

# **Versionsverwaltung**

Seminar Softwareentwicklung in der  
Wissenschaft

Robert Wiesner

- Gliederung
- Motivation
- Allgemeines

- Varianten der Versionsverwaltung
- Versionierungssysteme
- Git
- Zusammenfassung
- Diskussion

## Gliederung

- Motivation
- Allgemeines
- Varianten der Versionsverwaltung
- Versionierungssysteme
- Git als Versionierungssystem-Beispiel
- Zusammenfassung
- Diskussion

## Ziele der Versionsverwaltung

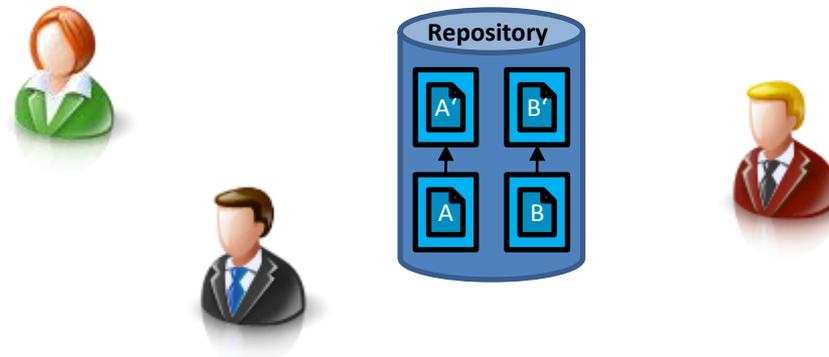
- Inkrementelle Entwicklung von *Versionen*
- Verfügbarkeit der *Historie*
- Parallele Entwicklung von *Branches* (Zweigen)
- *Integration* von Versionen vieler Entwickler
- Nachvollziehbarkeit von Änderungen
- *Koordinierte* Entwicklung
- Effiziente *Fehlerbehandlung* und -suche

## Zentrale Begriffe<sup>[LOE09]</sup>

- Version Control System (VCS)
- Revision Control System (RCS)
- Source Code Manager (SCM)
- Permutationen von
  - „revision“,
  - „version“,
  - „code“,
  - „content“,
  - „control“,
  - „management“,
  - „system“

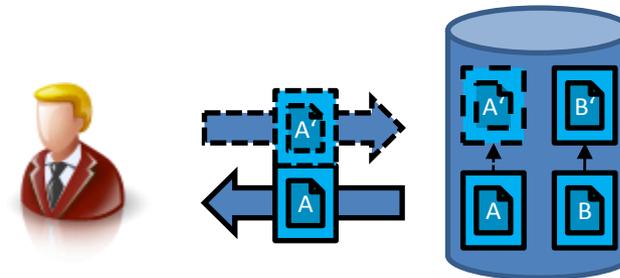
## Zentrale Begriffe

- Das *Repository* als Datenbank
- Verwaltet *Versionen* von Daten
- Zeitlich geordnete Versionen als *Historie*
- *Arbeitskopie*: Datenversion in Bearbeitung



## Arbeitsweise

- *Aktualisiere/Initialisiere/Klone* ein Repository
- Ändere Daten in der Arbeitskopie
- Aktualisiere das Repository (*commit*)



