Eine praktische Einführung in GIT

Versionsverwaltung mit git

Simon Balzer

GitHub: sibalzer

mail@simonbalzer.de

https://github.com/sibalzer/git_workshop

1. Einführung in die Versionsverwaltung und GIT
   - Was ist eine Versionsverwaltung
   - Allgemeiner Aufbau und Funktionsweise von GIT
   - Workflow Praxisbeispiel
2. Praxisübung
   - Einrichtung des Git Clients
   - Zwei Aufgabenteile
3. Ausblick
   - GUI Clients
   - GitHub/Gitlab
   - Do‘s and Dont‘s
   - Conventions
   - Einbinden in eine CI/CD-Pipeline

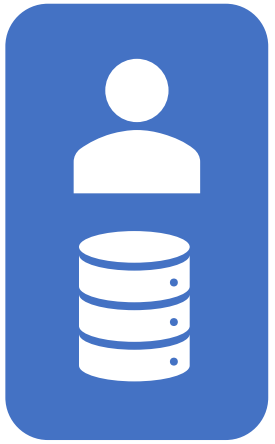
Versionsverwaltung – Was ist das?

„Eine Versionsverwaltung ist ein System, das zur Erfassung von Änderungen an Dokumenten oder Dateien verwendet wird. Alle Versionen werden in einem Archiv mit Zeitstempel und Benutzerkennung gesichert und können später wiederhergestellt werden.“

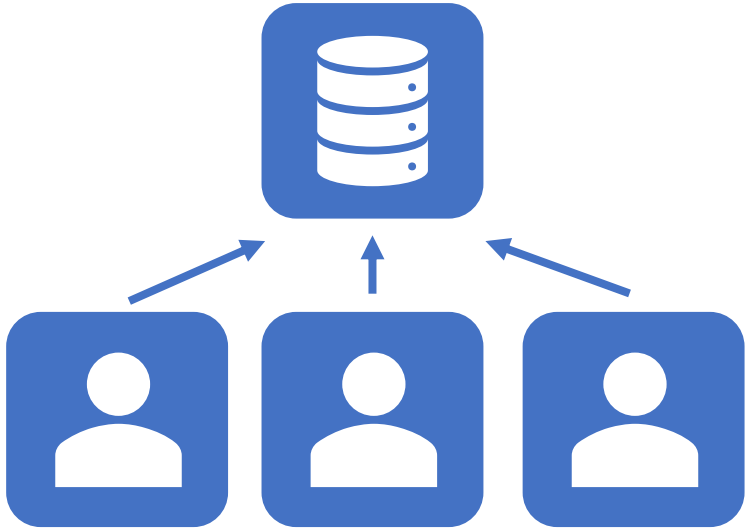
- Einfache Verwaltung von Code in Projekten
- Bessere Kollaboration auch bei größerer Software (Kein austauschen von Zip-Archiven o.Ä. nötig)
- Rückverfolgbarkeit von Änderungen

VCS-Konzepte

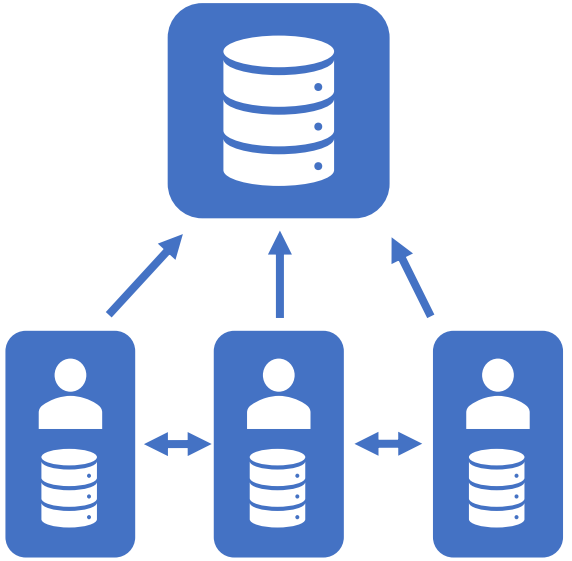
Lokale Versionsverwaltung



Zentrale Versionsverwaltung



Verteilte Versionsverwaltung



Zentralisiertes VCS

- Ein zentrales Repository
- Client-Server-System
- Internetverbindung muss vorhanden sein zum Bearbeiten
- Platzsparender bei großen Projekten

Beispiele: SVN, Perforce, CVS

Verteiltes VCS

- Jeder besitzt eine Kopie des Repository
- Schneller (da alles lokal passiert)
- Einfaches Branchen
- Nicht-lineare Entwicklung

Beispiele: git, mercurial

Warum VCS/Git?

- Einfach
- Zeitersparnis

- Performance
- Kryptographische Sicherheit
- Dezentral
- Open Source
- Standard für viele Open Source Projekte

