Git-Grundlagen für Entwickler

Thomas Claudius Huber
@thomasclaudiush

Thomas Claudius Huber

- Principal Consultant bei Trivadis
- Pluralsight-Autor, Buchautor
- C#, XAML, TypeScript, Azure

www.thomasclaudiushuber.com

@ThomasClaudiusH

Microsoft® MVP Most Valuable Professional

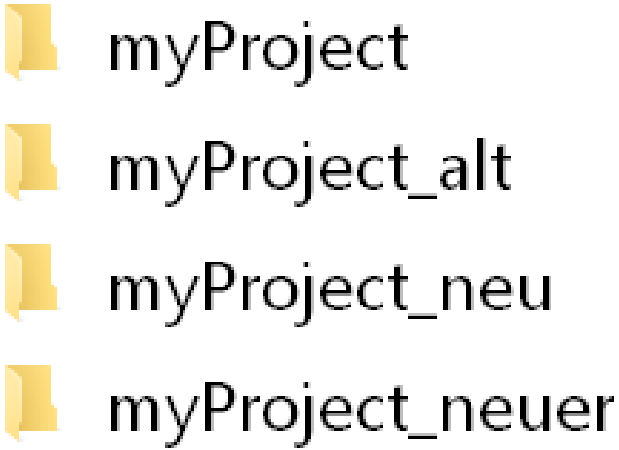
Agenda

- Git Grundlagen
- Init, Commit, Branching
- Remote Repositories

GIT GRUNDLAGEN

Ohne Version Control System (VCS)

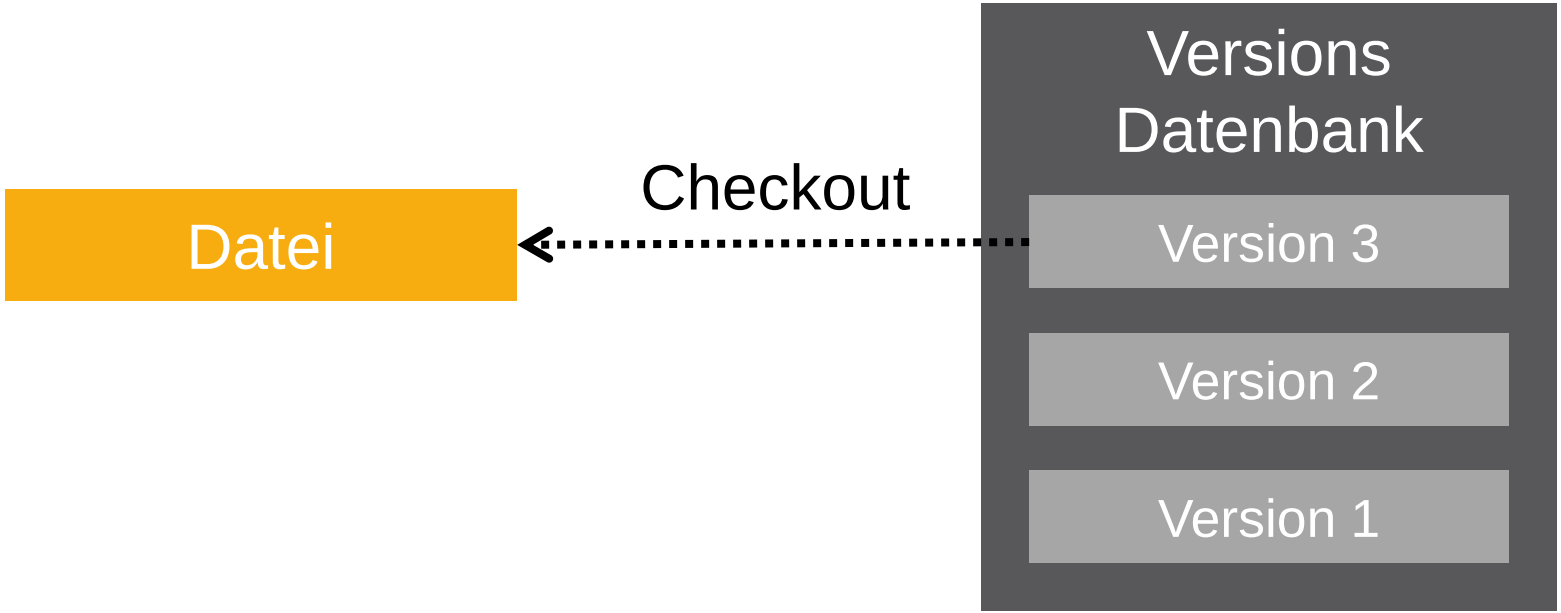
Ohne Version Control System (VCS)