## Arbeitsweise

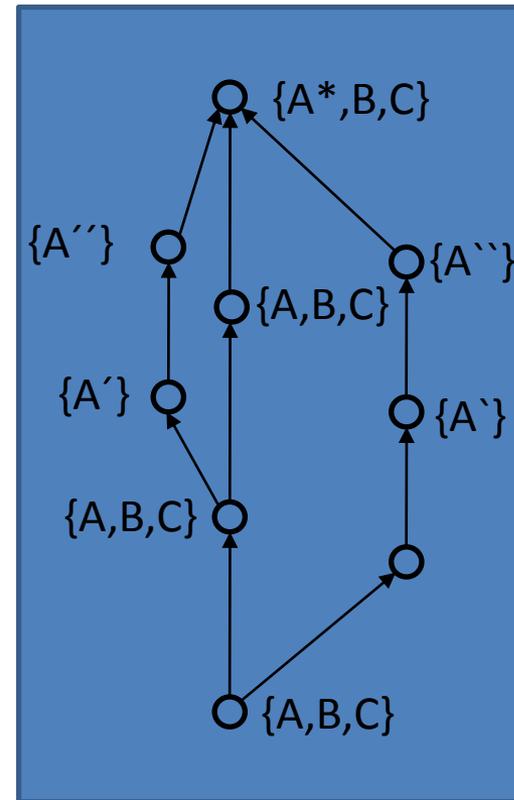
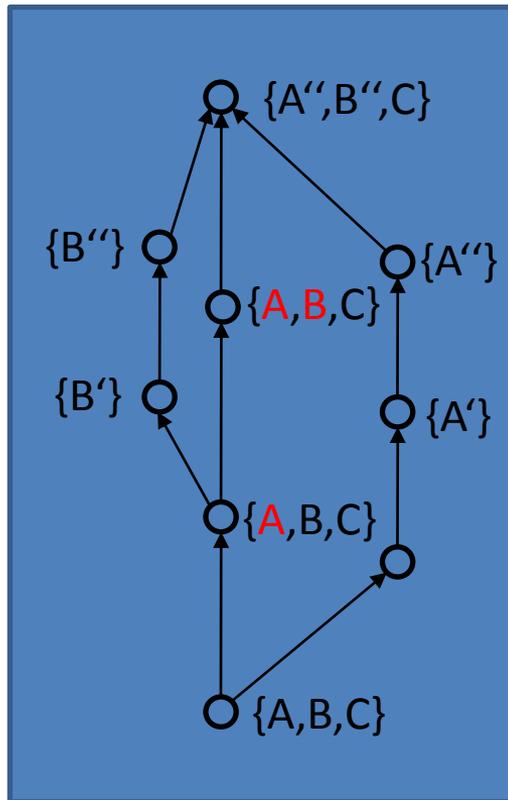
- Lock – Modify – Write
  - Verteilung über logische Einheiten
  - Sperrobjekte nicht nebenläufig bearbeitbar
  - Kein *Zusammenführen (Merging)* notwendig
- Copy – Modify – Merge
  - Mehrere Arbeitskopien einer Datenversion
  - Ermöglicht *Verzweigung*
  - Erfordert gegebenenfalls Zusammenführen

- Gliederung
- Motivation
- Allgemeines

- Varianten der Versionsverwaltung
- Versionierungssysteme
- Git

- Zusammenfassung
- Diskussion

# Lock – Modify – Write / Copy – Modify – Merge



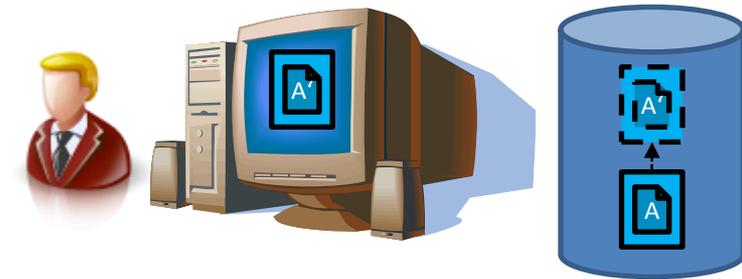
## Varianten der Versionsverwaltung

- Lokale Versionsverwaltung
  - Einzelner Rechner
- Zentrale Versionsverwaltung
  - Vernetzte Rechner
- Verteilte Versionsverwaltung
  - Vernetzte Rechner

