Git

Eine performante verteilte Versionsverwaltung (Revision/Version Control System)

Hartmut Ruhl

30.04.2020

Übersicht

Namensgebung

Woher kommt der Name GIT?

Dazu finden sich Informationen unter
`https://de.wikipedia.org/wiki/Git#Name`.

Was ist eine Versionskontrolle

Ein System das die Dateien kontinuierlich überwacht

- Überwachen bedeutet, dass es sämtliche Änderungen erkennt und darüber Buch führt.
- Man kann das System jederzeit eine Momentaufnahme (Snapshot/**Commit**) erstellen lassen (Einen Sicherungspunkt/Momentaufnahme des aktuellen Status).
- Andere können ebenfalls Änderungen durchführen und diese werden ebenfalls protokolliert.

Weitere Vorzüge

- Jeder Commit wird mit Zeitstempel und Autor-Informationen gespeichert (dadurch lassen sich z.B. Änderungen leicht zurück verfolgen).
- Änderungen lassen sich leicht jederzeit rückgängig machen (revert).
- Es ist sogar möglich zu einer früheren Revision zurück zu gehen, einen Fehler zu korrigieren und sämtliche nachfolgenden guten Änderungen wieder automatisch anwenden zu lassen !

Gründe für eine Versionskontrolle

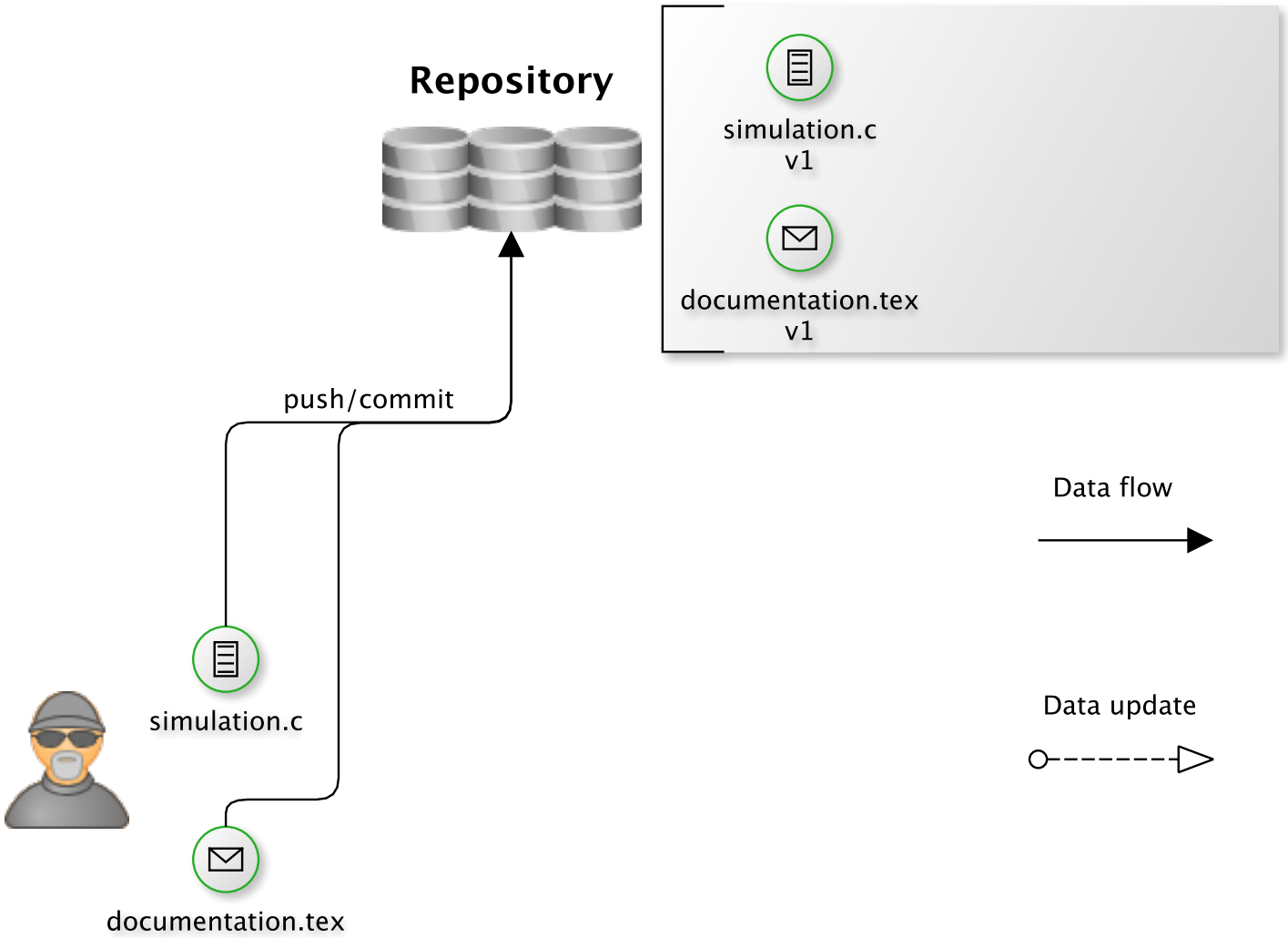
Einzelner Benutzer:

- Vereinfacht die Entwicklung digitaler Produkte (Texte, Programme, ...).
- Insbesondere bei Software-Projekten.

