Versionskontrolle mit Git

20.1.2014

Was ist Versionskontrolle? (VCS)

VCS zeichnen Änderungen an Dateien auf und erlauben späteren Zugriff auf spezifische Versionen dieser Dateien.

Warum VCS?

- Dateien wiederherstellen
- Zwischen Projektständen wechseln
- Änderungen und Bugs verfolgen
- Projektfortschritt anzeigen
- Kollaboration
- Backups
- …

Best Practice

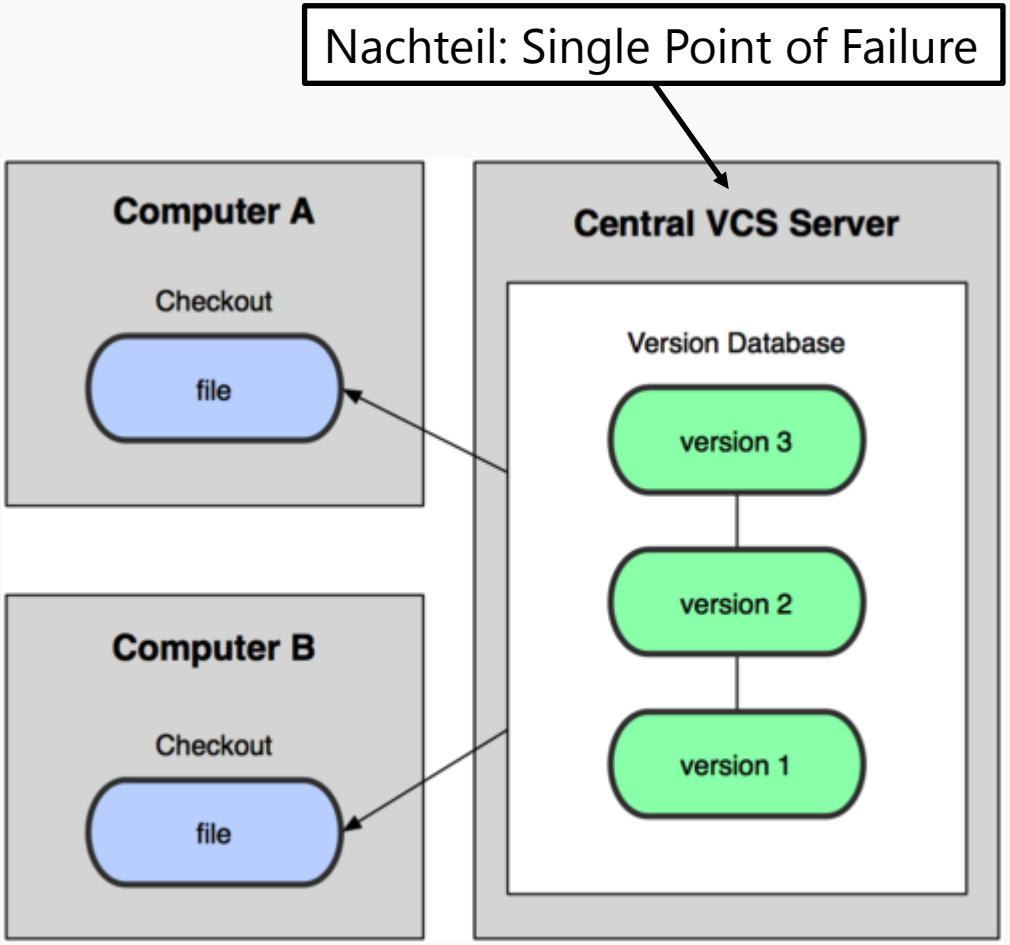
Versionskontrolle ist bei größeren Projekten Pflicht und **Best Practice**!

Ohne VCS…

- Dateien kopieren?
  Fehleranfällig…
- Cloud-Sync (Dropbox, Google Drive…)?
  Nicht ausgelegt auf Quellcode…

CVCS (Central Version Control Systems)