## Lokale Versionsverwaltung

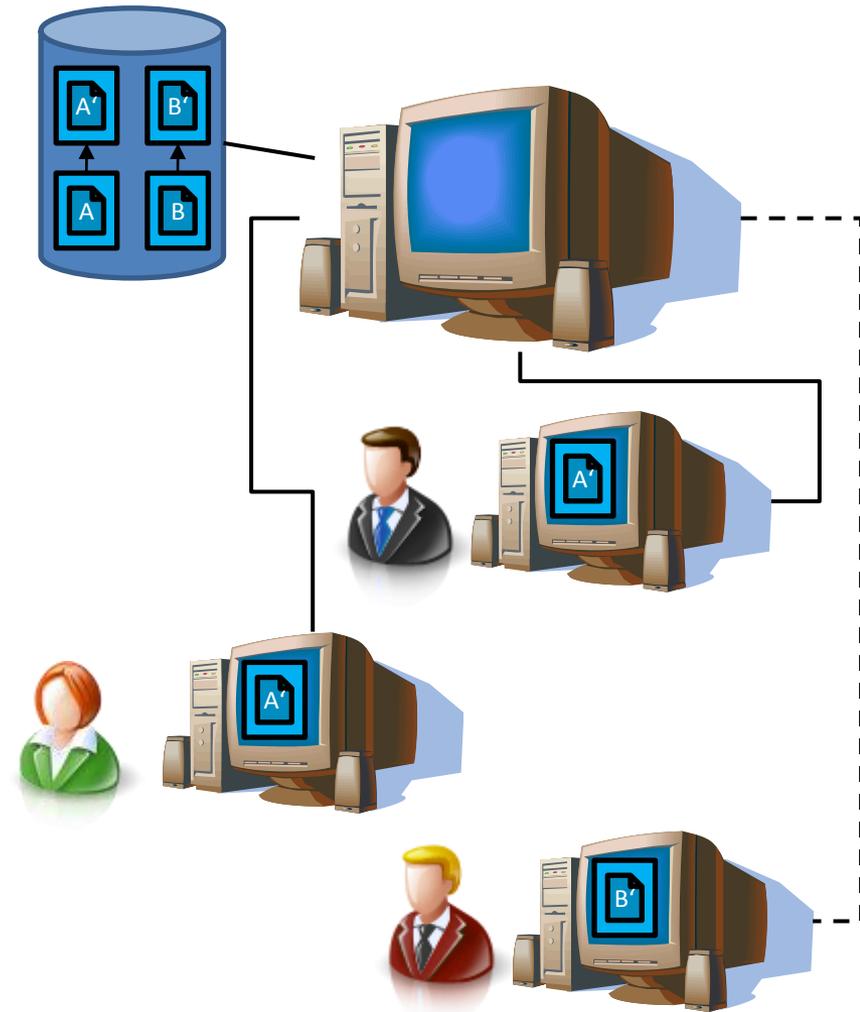
### Lokale Versionsverwaltung:

- Dateiorientiert
- Serielle Entwicklung
- In Büroanwendungen
- Für SIW eher uninteressant



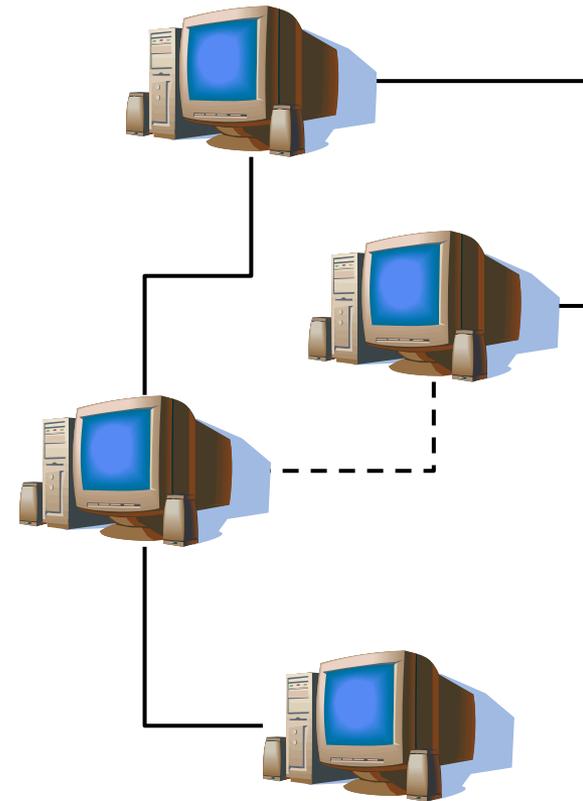
## Zentrale Versionsverwaltung

- Client-Server-Architektur
- Ein Repository
- Streng hierarchisch
- Leicht Koordinierbar
- Parallele Entwicklung
- Beliebig viele Entwickler
- Beliebig viele Kopien



## Verteilte Versionsverwaltung

- Ein Repository pro Entwickler
- Beliebig viele Entwickler
- Beliebig viele Kopien
- Kein Lock – Modify – Write
- Parallele Entwicklung
- Komplexe Koordination
- Zentrale Repositories möglich