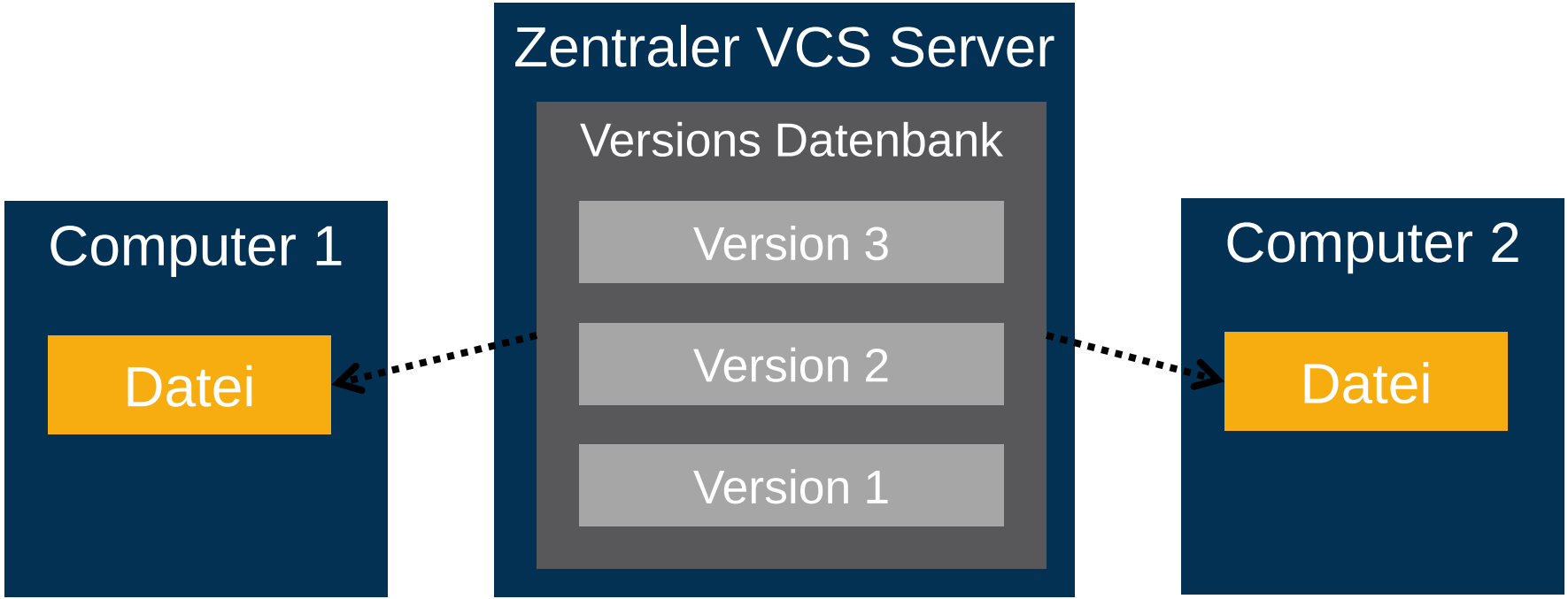
VCS Arten

- Lokale
- Zentralisierte
- Verteilte

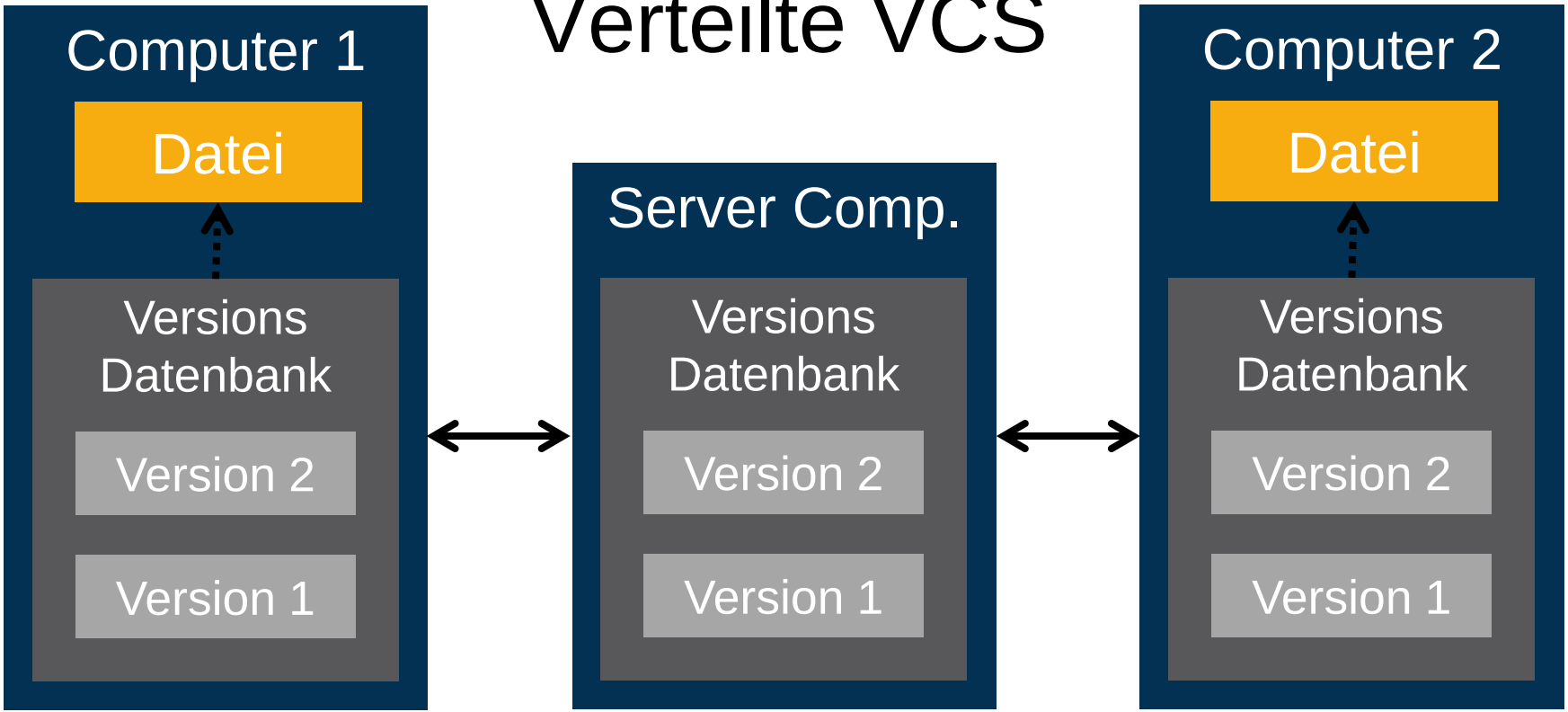
Lokale VCS


Versions
Datenbank
Version 3
Version 2
Version 1
Checkout
Datei

Zentralisierte VCS



Verteilte VCS



Git Grundlagen

DER LINUX-KERNEL

Verwaltung des Linux Kernels

1991

Änderungen am Linux Kernel via patches und archive files

2002

Linux Kernel wird mit BitKeeper verwaltet

2005

Bruch zwischen Community und BitKeeper

2005 – Git Anforderungen

Geschwindigkeit

Verteiltes VCS

Einfaches Design

Umgang mit riesigen Codebasen
(Linux-Kernel)