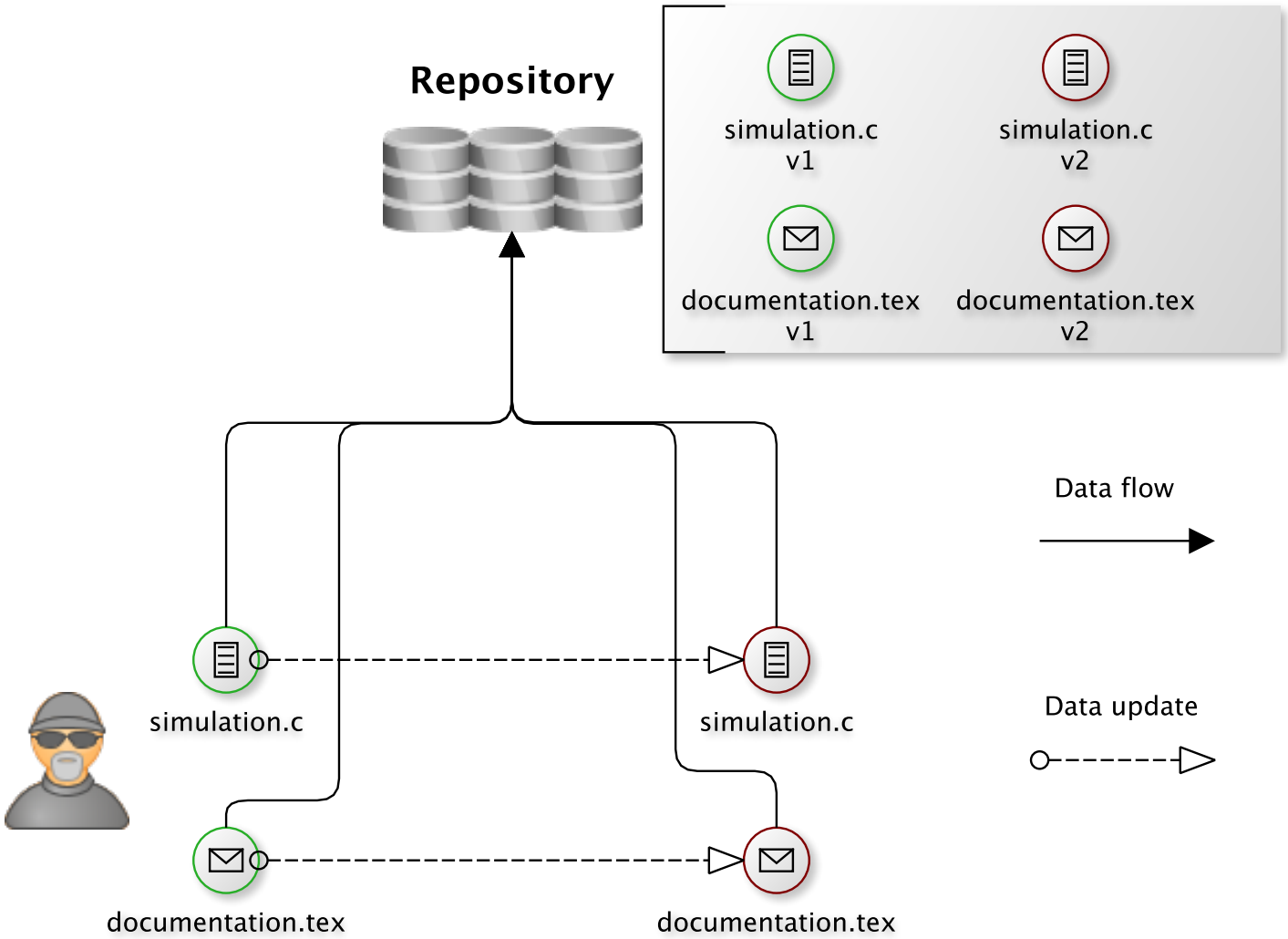
Im Team:

- Durch die Protokollierung ist es OK, dass alle Involvierten ohne Rücksprache Änderungen durchführen können.
- Alle (sinnvollen) Änderungen aller Beteiligten können (weitgehend) automatisch in einen finales Ergebnis zusammengeführt (“**merge**”) werden.
- Sofern es keine Überschneidungen/Konflikte gibt !
- Unterschiedliche Zweige (Branches) sind möglich, was es erlaubt sich zeitweise auf ein spezifisches Teilproblem zu fokussieren. Das Ergebnis kann später problemlos in die Hauptentwicklungslinie (“**master branch**”) “gemerged” werden.
- Sofern es keine Überschneidungen/Konflikte gibt !