- Gliederung
- Motivation
- Allgemeines

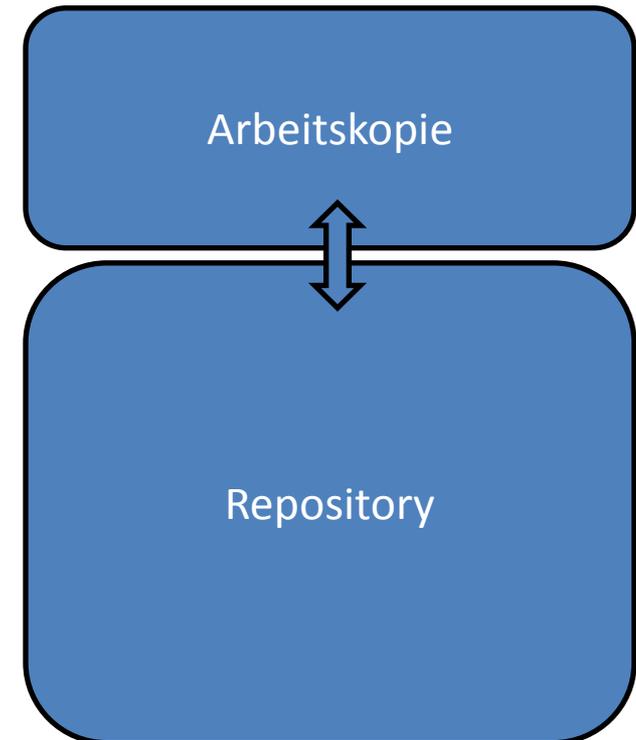
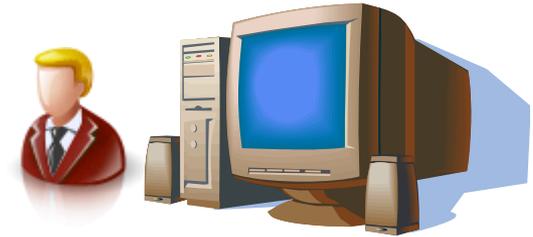
- Varianten der Versionsverwaltung
- Versionierungssysteme
- Git
- Zusammenfassung
- Diskussion

# Versionierungssysteme<sup>[BAE05]</sup>

Lokale Versionierungssysteme	Zentrale Versionierungssysteme	Verteilte Versionierungssysteme
SCCS (1972)	Concurrent Version System (CVS , 1986)	MS Visual SourceSafe (kommerziell, 1994)
SourceSafe ( => MS SourceSafe)	Subversion (SVN, 2004)	BitKeeper (kommerziell)
		Monotone
		Mercurial (hg)
		Git (2005)

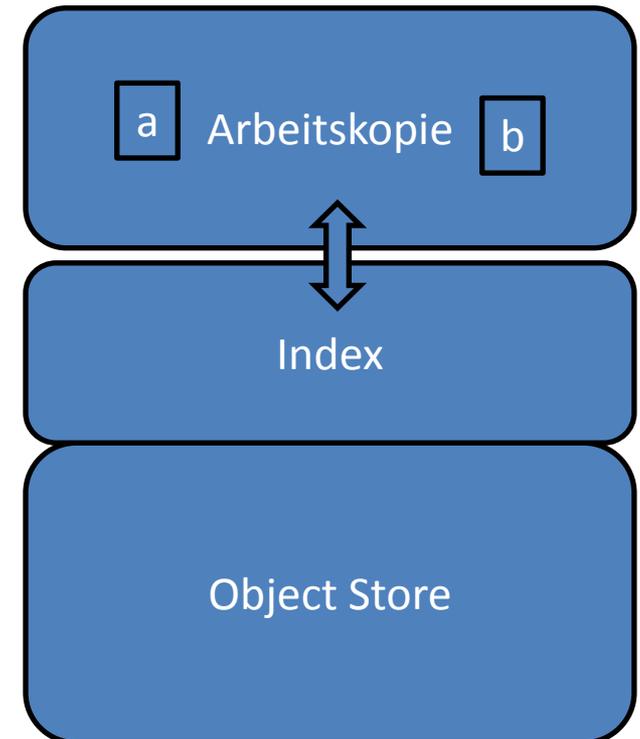
## Git als Versionierungssystem-Beispiel

- Eigenschaften von Git
  - Verteiltes Versionierungssystem
  - Copy – Modify – Merge
  - Zentrale Elemente:
    - Arbeitskopie
    - Repository