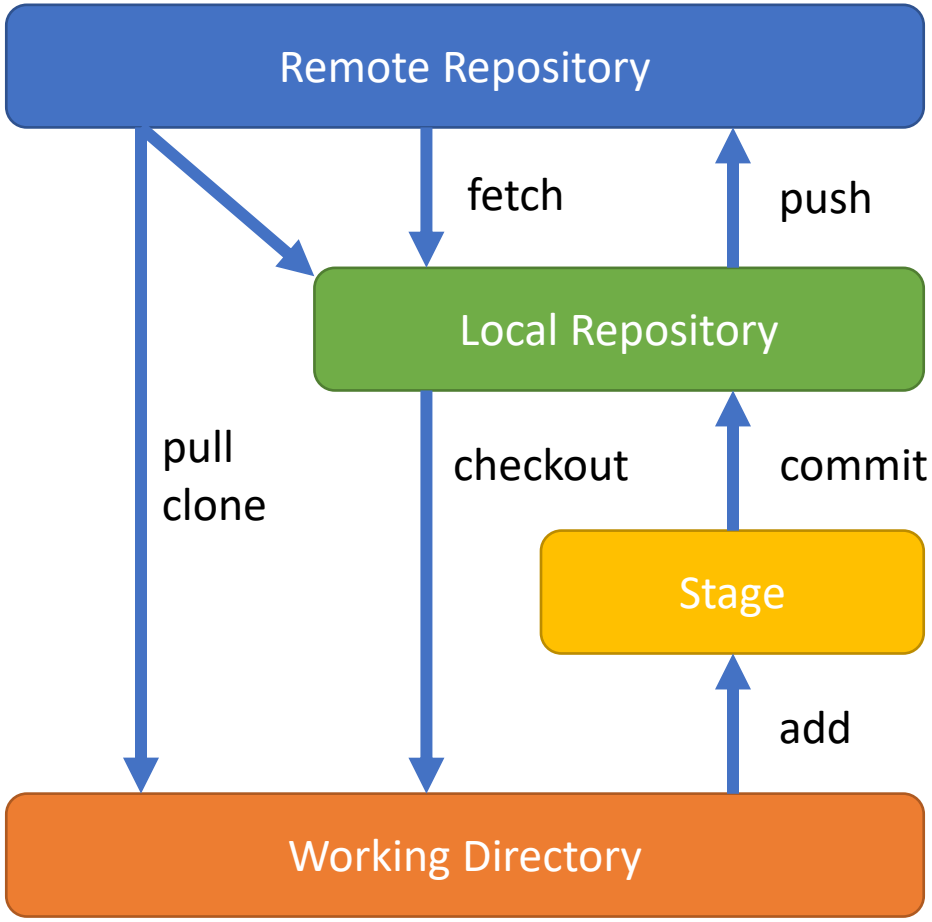
Git - Historie

“I’m an egotistical bastard, and I name all my projects after myself. First ‘Linux’, now ‘Git’.” - Linus Torvalds

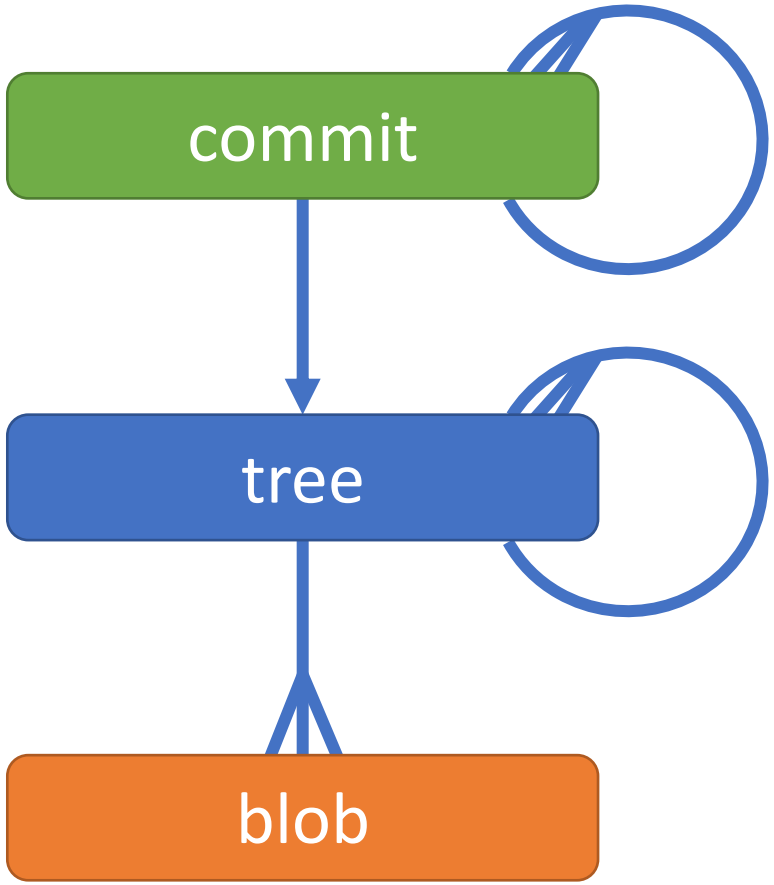
(git [engl. ugs.] = Idiot)

- 2005 entwickelt
- BitKeeper nichtmehr verfügbar
- Keine zufriedenstellende alternative

Aufbau von git



Git Model



ae668..

commit	size
tree	c4ec5
parent	a149e
author	Scott
committer	Scott
my commit message goes here and it is really, really cool	

c36d4..

tree		size
blob	5b1d3	README
tree	03e78	lib
tree	cdc8b	test
blob	cba0a	test.rb
blob	911e7	xdiff

5b1d3..

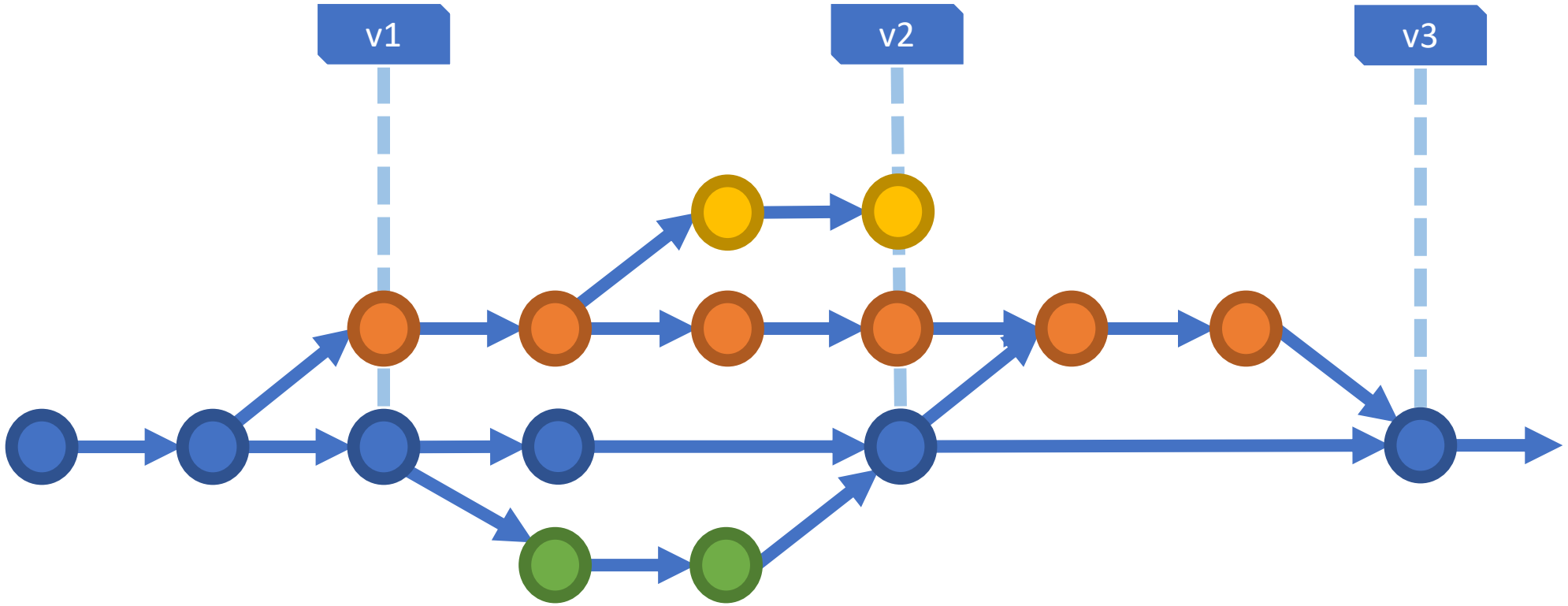
blob	size

```
#ifndef REVISION_H
#define REVISION_H

#include "parse-options.h"

#define SEEN            (1u<<0)
#define UNINTERESTING   (1u
#define TREESAME (1u<<2)
```