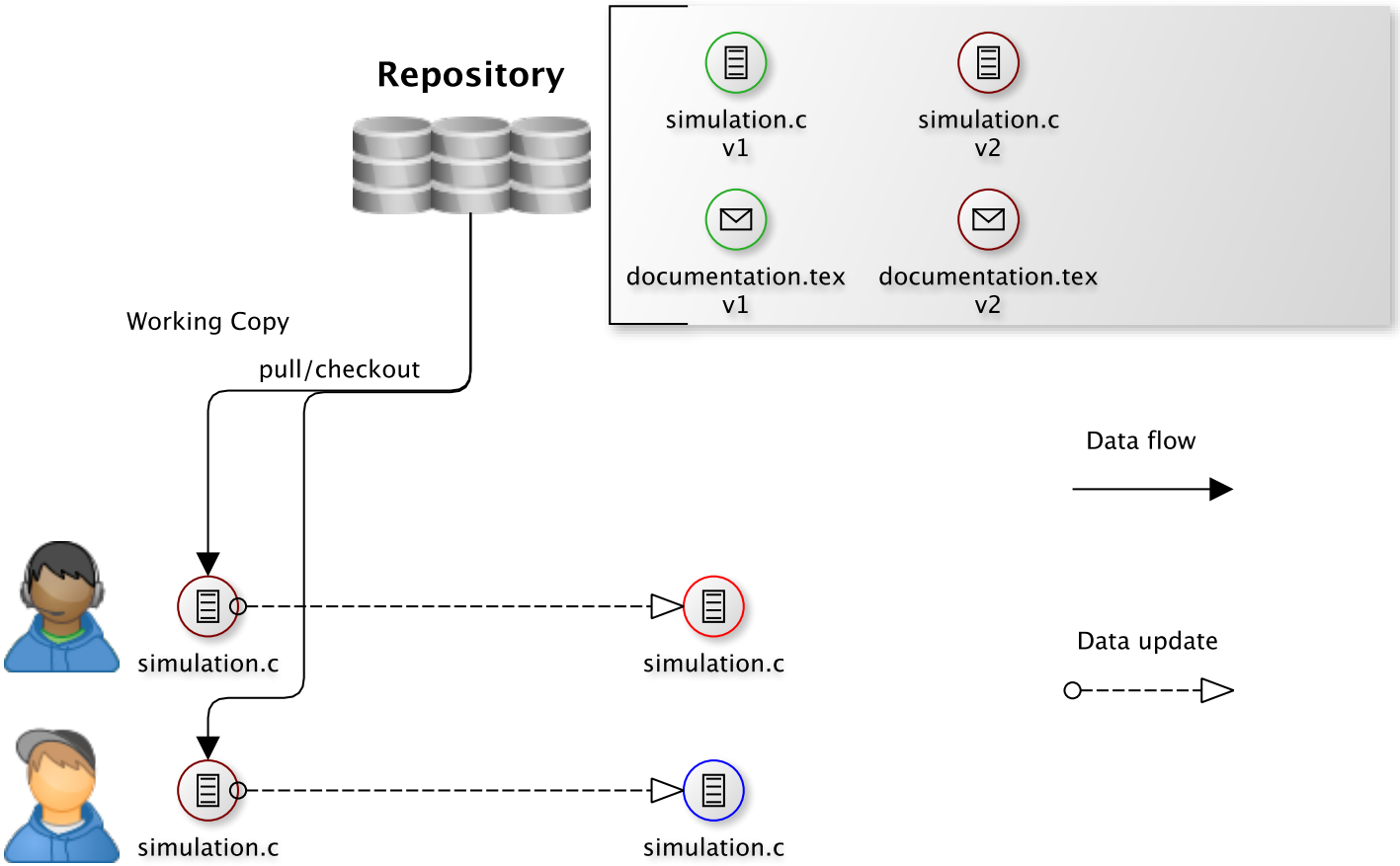
Arbeitsablauf - Commit



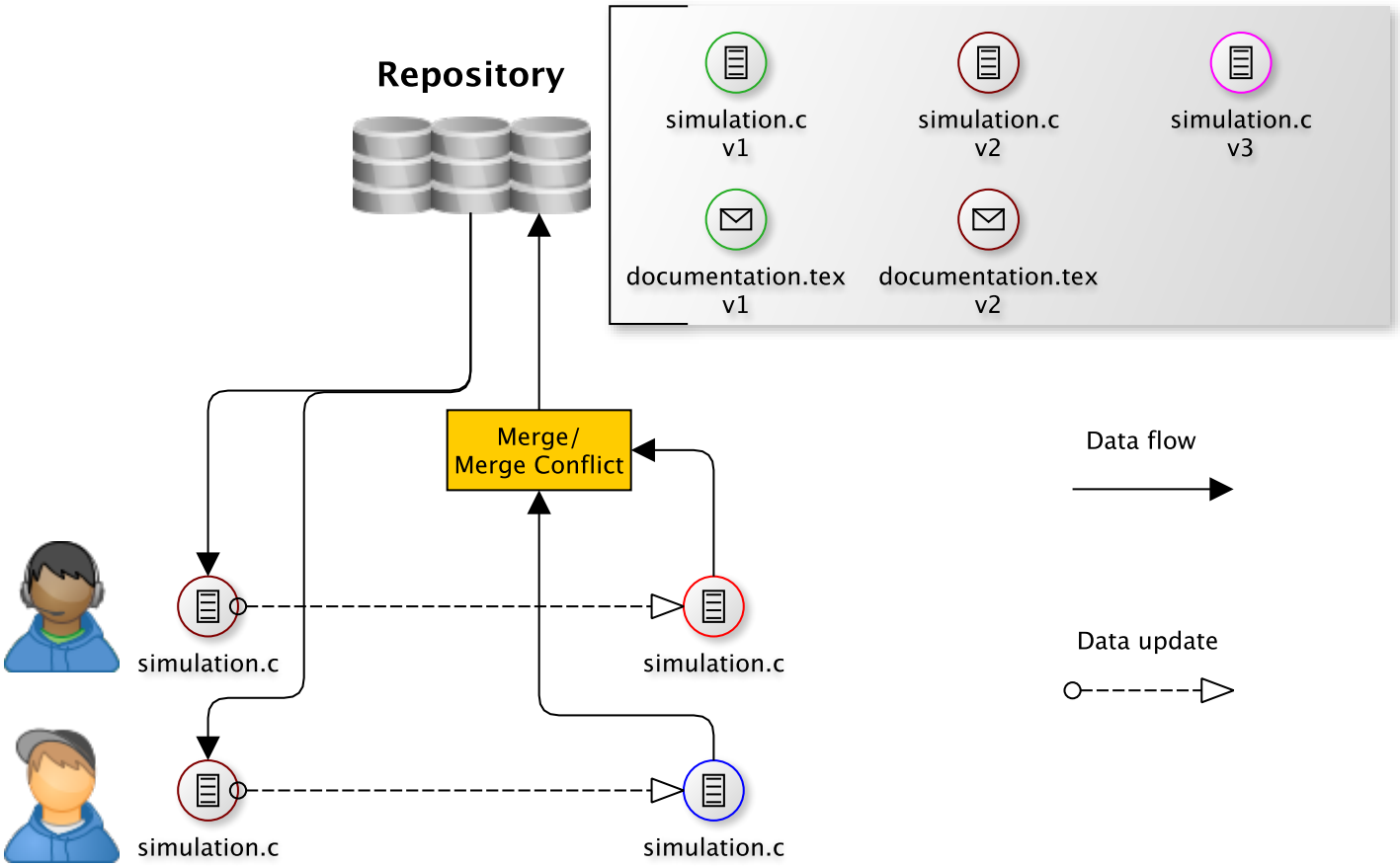
Arbeitsablauf - Commit



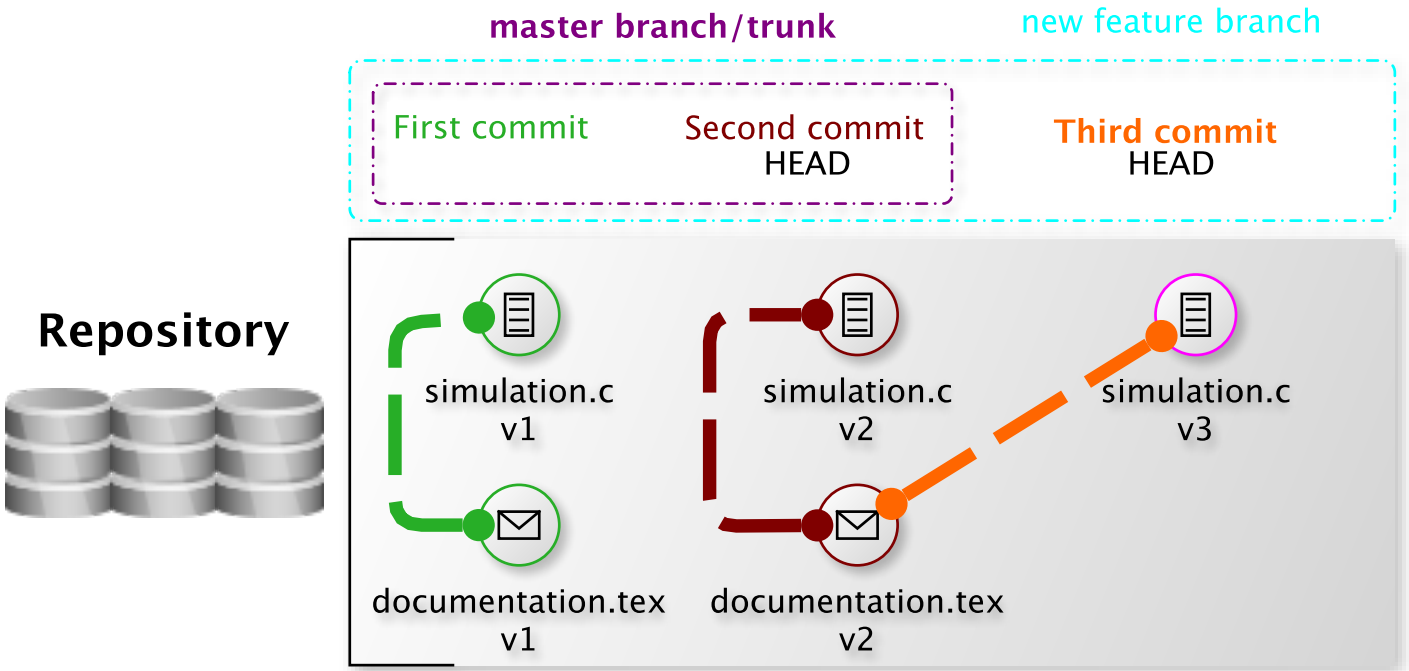
Arbeitsablauf - Checkout



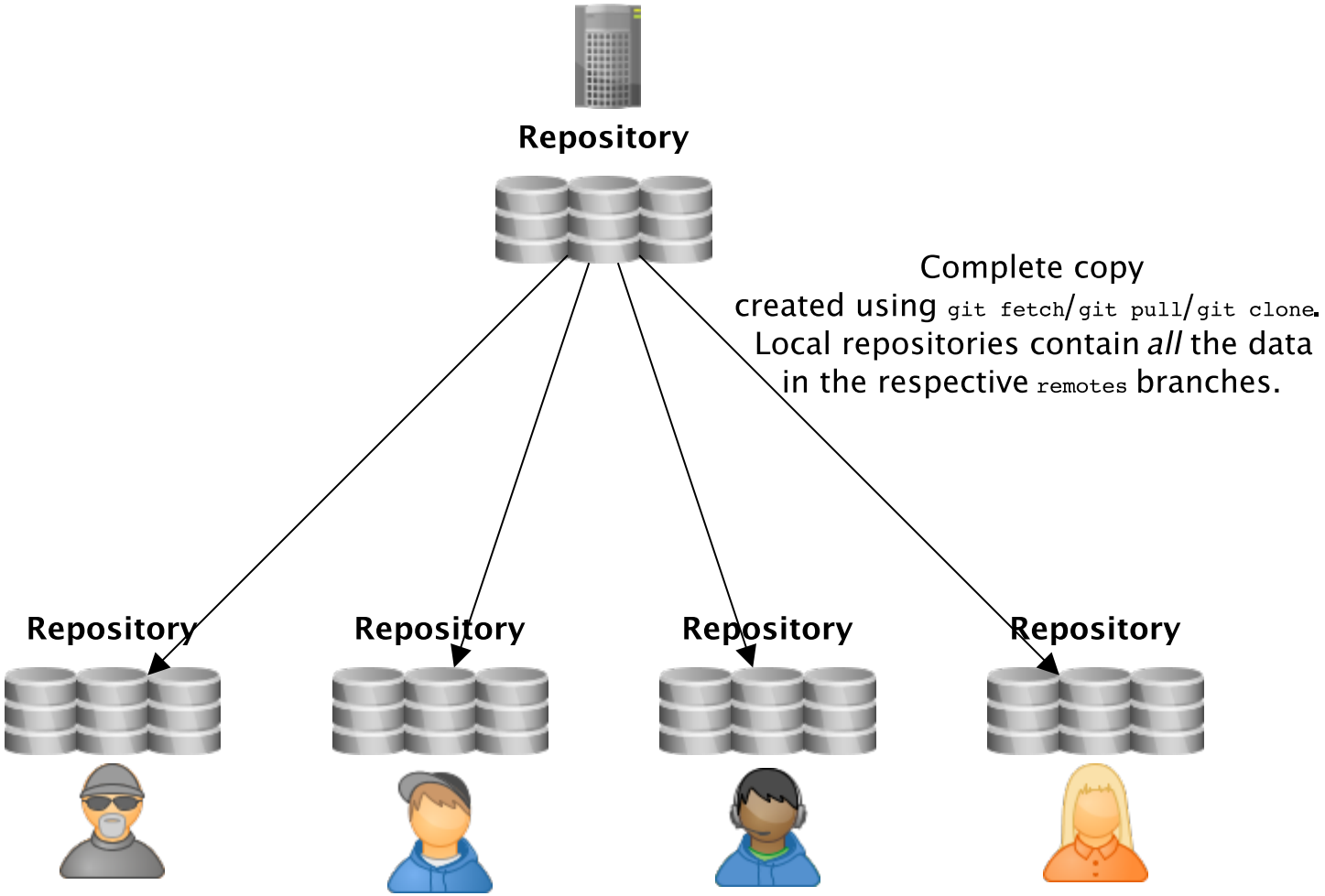
Arbeitsablauf - Merge