## Zentrale Elemente

- **Arbeitskopie**
  - Eine Datenkopie aus dem Repository
  - Datensatz in Bearbeitung
- **Repository**
  - Index
  - Object Store



# Repository

- Index

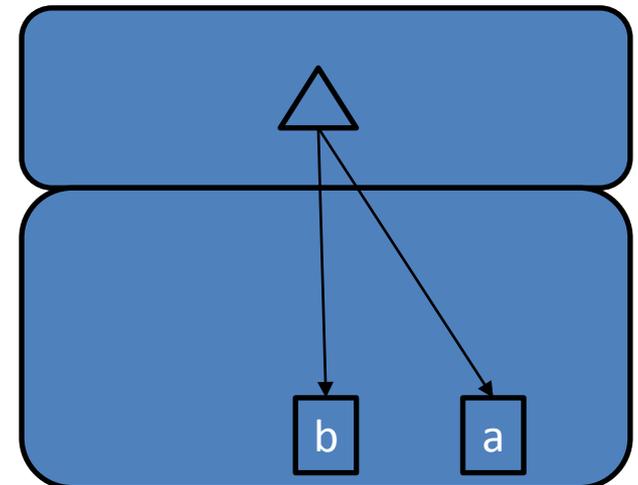
- *Staging Area:*

- Änderungen der Arbeitskopie im Repository speichern

- Schnittstelle zwischen Arbeitskopie und Repository

- Wesentliche Operationen

- Hinzufügen (*git add <file>*)
    - Löschen (*git rm <file>*)
    - Vergleichen (*git diff <--cached>*)

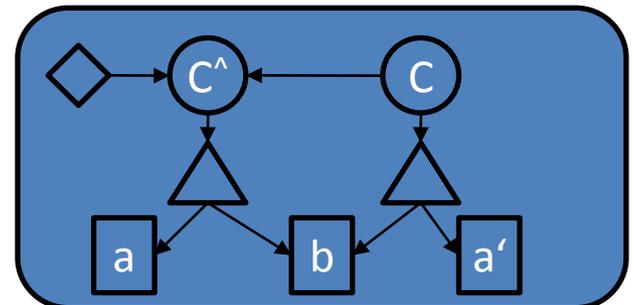


# Repository

- Object Store

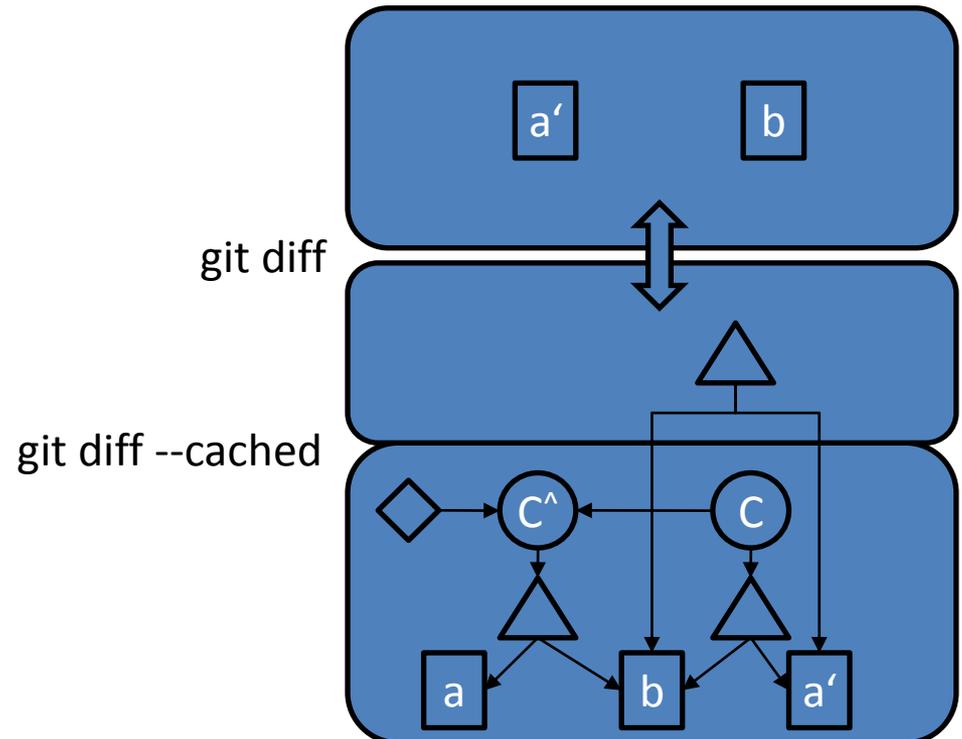
- Speichert Daten mit *git commit*
- Speichert Inhalte, keine Dateien (!= SVN)
- SHA1-Hash als Bezeichner
- Typen:

- Blobs 
- Trees 
- Commits 
- Tags 



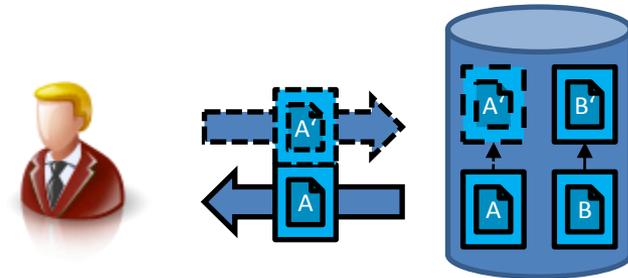
## Zwischenfazit

- Verteiltes Versionierungssystem
- Arbeitskopie
- Repository
  - Index
  - Object Store



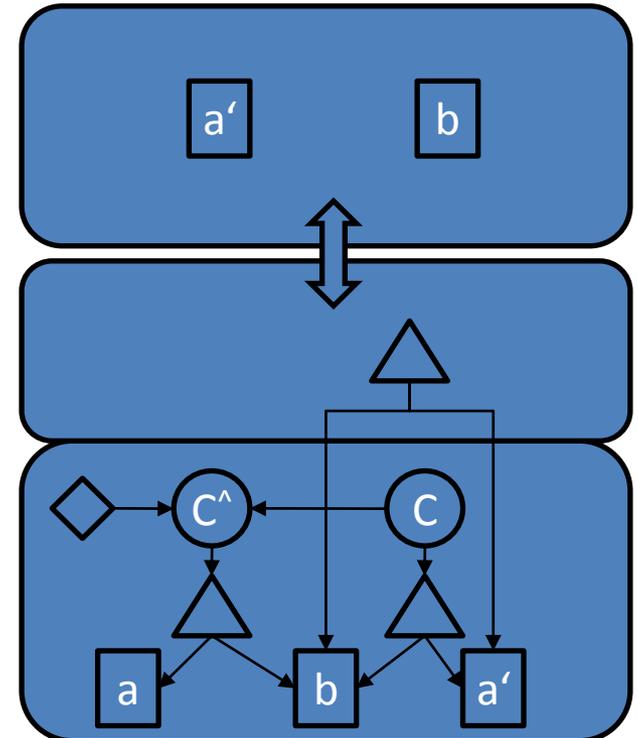
## Arbeitsweise - Wiederholung

- *Aktualisiere/Initialisiere/Klone* ein Repository
- Ändere Daten in der Arbeitskopie
- Aktualisiere das Repository (*commit*)
- Hilfe: `git help <cmd>`



