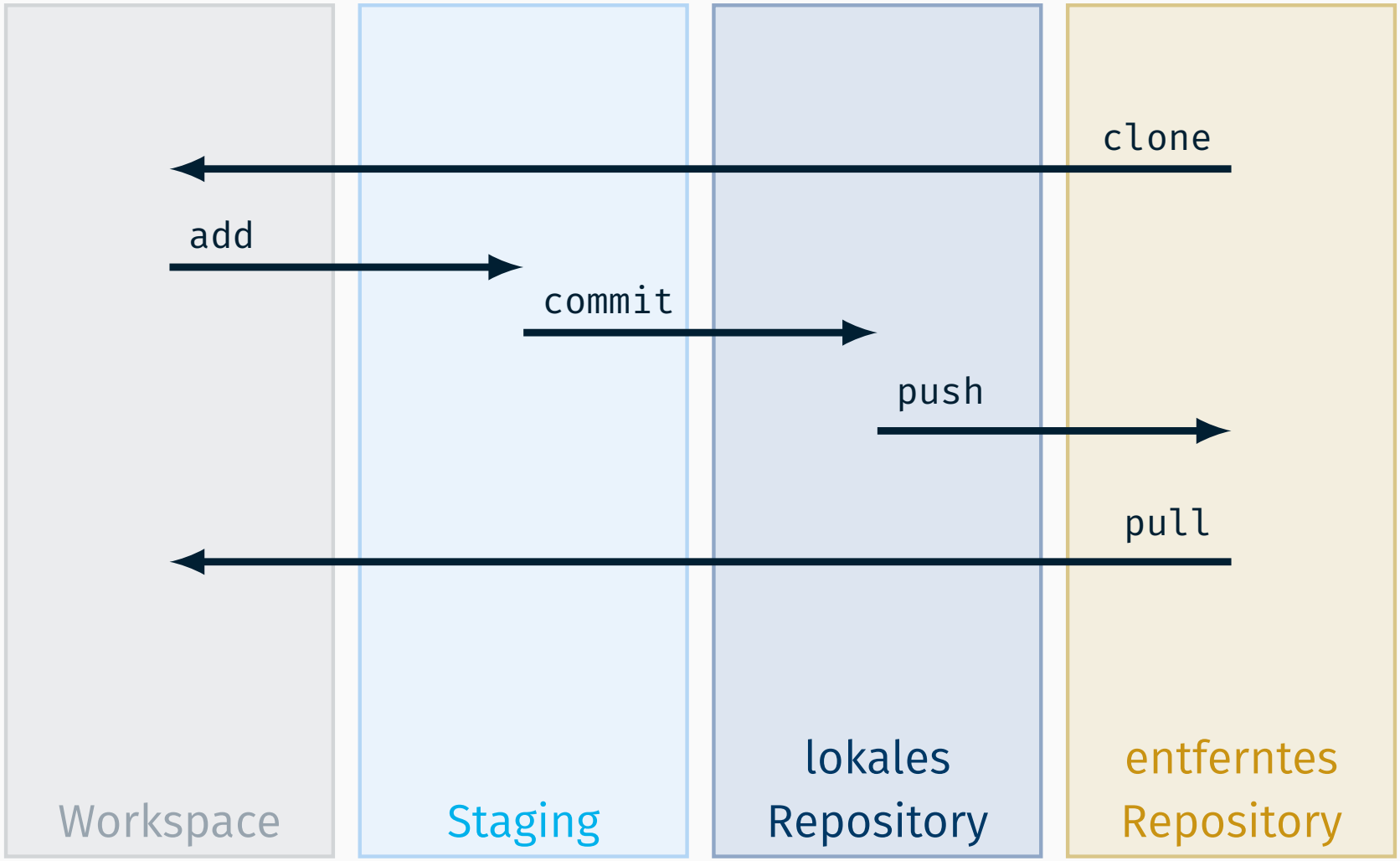


Dateien mit GIT verwalten


```
~/beispiel> git push
Enumerating objects: ...
    ...
To gitlab.cs.fau.de:i4sp/ss23/test.git
   bfb76ad..90f7cfe main -> main
```