CVS, SVN, Perforce…

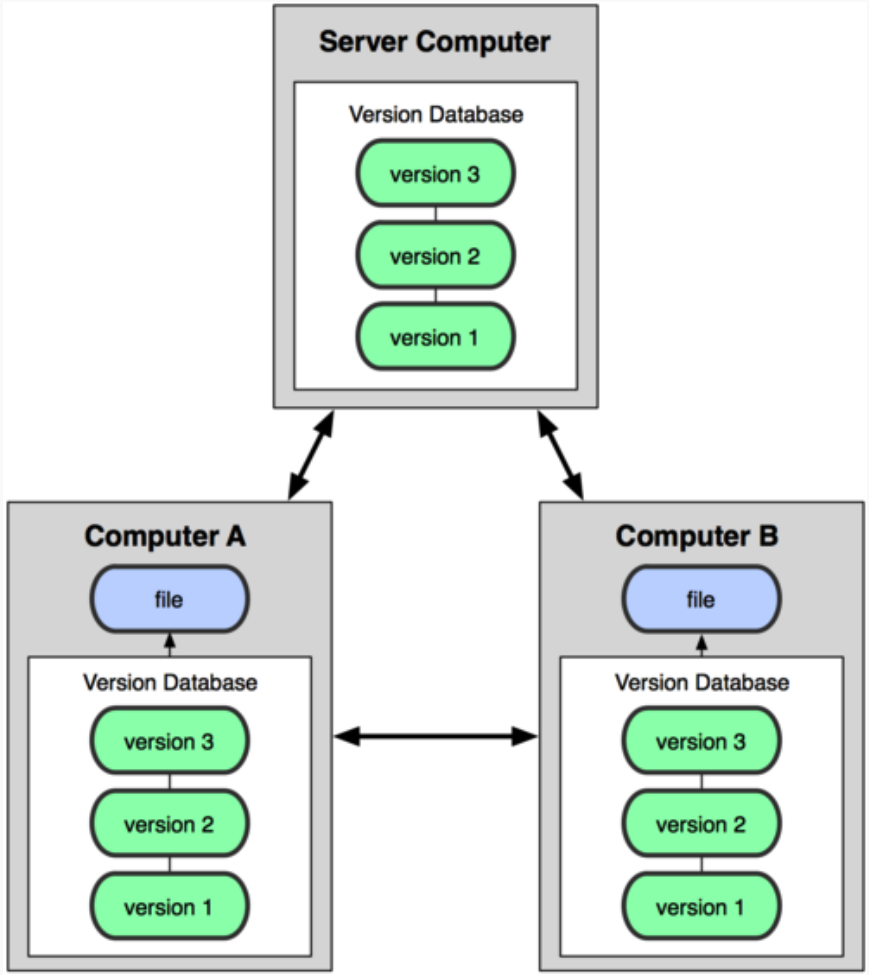


http://git-scm.com/figures/18333fig0102-tn.png

DVCS (Distributed Version Control Systems)

Git, Mercurial, Bazaar…

Vorteil: Vollständiges lokales Repository auf jedem Rechner (Security & Speed)



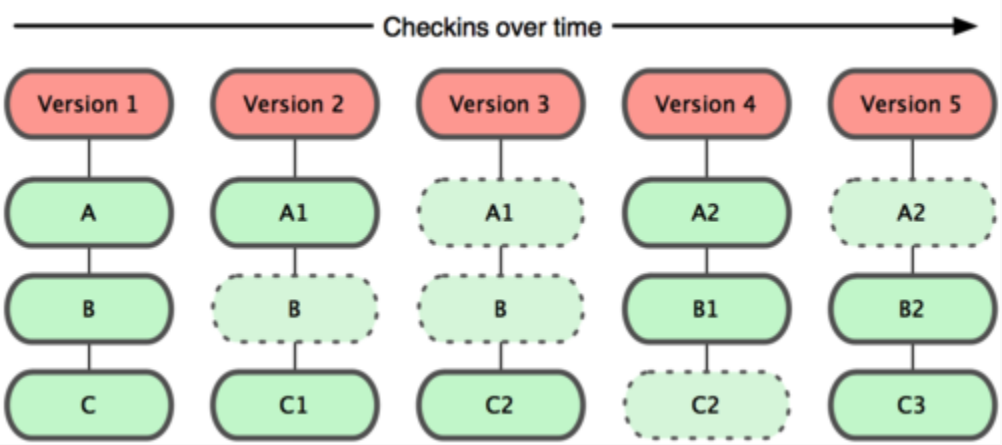
http://git-scm.com/figures/18333fig0103-tn.png

Git

- 2005 zur Verwaltung des Linux Kernels entwickelt (von Linus Torvalds himself…)
- Inzwischen weit verbreitet, populär für Open Source Projekte
- Github!