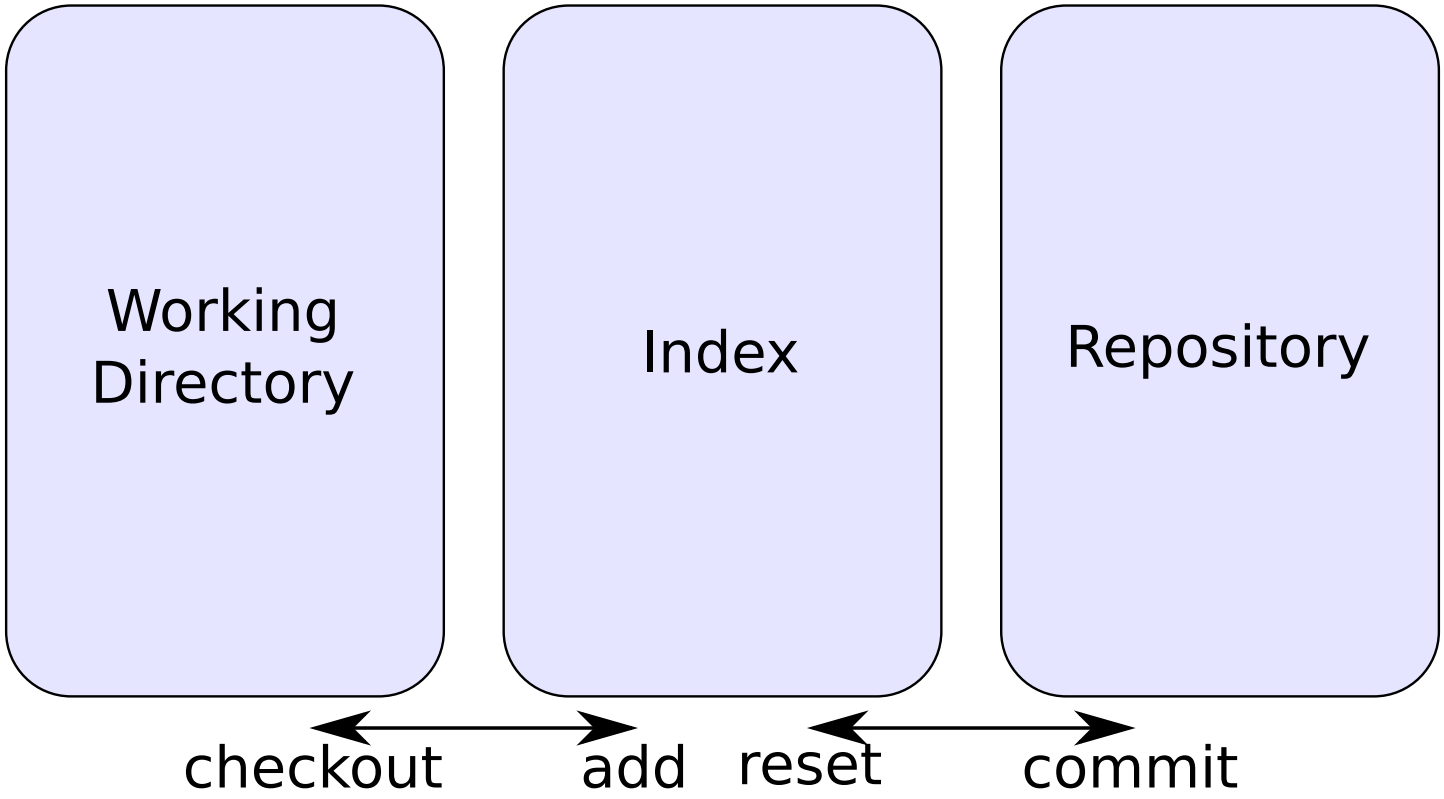
Arbeitsablauf - Branches



Git ist ein verteiltes Versionsverwaltungssystem



Arbeitskopie/ Working Copy - Index - Repository



Zweige/Branches

Wozu dienen **Branches**?

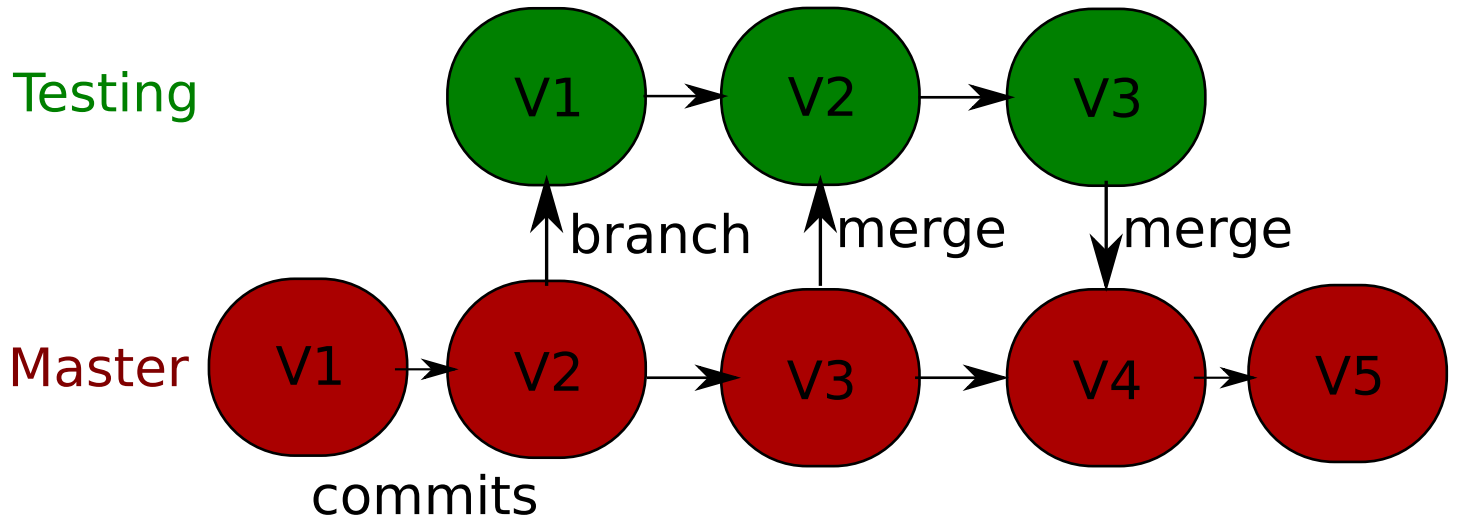
- Sie können verschiedene Versionen der gleichen Gesamtfunktionalität (Überschneidungen) darstellen.
- z.B. für experimentelle Änderungen.
- Oder sie beinhalten parallele Arbeiten mehrerer Entwickler am gleichen Projektstamm (trunk).

Welche branches existieren und warum:

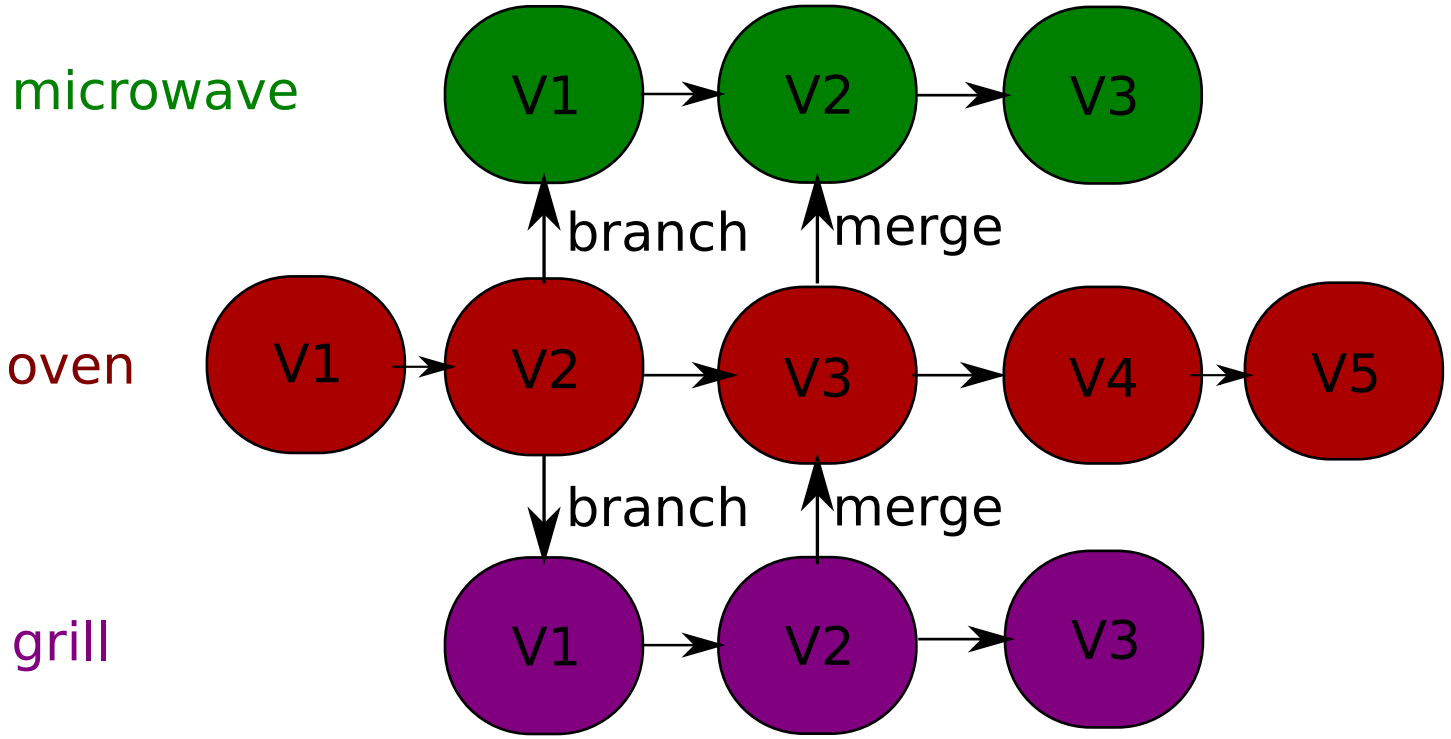
`git branch -a`

- Git hält von allen Branches in entfernten Repositories Informationen lokal vor. Solchen "entfernten Branches" ist ein `remotes/` vorangestellt. Mit dem Kommando `git fetch` werden diese lokalen Kopien aktualisiert. `git fetch` macht Änderungen eines anderen (remote) Repositories im eigenen Repo verfügbar.
- Man kann eine Referenz auf ein entferntes Repository mit dem Kommando `git remote add` hinzufügen und ein alias dafür setzen (wie z.B. origin bei geklonten Repositories).
- Branches die mit `remotes/` markiert sind, lassen sich nicht direkt verwenden. Man muss eine lokale Kopie erstellen, die dem remote branch folgt (tracked).

Branches/Zweige - Workflow Beispiel
Neue Features & Mehrere Entwickler



Branches/Zweige - Workflow Beispiel
Verschiedene Versionen



Git aufsetzen

Einrichtung von Git

- `git config --global user.name "Bob"`
- `git config --global user.email "bob@bob.com"`

Aufsetzen eines lokalen Repositories

```
git init
```

Kopieren (Klonen) eines bereits existierenden Gits

- `git clone Git-URL`
- Mögliche Git-URLs (nur HTTP(S), lokal und SSH):

HTTP(S)	https://gitlab.physik.uni-muenchen.de/path/to/repo.git
Lokal	/path/to/repo.git/
SSH	ssh://[user@]host.xz[:port]/path/to/repo.git/
	(erfordert auf GitLAB die Einrichtung von SSH-Keys, die nicht Teil der Vorlesung sind)

Alltägliche Arbeit mit Git

`git checkout`

- Auschecken": eines Zweigs/Branch: Setzt das Repo auf den letzten Commit des Branch (genannt HEAD) und damit den Index und die Arbeitskopie (working directory) auf den Inhalt des gewünschten Branch.
- Wichtiges Kommandozeilen-Argument:
  - `-b` Erzeugt einen neuen Branch und wechselt direkt zum neu erzeugten Branch

`git add`

Fügt lokale Änderungen zum Index (auch Staging Bereich genannt) hinzu, Datei für Datei.

`git diff`

Zeigt die Unterschiede zwischen der Arbeitskopie und dem Index (oder auch zwischen einzelnen Commits).

`git status`

Zeigt den aktuellen Status der Arbeitskopie, des Index und des Repositories

Alltägliche Arbeit mit Git

`git commit`

- Verschiebt die im Index vorgemerkten Änderungen in das Repo mit einem neuen Sicherungspunkt (Commit).
- Wichtige Kommandozeilen-Argumente:
  - `-a` Fügt für den Commit noch alle in der Arbeitskopie gemachten Änderungen zum Index hinzu.
  - `-m` Gefolgt von einem Text in Anführungszeichen, setzt diesen als Commit Message.

`git branch`

- Erzeugt einen neuen Branch oder zeigt die bereits Vorhandenen an.
- Wichtige Kommandozeilen-Argumente:
  - `-a` Zeigt alle existierenden Branches
  - `-d` Löscht einen Branch nur wenn alle Änderungen bereits in andere Branches übernommen (merge) wurden.
  - `-D` Löscht einen Branch" ohne Rücksicht auf Verluste.
- Beispiel: `git branch my_testing remotes/origin/testing`
  Erzeugt einen neuen Branch namens 'my_testing' der dem Branch 'testing' auf dem Remote-Repo 'origin' folgt.

Alltägliche Arbeit mit Git

`git merge`

- Wird normalerweise benutzt, um die Änderungen von Zweigen, die ihren Zweck erfüllt haben, in die aktuelle Hauptentwicklungslinie zu übernehmen. Um das zu erreichen, checken sie die Hauptentwicklungslinie aus und rufen folgenden Befehl auf: `git merge soon-to-be-abandoned-branch`
- Falls es 'Konflikte' gibt - es existieren auf beiden Zweigen Änderungen an der gleichen Stelle der gleichen Datei - erzeugt Git Markierungen dafür die folgendermaßen aussehen:

```
<<<<<<< master
Content of line x in branch master
=======
Content of line x in branch local
>>>>>>> local
```

  Diese müssen aufgelöst werden und das Ergebnis in einem Commit festgehalten werden!