Starke nicht lineare Entwicklung

2005 – die Geburt von Git

“Git” => englisch für

Blödmann

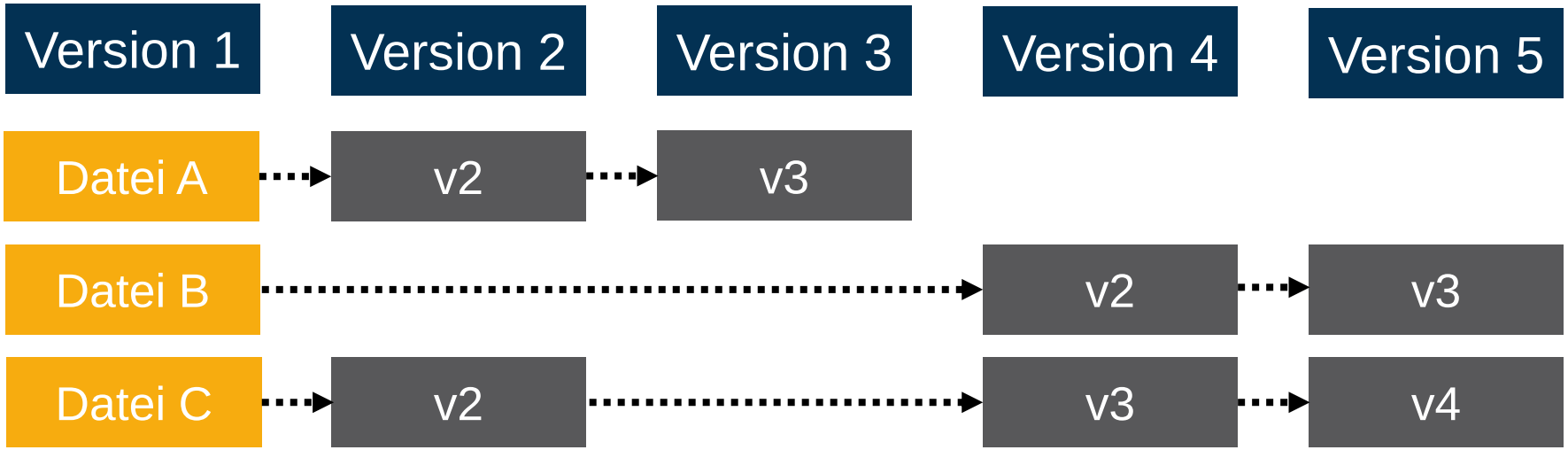
Linus Torvalds: *“Ich bin ein egoistischer Bastard, und ich benenne alle Projekte nach mir selbst, erst Linux, jetzt Git“*

- Gründe für den Namen: Kurz, leicht auszusprechen, leicht zu tippen, einzigartig, kein belegtes Standard Command auf der Konsole

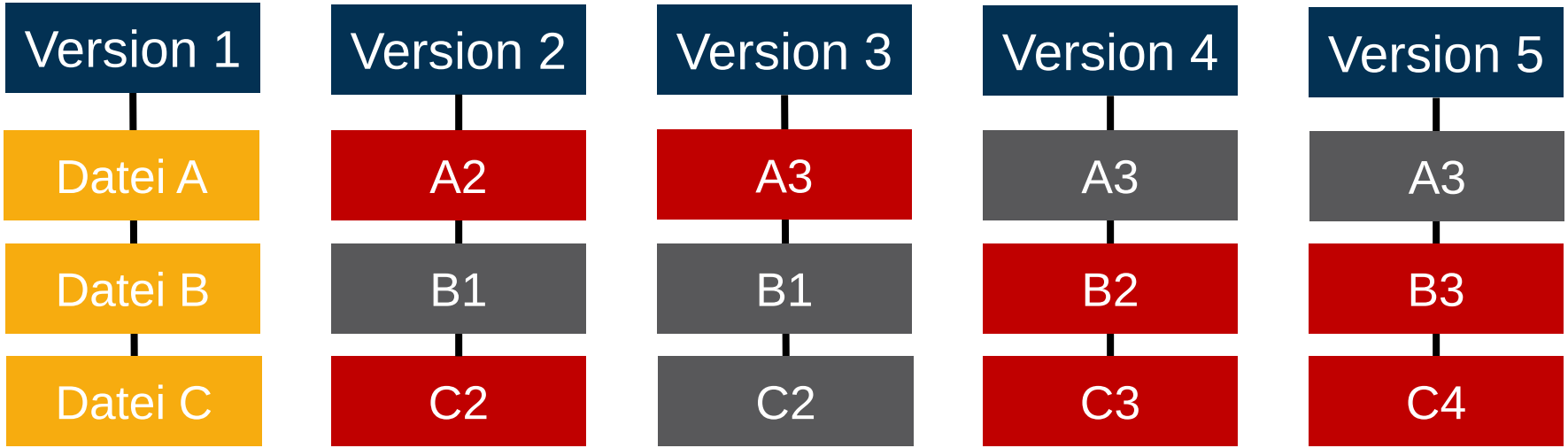
Git Grundlagen

SNAPSHOTS

Klassische Versionierung



Snapshots in Git



Snapshots in Git

- Werden über einen Hash-Wert identifiziert
- SHA-1 hash
  - String mit 40 Zeichen

```
b8bef49f3ff89c41b585d994489efd8fe8b2949b
```