Git Baum



Einrichten eines Repositorys

- Lokales (neues) Repository erstellen

```
~$ mkdir ordner
~$ cd ordner
~/ordner$ git init
```

- Remote Repository klonen

```
$ git clone git@github.com:github-tools/github.git
```

```
$ git clone https://github.com/github-tools/github.git
```

Zwischenschritt: .gitignore-Datei

- Eine Liste der Ordner und Dateien die standardmäßig ignoriert werden sollen
- Unterstützt Wildcards

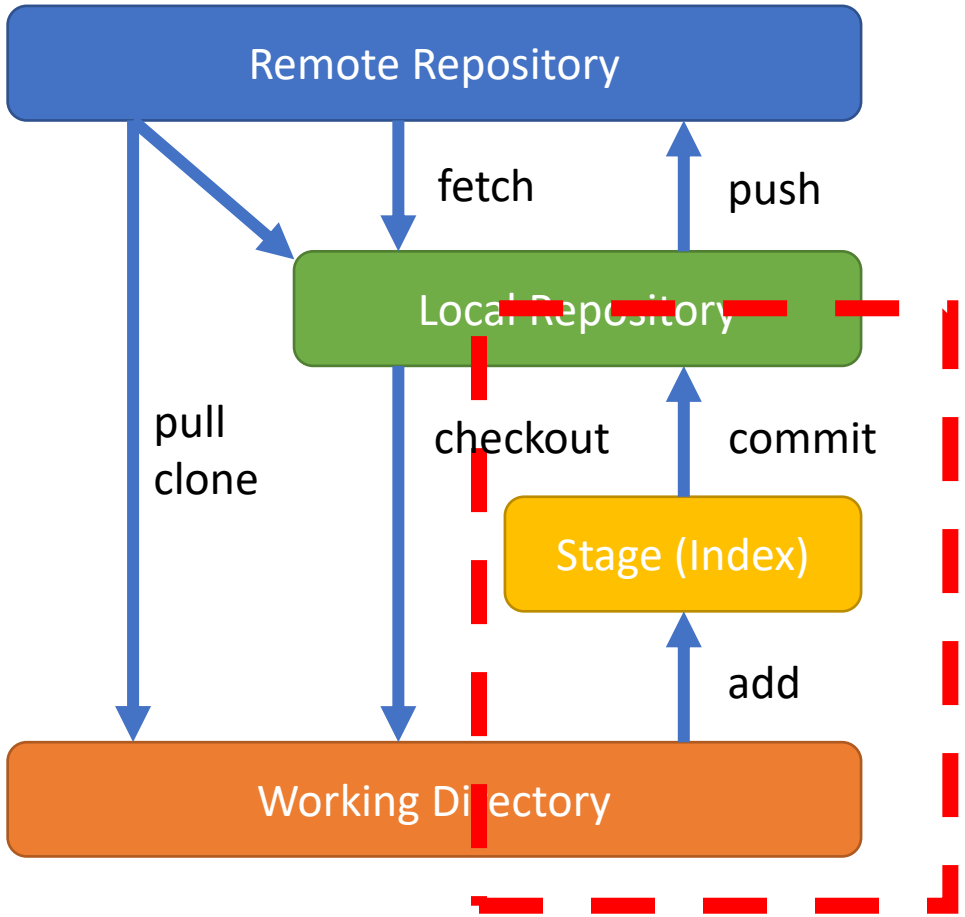
.gitignore

```
config.ini
build/
*.log
!example.log
build/**/compile/
```

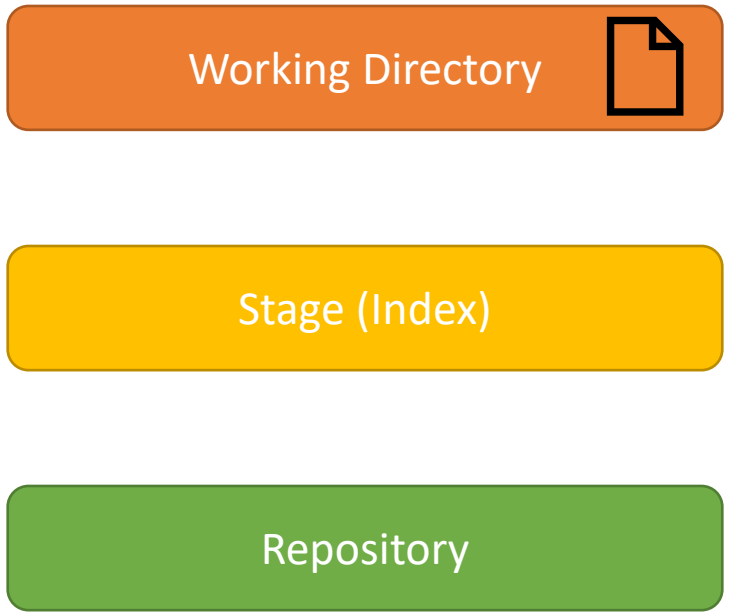
Workflow

1. Edit
2. Stage
3. Review
4. Commit

Workflow


Remote Repository
fetch
push
Local Repository
pull
clone
checkout
commit
Stage (Index)
add
Working Directory