Alltägliche Arbeit mit Git

`gitk`

Zeigt eine (schöne) grafische Ansicht des Repos, der Zweige und deren Zusammenhänge. Eine gute Alternative für Windows und Mac OS X wäre z.B. ’Source Tree’ (erfordert eine kostenlose Registrierung bei Atlassian)

Arbeiten im Team

`git pull`

pull führt eigentlich zwei Kommandos auf einmal aus: `git fetch` und `git merge`

- Das erste Argument kann eine Repo-URL sein. Sie wird für das fetch, also das Holen von Branch-Informationen benutzt. Das Ergebnis wird typischerweise lokal als ‘remotes/remote-alias/’ Branch gespeichert.
- Das zweite Argument ist der Branch auf dem Remote-Repo der für das merge benutzt werden soll, also der Branch, dessen Änderungen in den aktuell benutzten Branch übernommen werden sollen.
- Wenn tracking für den aktuellen Branch konfiguriert ist, also hinterlegt ist, welchem Branch der aktuelle Branch folgen soll, kann `git pull` ohne Argumente aufgerufen werden.
- Das Kommando arbeitet immer auf dem aktuell verwendeten Branch!
- Da ein merge enthalten ist, können natürlich Merge-Konflikte auftreten.

Arbeiten im Team

`git push`

- Versucht einen entfernten Branch (auf einem Remote-Repo) zu aktualisieren, also die Commits im aktuellen Branch dort hin zu kopieren.
- Funktioniert für gewöhnlich nur mit fast forwards (keine Konflikte) und das hat gute Gründe !!
- Ein push kann mit entsprechenden Rechten erzwungen werden $\rightarrow$ überschreibt ohne Rücksicht auf Verluste.
- Normales Vorgehen, wenn ein push nicht funktioniert:
  pull $\rightarrow$ alle Konflikte auflösen $\rightarrow$ neuen Commit pushen.
  Dieser ist definitiv ein Nachfolger des aktuellen Head auf dem Remote-Branch.
- Konflikte lokal lösen $\rightarrow$ konfliktfreies Remote mit unveränderbarer Geschichte $\rightarrow$ erspart anderen Team-Mitgliedern Probleme.

Arbeiten im Team

`git push`

- Nettes Feature: Hooks
  Im wesentlichen sind dies Programme/Skripte/Aktionen, deren Aufruf an bestimmte Ereignisse, z.B. ein push, gebunden ist.
  `https://www.atlassian.com/git/tutorials/git-hooks`
  Hooks machen z.B. Sinn, um Team-Mitglieder automatisch zu informieren, oder automatisierte Tests für eine neue Programm-Code-Version laufen zu lassen.

Fortgeschrittene Kommandos

`git rebase`

- Umschreiben der Geschichte
- Es können Commits zusammen gefasst werden, da gilt commit early, commit often.
- Es kann eine lineare History erzeugt werden, die die Wirren temporärer lokaler Test-Branches aufräumt.
- $\rightarrow$ Einfaches aufgeräumtes Remote.

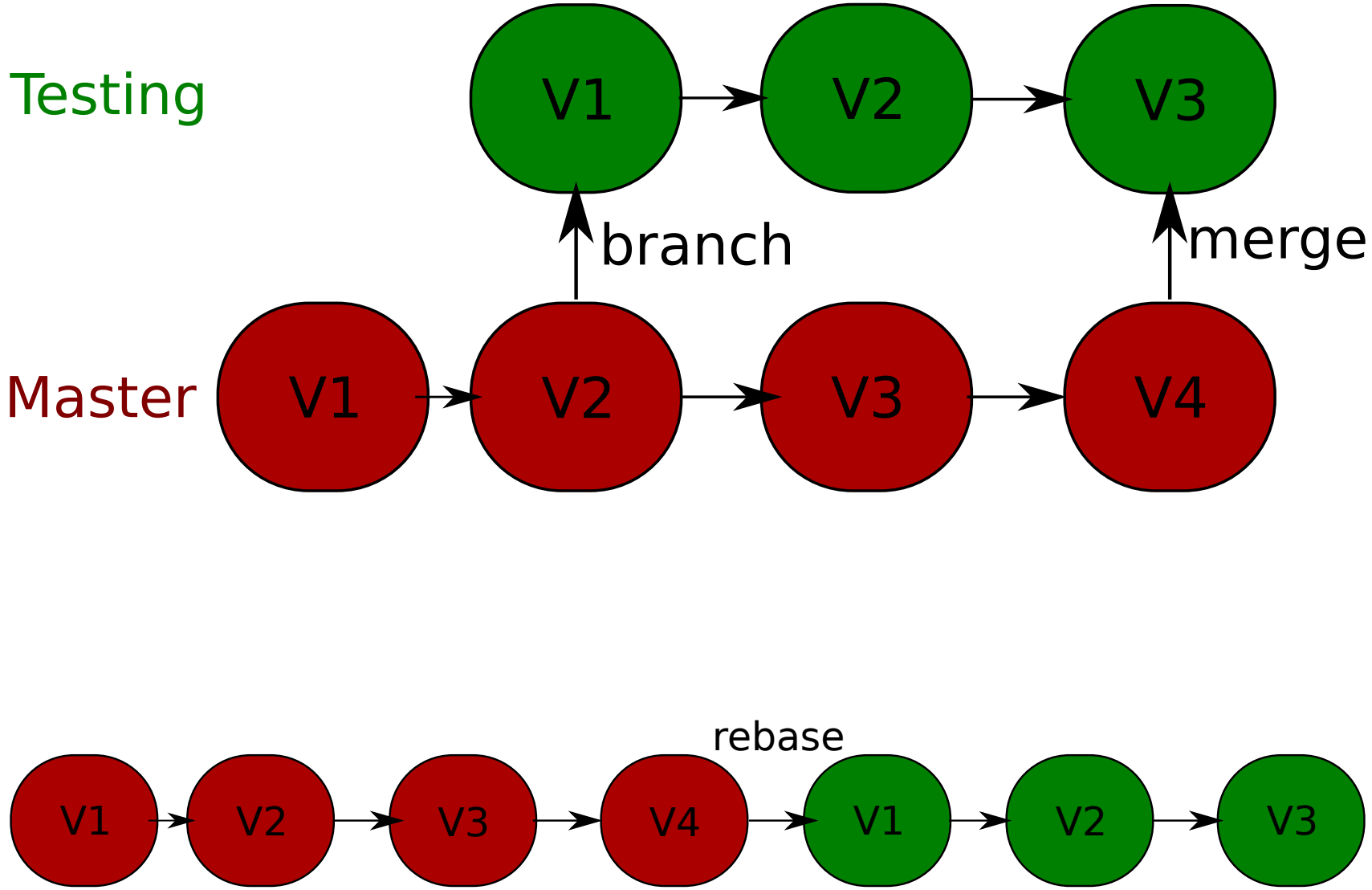
`git grep`

- Sehr gute Suchfunktion innerhalb eines Repos (ähnliche Syntax wie 'grep' in Unix).