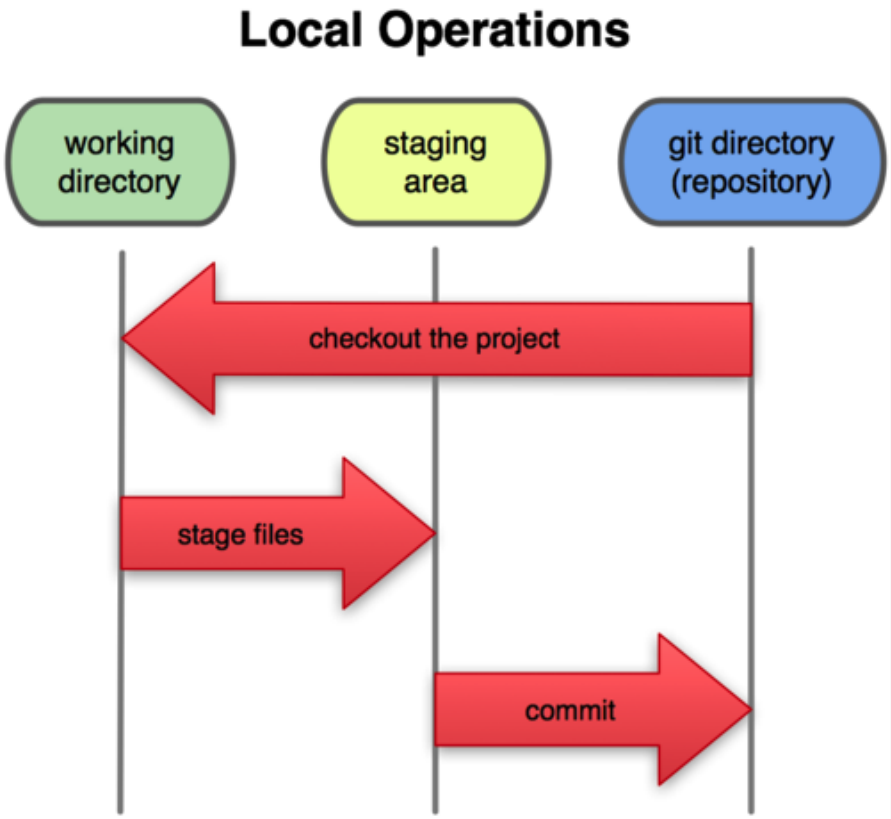
Snapshots

- Git speichert Snapshots
- Alle Operationen werden zunächst lokal angewendet und funktionieren offline



http://git-scm.com/figures/18333fig0105-tn.png

Git Workflow



http://git-scm.com/figures/18333fig0106-tn.png

1. Dateien verändern
   (Änderungen in working directory)
2. Dateien „stagen“
   (Snapshots der Dateien der staging area hinzufügen)
3. Dateien „committen“
   (Snapshot der staging area in Git-Verzeichnis speichern)

Repository klonen

```
cd /path/to/my/myworkspace
```

(Nach Workspace-Verzeichnis wechseln.)

`git clone https://github.com/UniRegensburg/MME.git`

(Alle Dateien aus Remote Repository in Verzeichnis MME in /path/to/my/myworkspace klonen.)

```
cd /path/to/my/myworkspace/MME
ls -al (Windows: dir)
```

(Alle Dateien auflisten.)

.gitignore

- Konfigurationsdateien von Entwicklungsumgebungen
- Sensitive Daten (Passwörter…)
- Temporäre Dateien
- Log-Dateien
- .DS_Store
- …

Dateien stagen (Index)

```
git status
```

`git add -A`

(Alle Dateien „stagen" – neue, geänderte, gelöschte…)

```
git status
```

Dateien committen

```
git commit –m "Adding project files"
```

(Bereits getrackte Dateien mit dieser Message (-m) committen.

History ausgeben

```
git log
git log --pretty=oneline -2
git log --pretty=format:"%h - %an, %ar : %s"
git log --pretty=format:"%h %s" --graph
```

Undo

```
git reset HEAD test.txt
```

(Datei aus staging area entfernen, „unstagen“)

```
git checkout -- test.txt
```

(Datei in working directory mit Datei aus letztem Commit ersetzen)

Remote Repositories

```
git pull origin master
```
(Änderungen abrufen und in lokalen Master-Branch mergen)

```
git push origin master
```
(Änderungen in Remote Repository übertragen)
(Erst pull, dann push!)

Branching

`git checkout -b my-new-feature`

(Neuen Branch my-new-feature erstellen und zu diesem Branch wechseln.)

```
git branch
```

(Alle Branches auflisten.)

```
[edit some files...]
```

```
git add -A
git commit -m "Add new feature for…"
```

(Änderungen in my-new-feature stagen und committen..)

Merging

```
git checkout master
```
(Zu Master-Branch wechseln.)

`git merge my-new-feature`
(Branch my-new-feature in den Master-Branch mergen.)

```
git push origin my-new-feature
```
(Lokalen Branch my-new-feature in Remote Branch my-new-feature übertragen.)

```
git branch -d my-new-feature
```
(Branch my-new-feature löschen.)

Dokumentation: Pro Git

http://git-scm.com/book