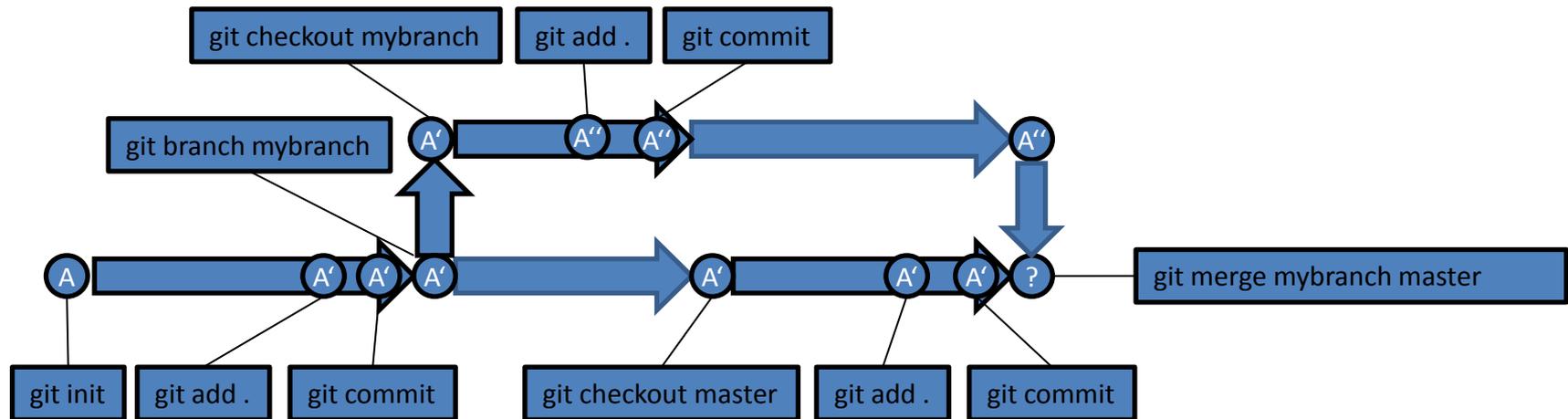
## Arbeitsweise - Spezifisch

- Initialisiere/Klone ein Repository
  - *git init* [*<Ziel>*] / *git clone* *<Quelle>* [*<Ziel>*]
- Arbeitskopie modifizieren
- Index aktualisieren (staging)
  - *git add* [*<Datei>* {*<Datei>*}| .]
- Repository aktualisieren
  - *git commit*



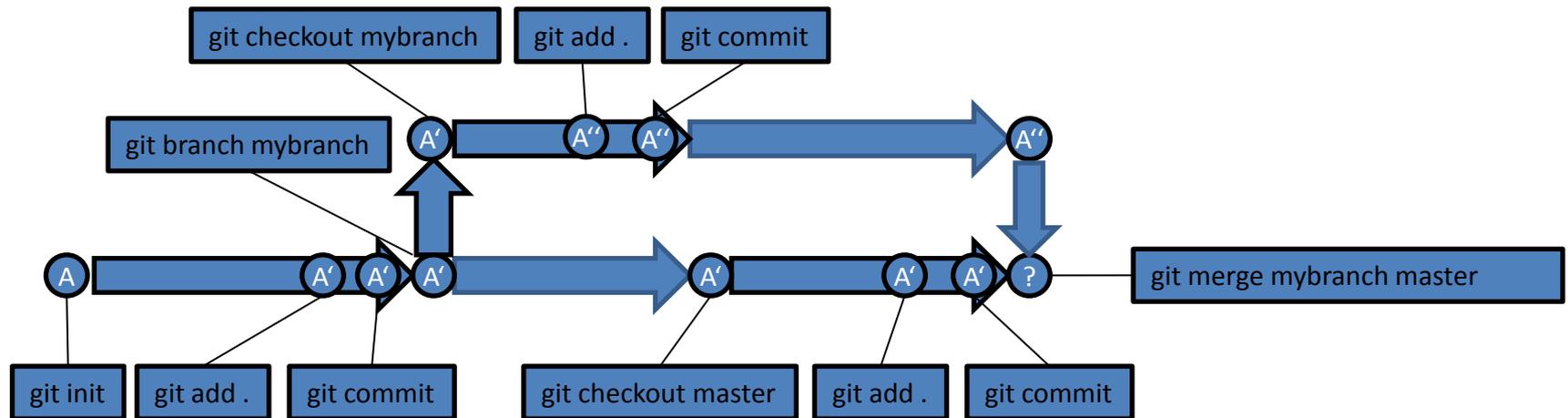
## Branching / Merging

- Das Repository als Graph
  - Commits und Branches als Knoten
- Beim Mergen ggf. Konflikte beheben
- Branch nur nach commit wechseln



# Anwendungsbeispiel

- Live-Anwendung



## Git Cheat Sheet

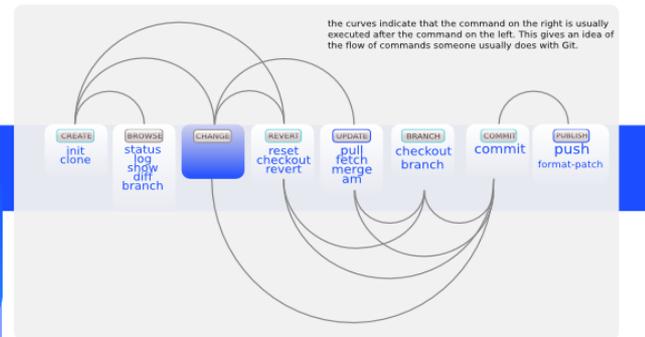
<http://git.or.cz/>

Remember: `git command --help`  
 Global Git configuration is stored in `$HOME/.gitconfig` (`git config --help`)