Workflow



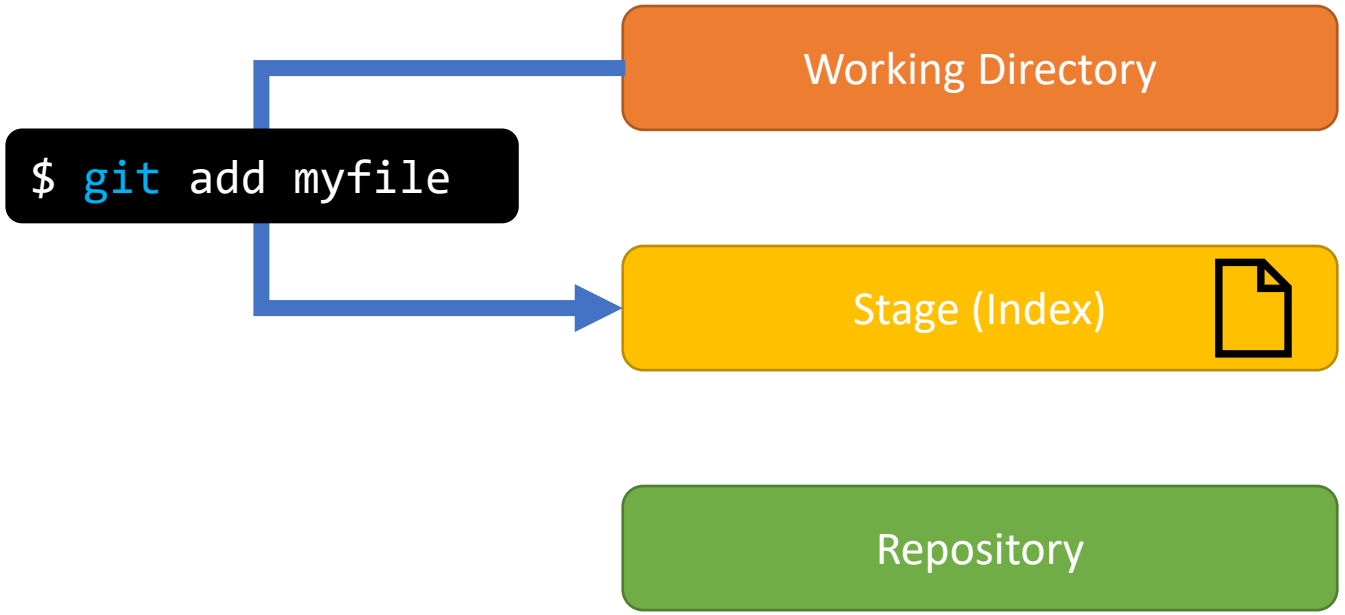
Stage

```
$ git add myfile
```

```
$ git add src/
```

```
$ git add *
```

Workflow



Review

```
$ git status
```

```
$ git diff --cached
```

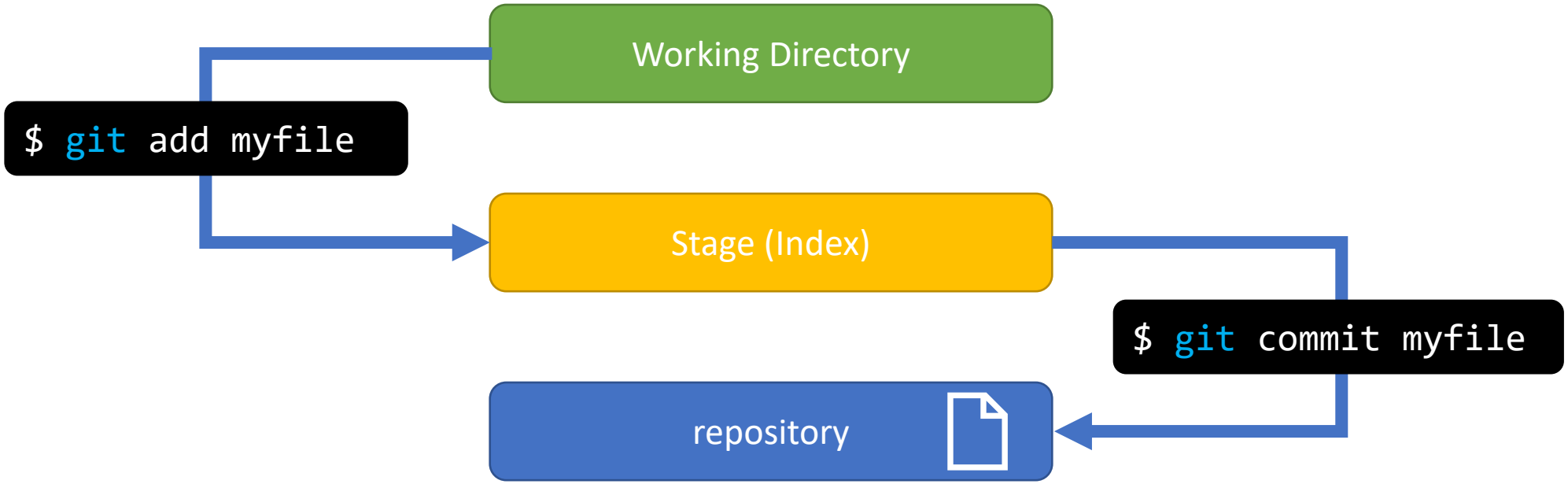
```
diff --git a/src/helloworld.py b/src/helloworld.py
new file mode 100644
index 0000000..81024b2
--- /dev/null
+++ b/src/autofill.py
@@ -0,0 +1 @@
+print("Hello world.")
```