Überblick: Dateien mit GIT verwalten



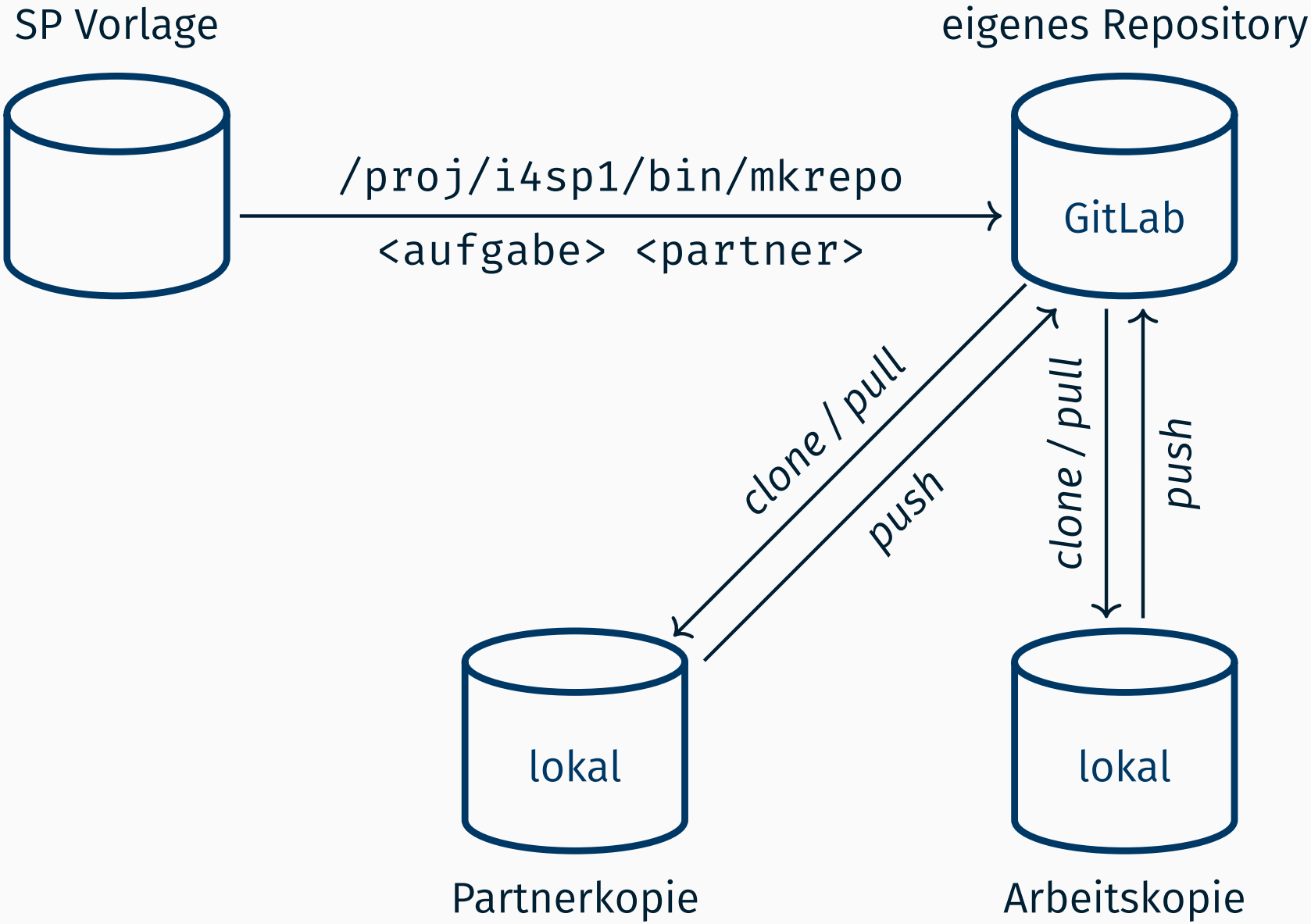
Cheatsheet

git add <*file*>	Datei als Kandidat für nächsten *commit* markieren
git commit	Änderungen versionieren
git diff	unversionierte Änderungen anzeigen
git show	neuste (versionierte) Änderungen anzeigen
git status	Änderungen zum Vorgänger anzeigen
git log	Historie anzeigen
git clone <*url*>	initiales Kopieren von einer Quelle
git pull	kurz für holen und zusammenfügen
git push	in entfernte Quelle übertragen

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

Agenda

Arbeitsablauf in SP



SP-Abgabesystem

- Für jede Aufgabe muss sich jede Übungsgruppe ein neues Vorgabe-Repository erstellen:
  `/proj/i4sp1/bin/mkrepo <aufgabe> [<partner>]`
  - `aufgabe`: aktuell zu bearbeitende Aufgabe
  - bei Gruppenabgaben: Nutzerkennung des `partners`
- Repository enthält Vorgabe
  (z.B. Programmgerüste und Beispieleingaben)
- Repository-Link:
  `https://gitlab.cs.fau.de/i4sp/ss23/<TUEB>/<user>/<aufgabe>`
- Nutzung von Git zum Erstellen von lokalen Arbeitskopien

SP-Abgabesystem

- Zum Abgabezeitpunkt wird der neueste Commit im Repository auf `https://gitlab.cs.fau.de/i4sp/ss23/...` eingesammelt
  - von dem Hauptbranch (`main`)
  - dieser Commit ist Abgabe der Übungsaufgabe
  - zu bewertende Änderungen stets mit **push** verbreiten
  - anderenfalls **keine Bewertung!**
  - bis zum Abgabezeitpunkt kann beliebig oft aktualisiert werden
- **Eigener** Abgabetermin kann per Skript erfragt werden

```
> /proj/i4sp1/bin/get-deadline aufgabe1
Dein Abgabezeitpunkt fuer die Aufgabe 1: lilo ist 01.01.1970
um 17:30:00 Uhr
```

Beispiel-Workflow für Aufgabe 1

```
# Repository anlegen
student@cip ~ > /proj/i4sp1/bin/mkrepo lilo
...
# Lokale Kopie des Repositories anlegen
student@cip ~ > git clone https://gitlab.cs.fau.de/i4sp/...
...
# Bearbeiten der Dateien
student@cip ~ > cd lilo
student@cip ~/lilo > nano lilo.c
...
# Hinzufügen der Änderungen
student@cip ~/lilo > git add lilo.c
student@cip ~/lilo > git commit -m "add lilo.c"
...
# weitere Änderungen
student@cip ~/lilo > vim lilo.c
...
student@cip ~/lilo > git add lilo.c
student@cip ~/lilo > git commit -m "bugfix in printf"
...
# Aktualisieren der Abgabe
student@cip ~/lilo > git push
...
```