Git Grundlagen

GIT INSTALLIEREN

Git installieren

- Installieren via https://git-scm.com/downloads
  - Enthält neben Git auch die Git Bash und Git GUI
  - Erlaubt Git auf der Commandline
- Visual Studio enthält Git

Git konfigurieren

- Konfiguration anzeigen:

```
git config -l
```

- Benutzer einrichten (in jedem Commit):

```
git config --global user.name "Thomas Claudius Huber"
git config --global user.email "thomas@....com"
```

- Einzelne Konfiguration abfragen:

```
git config user.name
```

Git konfigurieren

- Config wird in verschiedenen Ebenen gespeichert

C:/ProgramData/Git/config

C:\Users\$USER

- Darüber hinaus pro Repository in

.git/config

- Ursprung der Konfiguration anzeigen:

```
git config –l --show-origin
```

DEMO:
Git konfigurieren

Agenda

- Git Grundlagen
- Init, Commit, Branching
- Remote Repositories

Ein Repository erstellen

- Repository initialisieren

```
git init
```

- Den Status abfragen

```
git status
```

- Den Status in Kurzform abfragen

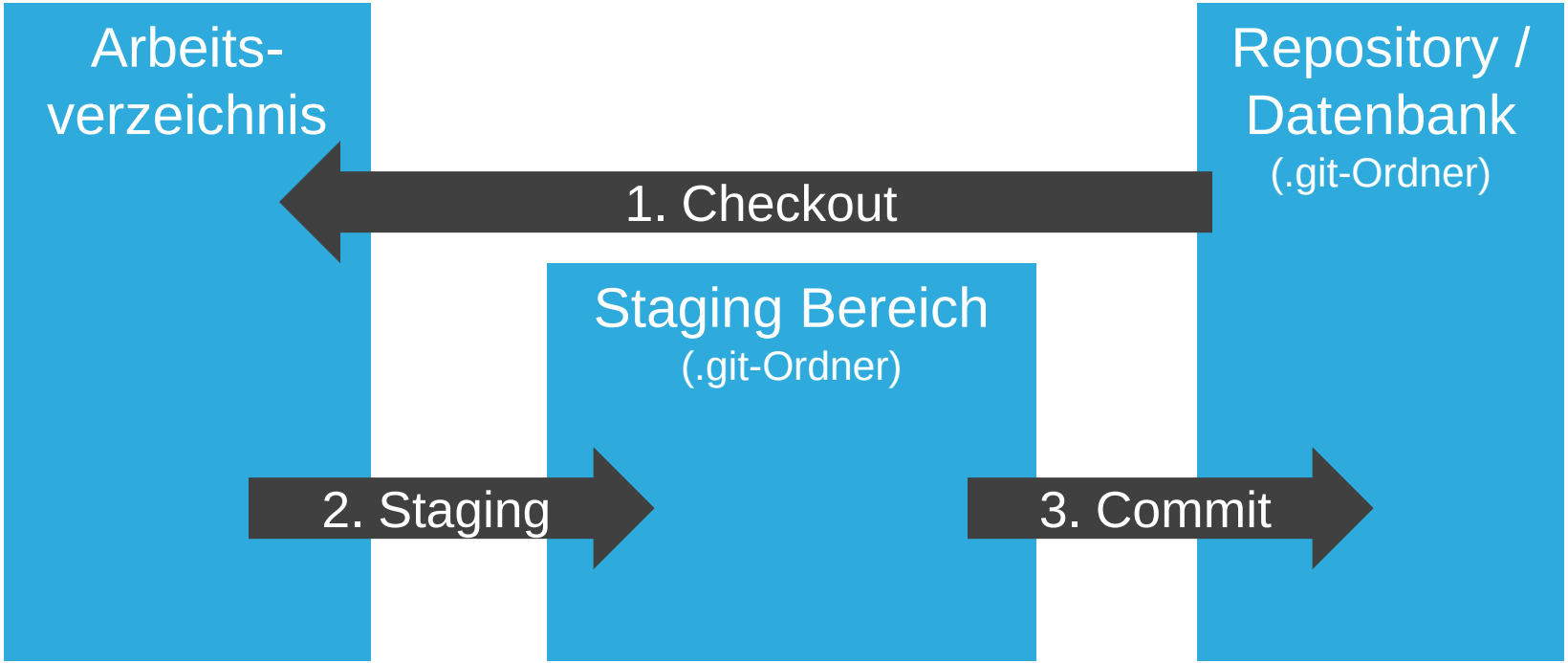
```
git status -s
```

DEMO:
Repository erstellen

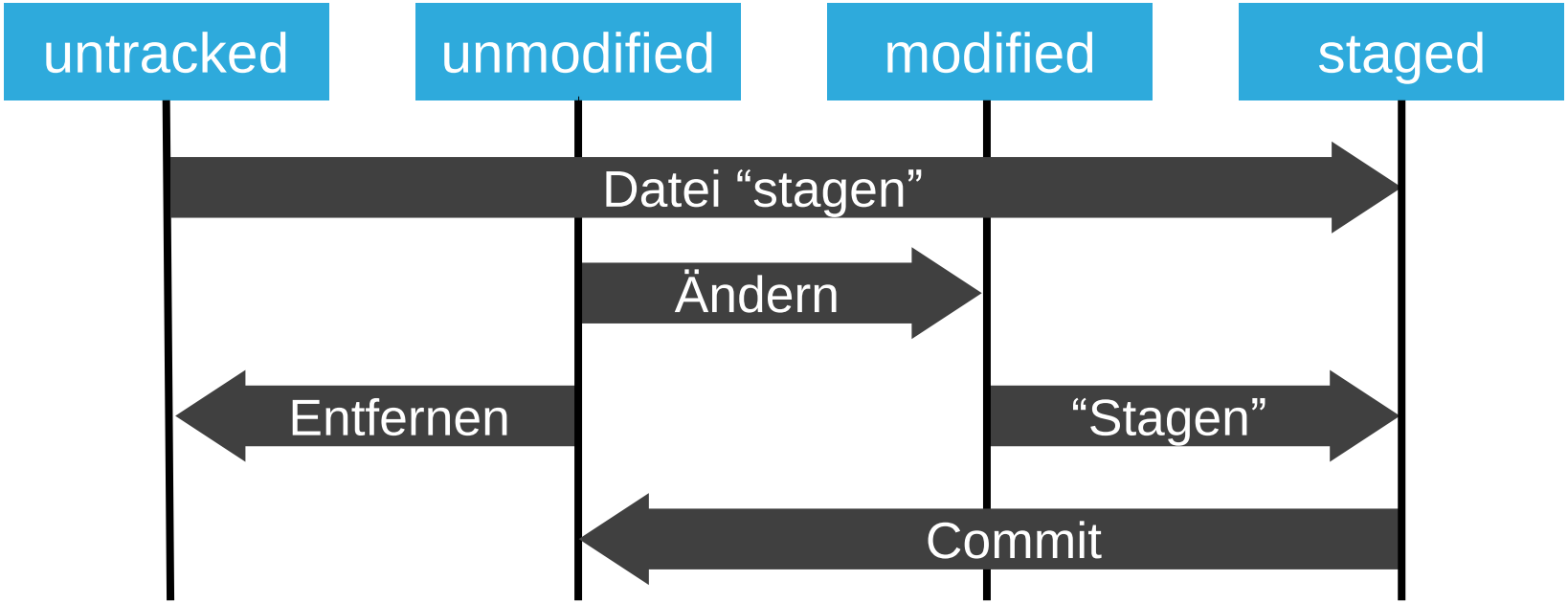
Mit Repositories arbeiten

DAS ARBEITSVERZEICHNIS

Das Arbeitsverzeichnis



Die Dateizustände



Git auf der Kommandozeile nutzen

- Status abfragen

```
git status
```

- Datei “stagen”

```
git add dateiname.txt
```

```
git add .
```

- Committen

```
git commit –m “Meine Commit-Nachricht”
```

DEMO:
Mit Dateien arbeiten

Git auf der Kommandozeile nutzen

- Datei entfernen

```
git rm dateiname.txt
```

- Diff anzeigen

```
git diff
```

```
git diff --staged
```

- Diff in Tool anzeigen

```
git difftool
```

```
git difftool --tool-help
```

DEMO:
Diff anzeigen

Log ansehen

- Zeigt Commits an (mit Snapshot-Hash (SHA1)):

```
git log
```

```
git log --pretty=oneline
```

- git checkout “Hashanfang” checkt den commit aus:

```
git checkout fea23ce22
```