git commit

```
$ git commit
```

```
$ git commit –m „feat: add feature xy“
```

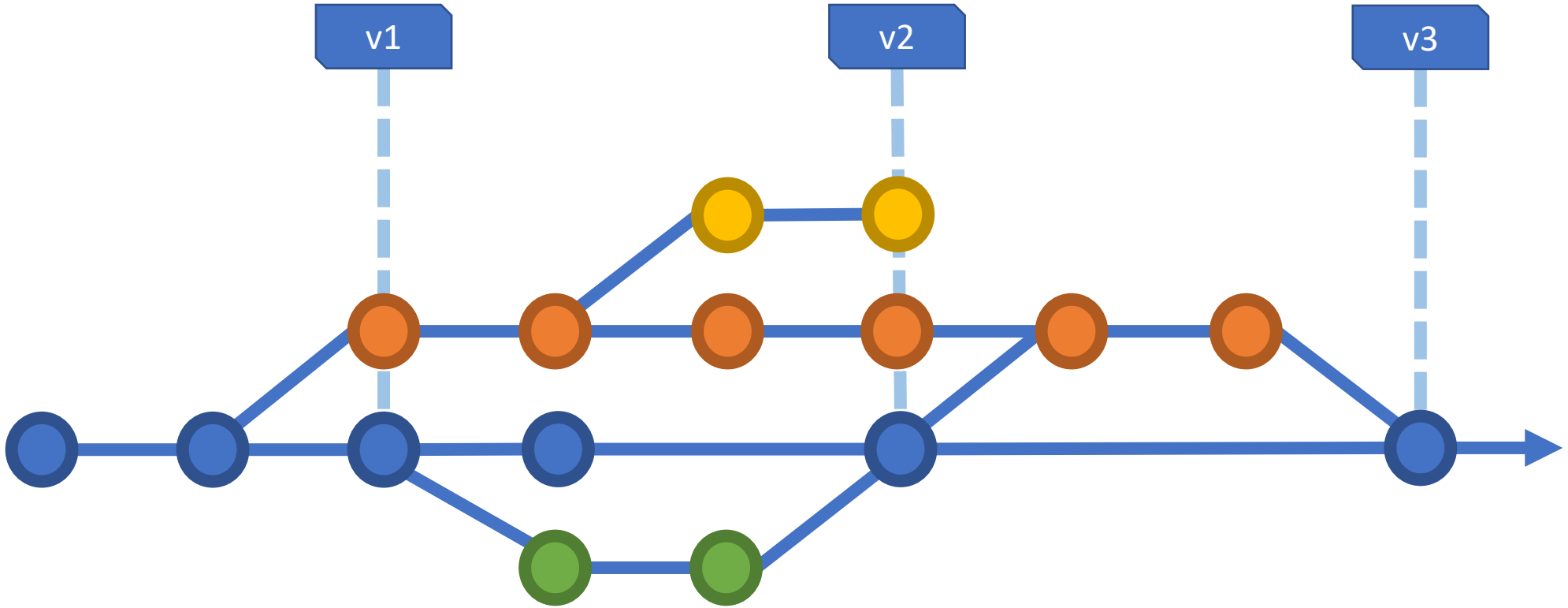
Workflow


Working Directory
$ git add myfile
Stage (Index)
$ git commit myfile
repository

Workflow

1. Edit	vim / IDE
2. Stage	git add
3. Review	git status / git diff
4. Commit	git commit

Branching



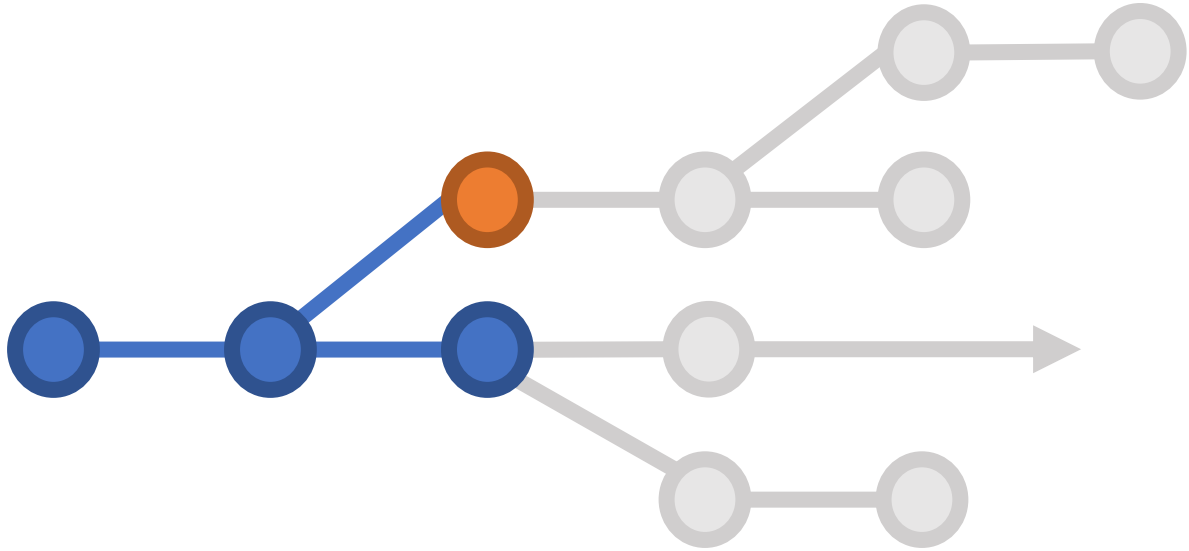
Branching

Branch erstellen und hineinwechseln

```
$ git checkout -b feature
```

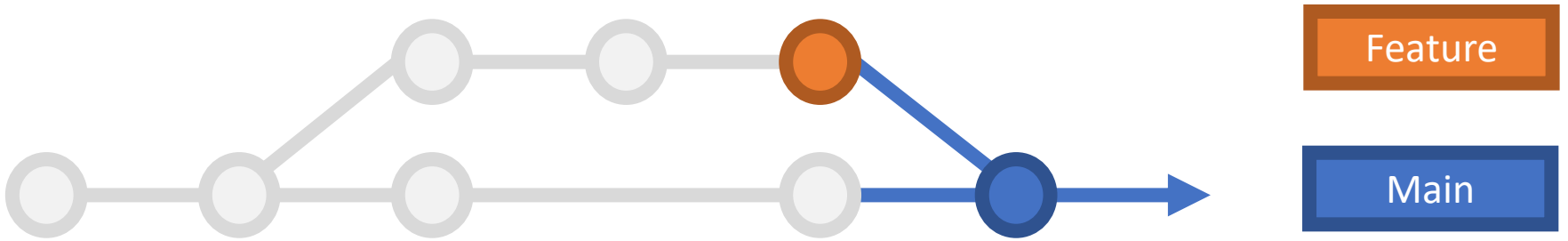
oder

```
$ git branch feature
$ git checkout feature
```