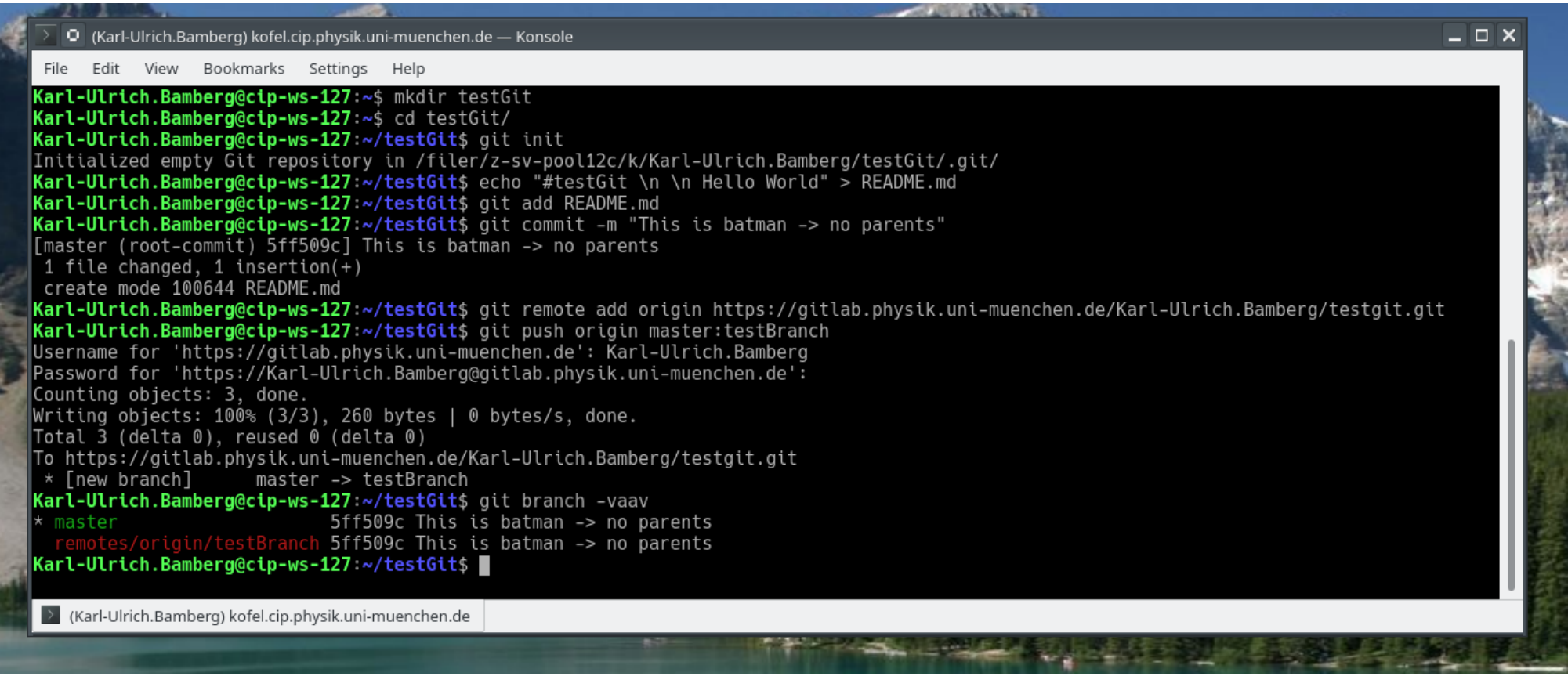
`git stash`

- Verschiebt alle aktuellen Änderungen der Arbeitskopie an einen sicheren Ort und setzt den Index und die Arbeitskopie auf den letzten Commit zurück.
- Nützlich, um mal eben in einen anderen Branch zu schauen, (checkout) was i.d.R. eine saubere Arbeitskopie erfordert.

Fortgeschrittene Kommandos - rebase



Ein Beispiel mit dem Command Line Interface von Git



Zugang zum Git der Vorlesung

Zugang zum Git der Vorlesung

- Installieren Sie Git oder benutzen Sie einen Rechner mit Git.
- Überprüfen Sie Ihre Zugangsberechtigung zum GITLAB: `https://gitlab.physik.uni-muenchen.de`. Benutzen Sie LDAP plus Ihre Campus-Kennung. Dies ist so etwas wie vorname.zunahme@campus.lmu.de. Sollten Sie keinen Zugang haben, so müssen Sie zuerst dafür sorgen, dass Sie für GITLAB freigeschaltet werden.
- Wechsel Sie in Ihr Home: cd $HOME.
- Überprüfen Sie Ihre Freischaltung des REPOS zur Vorlesung: git clone `https://gitlab.physik.uni-muenchen.de/ls-ruhl/lectures/progtechs/progtech_17022_sose2020.git`. Sie werden nach Ihrem Benutzernamen und Ihrem Campus-Password gefragt.
- Falls Sie Erfolg haben, liegt das Verzeichnis $HOME\progtech_17022_sose2020 vor. In diesem befindet sich ein .git sowie das PDF-File der Vorlesung zum 30.04.2020.

Zugang zum Git für Ihren Feedback

Zugang zum Git für Ihren Feedback

- Installieren Sie Git oder benutzen Sie einen Rechner mit Git.
- Überprüfen Sie Ihre Zugangsberechtigung zum GITLAB: `https://gitlab.physik.uni-muenchen.de`. Benutzen Sie LDAP plus Ihre Campus-Kennung. Dies ist so etwas wie vorname.zunahme@campus.lmu.de. Sollten Sie keinen Zugang haben, so müssen Sie zuerst dafür sorgen, dass Sie für GITLAB freigeschaltet werden.
- Wechsel Sie in Ihr Home: cd $HOME.
- Überprüfen Sie Ihre Freischaltung des REPOS zum Feedback: git clone `https://gitlab.physik.uni-muenchen.de/ls-ruhl/lectures/progtechs/progtech_17022_sose2020_feedback.git`. Sie werden nach Ihrem Benutzernamen und Ihrem Campus-Password gefragt.
- Falls Sie Erfolg haben, liegt das Verzeichnis $HOME\progtech_17022_sose2020_feedback vor. In diesem befindet sich ein .git.
- Halten Sie sich bitte an die Namenskonventionen `https://www.physik.uni-muenchen.de/lehre/vorlesungen/sose_20/programmiertechniken_ein/index.html` für Ihren Feedback und verwenden Sie LATEX.

Referenzen

- http://book.git-scm.com/index.html
- http://www.kernel.org/pub/software/scm/git/docs/git.html