DEMO:
Logs ansehen und
Commits laden

Dateien ignorieren

- .gitIgnore-Datei anlegen
- Syntax:
  - Leere und mit # startende Zeilen werden ignoriert
  - Negation mit einem Ausrufezeichen (!)
  - Erlaubt standard Glob patterns (=wie einfache Regex)
  - / am Anfang, um Rekursion zu vermeiden
  - / am Ende, um Verzeichnis zu wählen

Dateien ignorieren: Glob Patterns

* - keines, eines oder mehrere Zeichen

? – genau ein Zeichen

[abc] – einen Buchstaben in den Klammern

[0-9] – ein Zeichen in der Reihe (Zahlen von 0 bis 9)

** - um verschachtelte Verzeichnisse zu erhalten:

a/**/x findet a/x, a/b/x, a/b/c/x etc.

.gitIngore

```
# Alle txt-Dateien ignorieren
*.txt
# Nur txt-dateien aus root-Ordner ignorieren
/*.txt
# Aber nicht die readme.txt
!readme.txt
# Alle Dateien im myProject/bin-Verzeichnis ignorieren
myProject/bin/
# Alle .suo-Dateien im myProject-Verzeichnis
# (incl. Unterverzeichnisse ignorieren)
myProject/**/*.suo
```

.gitIngore

- Viele gute *.gitIngore*-Beispiele sind auf

https://github.com/github/gitignore

DEMO:
.gitIngore anlegen

Branches erstellen

- Branches anzeigen (Aktueller mit * gekennzeichnet

```
git branch
```

- Branch erstellen und ins Arbeitsverzeichnis auschecken

```
git branch myNewFeature
```

```
git checkout myNewFeature
```

- Zurück zum master wechseln

```
git checkout master
```

Branches mergen

- Branches mergen:

```
git merge myNewFeature
```

- Neben merging gibt es auch das sog. “rebasing”
  - Commits werden auf branch appliziert
    - Niemals rebasing auf remote-branch machen!
- Mit --graph den Graph auf der Console sehen:

```
git log --pretty=oneline --graph
```

DEMO:
Mit Branches arbeiten

Tags erstellen

- Tag erstellen

```
git tag –a MeinTag –m “Ein schöner Tag”
```

- Tag auschecken

```
git checkout MeinTag
```

- Zurück zum master-branch wechseln

```
git checkout master
```

DEMO:
Tags nutzen

Agenda

- Git Grundlagen
- Init, Commit, Branching
- Remote Repositories

Remotes hinzufügen

- Remotes abfragen

```
git remote
```

- Remotes anlegen

```
git remote add <name> <url>
```

Remote aktualisieren

- Branch in Remote pushen

```
git push <remote-name> <branch-name>
```

```
git push origin master
```

- Upstream mit –u-parameter setzen:

```
git push –u origin master
```

```
git push
```

Lokales Repo aktualisieren

- Zum ersten Mal in aktuelles Verzeichnis kopieren:

```
git clone <remote-url> .
```

  - fügt remote automatisch unter Namen “origin” hinzu

- Neue Daten laden und mergen:

```
git pull origin master
```

- Oder mit gesetztem Upstream einfach

```
git pull
```

Pulling

- Git pull merged remote-branch in lokalen:

```
git pull
```

- Auch in zwei Schritten möglich

```
git fetch
```

```
git merge
```

Remotes auflisten

- Alle Branches anzeigen, nicht nur lokale:

```
git branch -a
```

DEMO:
In GitHub-Repo pushen/pullen

Was ist ein Pull-Request

- Anfrage, den eigenen Branch in den Hauptbranch zu mergen
- *git pull* = *git fetch* und *git merge*

DEMO:
Pull-Request durchführen

Agenda

- Git Grundlagen
- Init, Commit, Branching
- Remote Repositories

Genutzte Befehle

```
git clone <repo-url>
git add .
git status
git commit –m “Comment”
git push –u origin master
git fetch
git merge

git --help
git <befehl> -help
```

Summary

- Git ist ein verteiltes Versionskontrollsystem
- Es gibt drei Bereiche
  - Repository
  - Staging
  - Arbeitsverzeichnis
- Fast alle Befehle lassen sich lokal ausführen

Danke

thomas.huber@trivadis.com

www.trivadis.com

www.thomasclaudiushuber.com

@ThomasClaudiusH