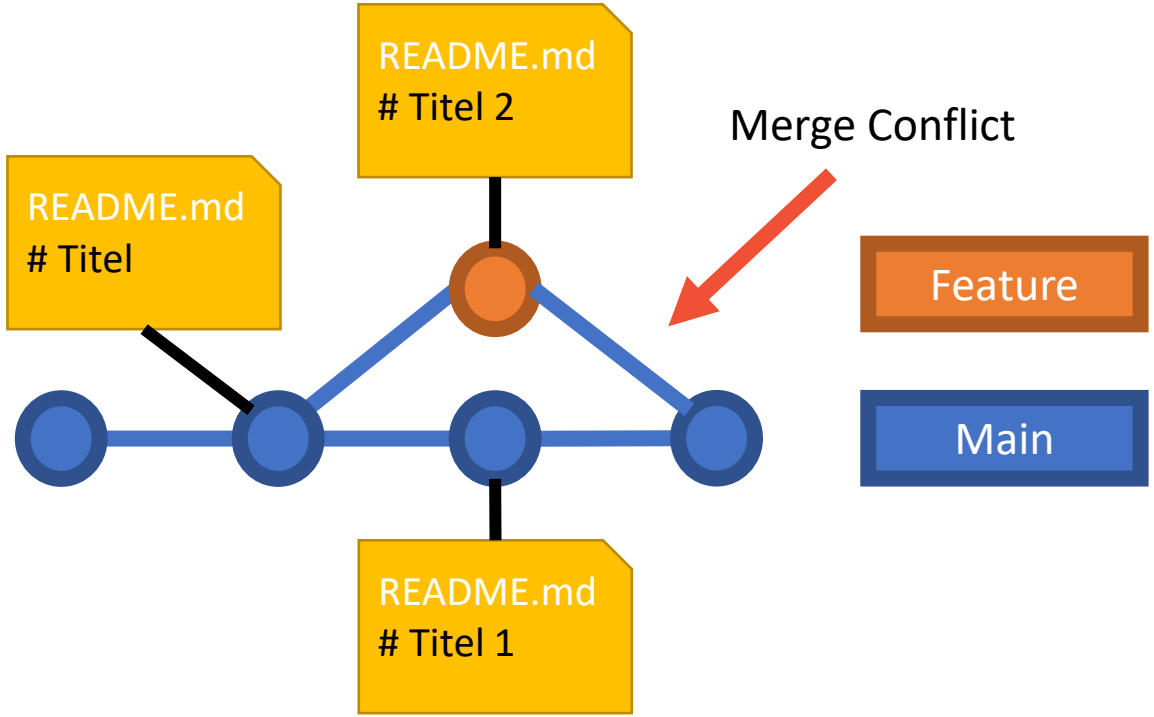


Merge

```
# In den Base Branch (main) wechseln
$ git checkout main
# Merge Feature in main
$ git merge feature
# Lösche den feature-Branch
$ git branch -d feature
```



Merge Conflict



Merge Conflict

```
$ git merge feature
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then
commit the result.
```

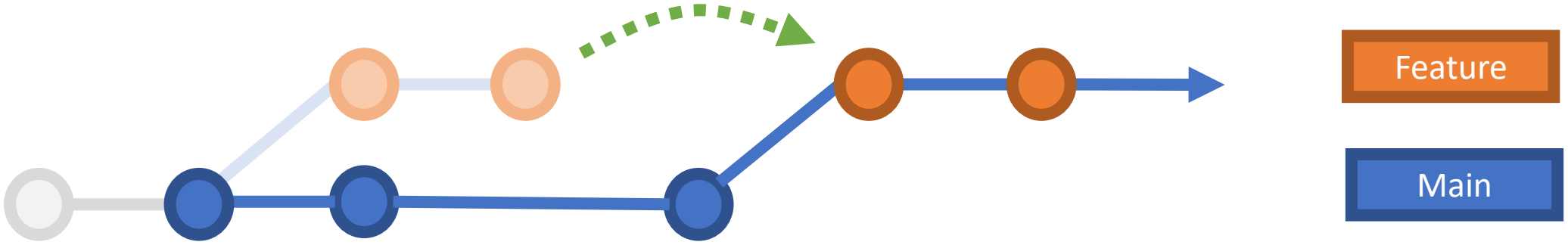
README.md
<<<<<<< HEAD
Titel 1
=======
Titel 2
>>>>>>> feature

1. Datei öffnen
2. Konflikte in den markierten Bereichen lösen
3. Stagen
4. Merge-Commit erstellen

=> Merge manuell durchgeführt

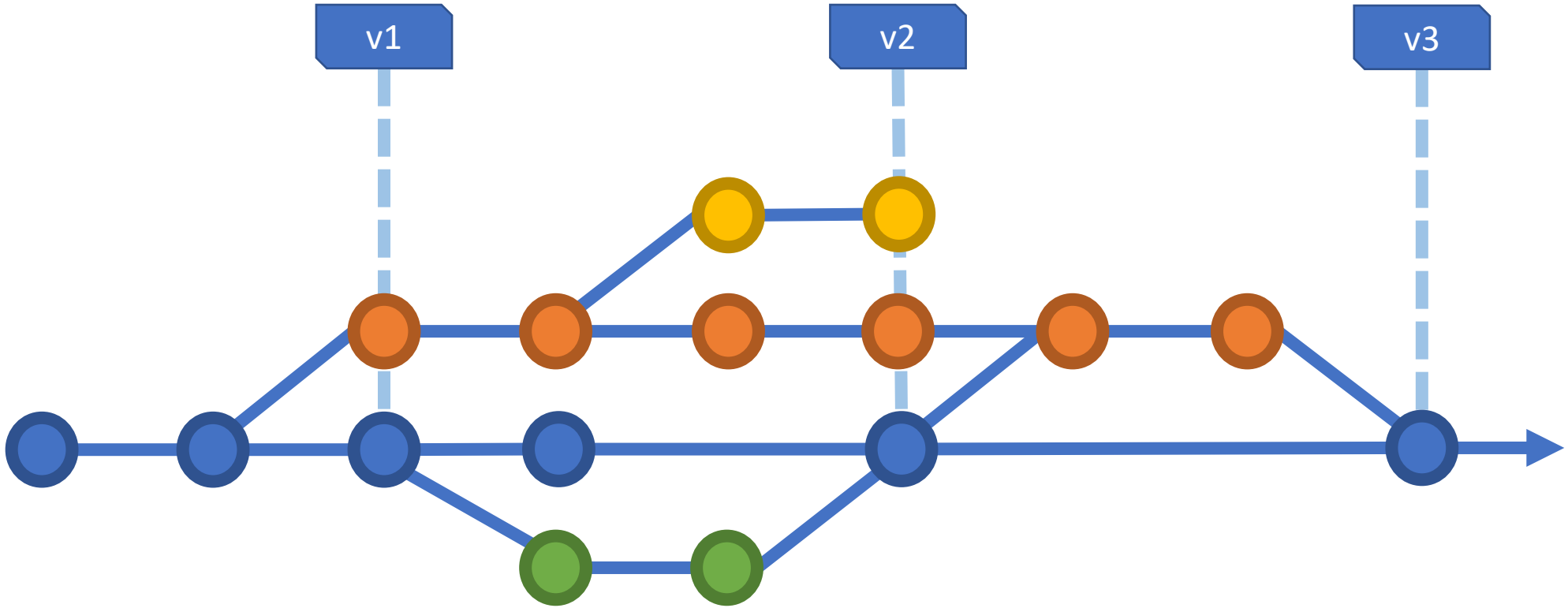
Rebase

```
# In den Base Branch (main) wechseln
$ git checkout main
# Merge Feature in main
$ git rebase feature
# Lösche den feature-Branch
$ git branch -d feature
```



Tagging

```
$ git tag [-a] v1.0
```



Branch veröffentlichen / Änderungen holen

- Branch veröffentlichen

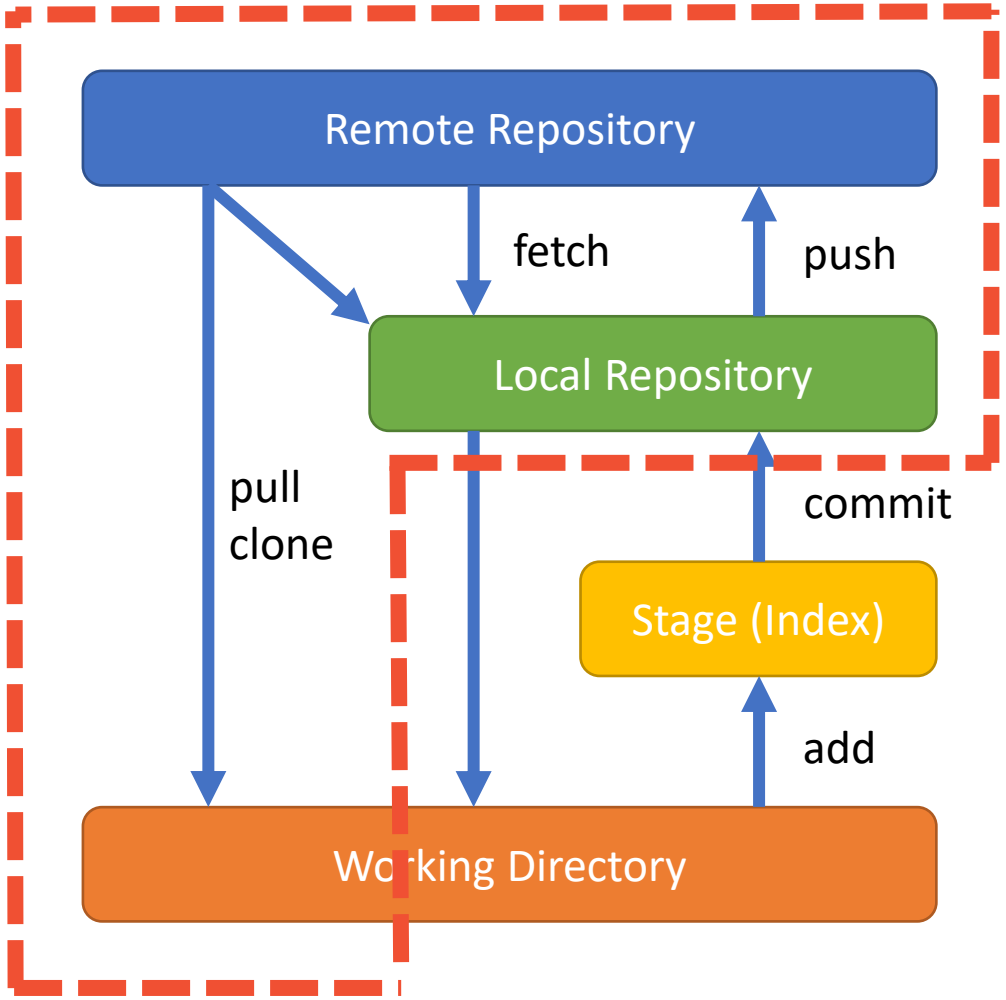
```
$ git push [origin mybranch]
```

- Änderungen holen

```
$ git pull
```

oder

```
$ git fetch
$ git rebase
```



Zwischenablage

- Alle Änderungen in die Zwischenablage verschieben

```
$ git stash
```

- Änderungen zurückholen

```
$ git stash apply
```

```
$ git stash apply stash@{0}
```

- Auflisten der Objekte in der Zwischenablage

```
$ git stash list
```

- Obersten Eintrag löschen

```
$ git stash drop
```

```
$ git stash apply drop@{1}
```

Sonstige

- Infos zur letzten Änderung

```
$ git blame
```

- Historie anzeigen

```
$ git log
```

- Letzten Commit löschen

```
$ git commit -amend
```

- Working Directory auf HEAD zurücksetzen

```
$ git reset [--hard]
```

- Einen Commit per „cherry pick“ rückgängig machen

```
$ git revert <hash>
```

Zusammenfassung

- Einrichten eines Repositorys

```
$ git init
```

```
$ git clone
```

- Änderungen einpflegen

```
$ git add
```

```
$ git status
```

```
$ git commit [-m „msg“]
```

- Mit dem Remote Repository Kommunizieren

```
$ git push
```

```
$ git fetch
```

```
$ git pull
```

- Branchen

```
$ git checkout [-b]
```

- Versionen taggen

```
$ git tag
```

Einmalige Einrichtung von Git

- Als erstes muss git Konfiguiert werden dafür gibt es diese beiden Befehle:

```
$ git config --global user.name „Simon Balzer"
$ git config --global user.email „mail@simonbalzer.de"
```

- Um einen SSH-Schlüsselpaar zu generieren enthält git zudem den Befehl: `$ ssh-keygen`

Aufgabenteil 1

- Lade das Repository runter
- Erstelle einen Branch mit deinem Namen
- Kopiere example.html zu <meine_seite>.html
- Füge diese zur Versionsverwaltung hinzu
- Lade deinen Branch hoch

Aufgabenteil 2

- Verändere die Überschrift deiner Website
- Füge die Änderung zur Versionsverwaltung hinzu
- Versucht diesen Branch hochzuladen (Was ist passiert?)

Graphische Git-Clients

- GIT-GUI
- Sourcetree
- Git Kraken
- Github-Client
- IDEs besitzen meist per Default oder Addon einen Client (bis auf wenige Ausnahmen)
  - IntelliJ IDEA
  - Visual Studio
  - VS Code
  - Eclipse
  - NetBeans
  - Android Studio

Conventional Commits

- Convention um Commit-Nachrichten maschinenlesbar zu machen
- Eine Nachricht besteht aus einem Typen (einem optionalen Scope)
  - Ein Ausrufezeichen nach dem Scope signalisiert einen Commit mit „BREAKING CHANGES“
  - Zusätzlich/Alternativ kann es noch einen Footer mit der Bezeichnung „BREAKING CHANGE“ geben
- Typen sind feat, fix, refactor, perf, style, docs, test, chore, revert, ci, build

https://www.conventionalcommits.org

<type>[(optional scope)] [optional !]: <description>
[optional body]
[optional footer]

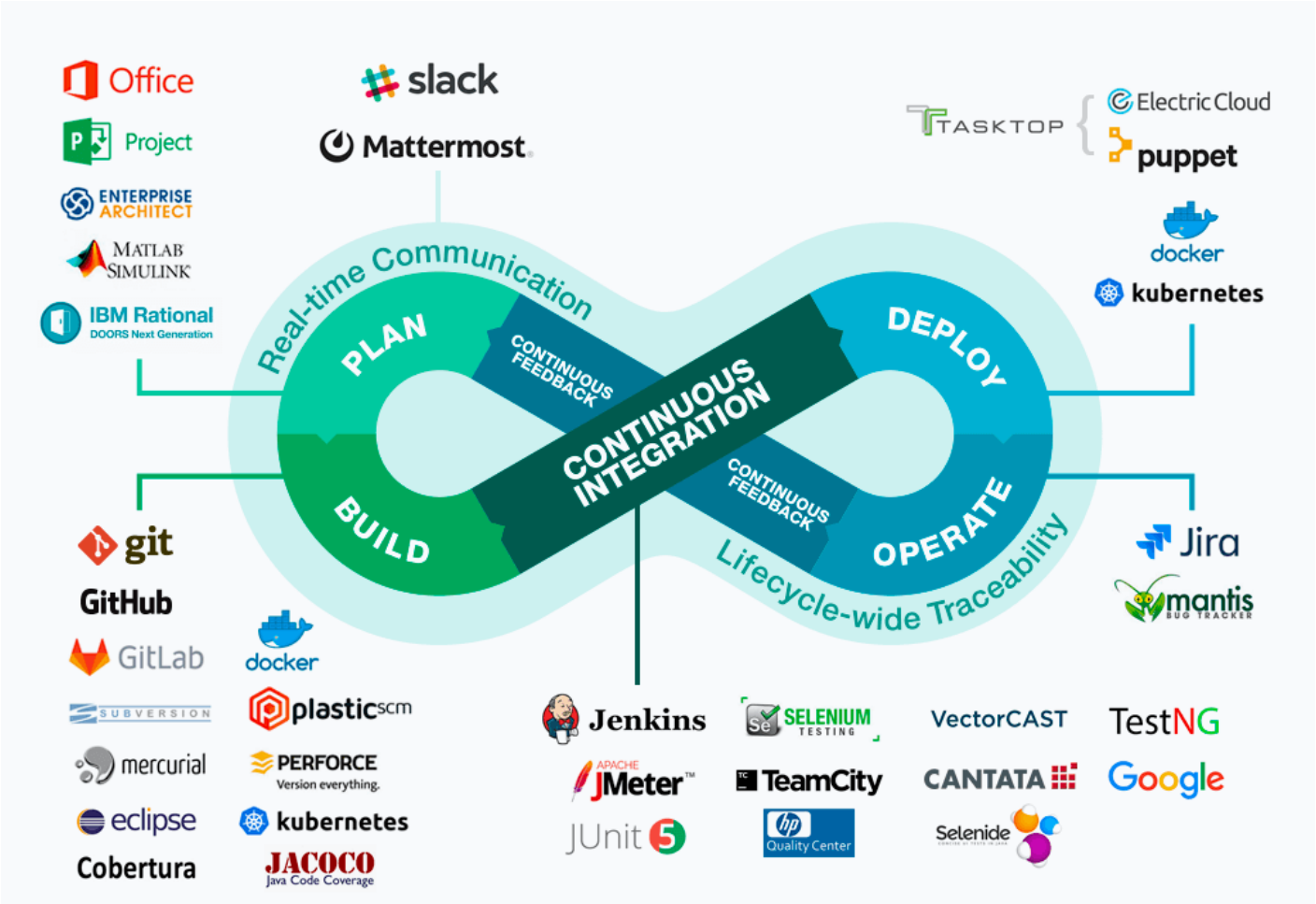
feat(lang)!: added klingon language
BREAKING CHANGE: database requieres ‚klingon‘ encoding set

Workflow mit GitHub/GitLab

1. Issue erstellen
2. (Repository forken)
3. Neuen Development Branch erstellen
4. Änderungen in diesen Einpflegen
5. Pull Request erstellen
6. Approval bekommen
7. Merge

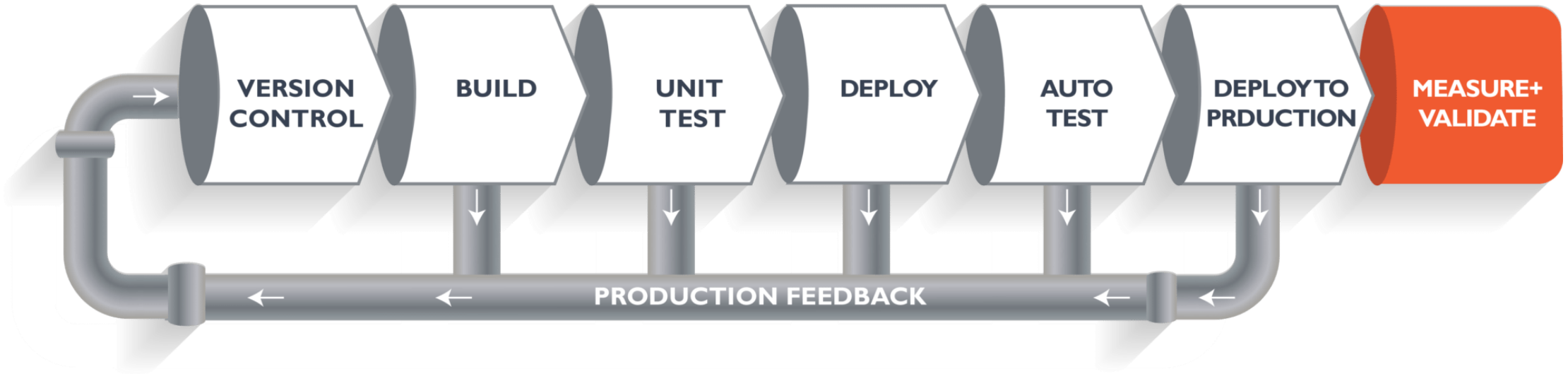
1. Issue erstellen
2. (Repository forken)
3. Neuen Merge-Request erstellen
4. Änderungen in diesen Einpflegen
5. Approval bekommen
6. Merge

Git im DevOp Cycle



Quelle: Intland Software
https://intland.com/codebeamer/devops-it-operations/

CI/CD mit Git



Quelle: DZone
https://dzone.com/articles/learn-how-to-setup-a-cicd-pipeline-from-scratch

Wo finde ich Hilfe?

- https://www.atlassian.com/git/tutorials
- https://rogerdudler.github.io/git-guide/index.de.html
- https://rogerdudler.github.io/git-guide/files/git_cheat_sheet.pdfhttps://education.github.com/git-cheat-sheet-education.pdf
- http://git-scm.com
- http://git-scm.com/book/en/v2

Danke für die Aufmerksamkeit – Feedback?

GitHub: sibalzer

mail@simonbalzer.de

https://github.com/sibalzer/git_workshop