Create	Concepts
<b>From existing data</b> <code>cd ~/projects/myproject</code> <code>git init</code> <code>git add .</code>	<b>Git Basics</b> master : default development branch origin : default upstream repository HEAD : current branch HEAD~* : parent of HEAD HEAD~4 : the great-great-grandparent of HEAD
<b>From existing repo</b> <code>git clone ~existing/repo ~/new/repo</code> <code>git clone git://host.org/project.git</code> <code>git clone ssh://you@host.org/proj.git</code>	<b>Revert</b> Return to the last committed state <code>git reset --hard</code> <small>you cannot undo a hard reset</small>
<b>Show</b> Files changed in working directory <code>git status</code>	Revert the last commit <code>git revert HEAD</code> <small>Creates a new commit</small>
Changes to tracked files <code>git diff</code>	Revert specific commit <code>git revert \$id</code> <small>Creates a new commit</small>
What changed between \$ID1 and \$ID2 <code>git diff \$id1 \$id2</code>	Fix the last commit <code>git commit -a --amend</code> <small>(after editing the broken files)</small>
History of changes <code>git log</code>	Checkout the \$id version of a file <code>git checkout \$id \$file</code>
History of changes for file with diffs <code>git log -p \$file \$dir/ectory/</code>	<b>Branch</b> Switch to the \$id branch <code>git checkout \$id</code>
Who changed what and when in a file <code>git blame \$file</code>	Merge branch1 into branch2 <code>git checkout \$branch2</code> <code>git merge branch1</code>
A commit identified by \$ID <code>git show \$id</code>	Create branch named \$branch based on the HEAD <code>git branch \$branch</code>
A specific file from a specific \$ID <code>git show \$id:\$file</code>	Create branch \$new_branch based on branch \$other and switch to it <code>git checkout -b \$new_branch \$other</code>
All local branches <code>git branch</code> <small>(star * marks the current branch)</small>	Delete branch \$branch <code>git branch -d \$branch</code>

**Cheat Sheet Notation**  
 \$id : notation used in this sheet to represent either a commit id, branch or a tag name  
 \$file : arbitrary file name  
 \$branch : arbitrary branch name

### Commands Sequence



**Update**

Fetch latest changes from origin  
`git fetch` (but this does not merge them)

Pull latest changes from origin  
`git pull` (does a fetch followed by a merge)

Apply a patch that some sent you  
`git am -3 patch.mbox` (in case of a conflict, resolve and use git am --resolved)

**Publish**

Commit all your local changes  
`git commit -a`

Prepare a patch for other developers  
`git format-patch origin`

Push changes to origin  
`git push`

Mark a version / milestone  
`git tag v1.0`

**Useful Commands**

Finding regressions  
`git bisect start` (to start)  
`git bisect good $id` (\$id is the last working version)  
`git bisect bad $id` (\$id is a broken version)  
`git bisect bad/good` (to mark it as bad or good)  
`git bisect visualize` (to launch gitk and mark it)  
`git bisect reset` (once you're done)

Check for errors and cleanup repository  
`git fsck`  
`git gc --prune`

Search working directory for foo!  
`git grep "foo!"`

**Resolve Merge Conflicts**

To view the merge conflicts  
`git diff` (complete conflict diff)  
`git diff --base $file` (against base file)  
`git diff --ours $file` (against your changes)  
`git diff --theirs $file` (against other changes)

To discard conflicting patch  
`git reset --hard`  
`git rebase --skip`

After resolving conflicts, merge with  
`git add $conflicting file` (do for all resolved files)  
`git rebase --continue`

<http://zrusin.blogspot.com/2007/09/git-cheat-sheet.html>

## Zusammenfassung

- Zentrale und verteilte Versionierungssysteme
- Git als (gutes) Werkzeug für SIW
- Koordination bleibt wichtig
- Save early – save often
- Erstelle ein Versionierungskonzept...
- ...und halte es ein!!!

## Literatur

- [LOE09]: Jon Loeliger, 2009, „Version Control with Git“, O‘Reilly
- git-Webseite: <http://git-scm.com/>
- [BAE05]: Stefan Baerisch, 2005, „Versionskontrollsysteme in der Softwareentwicklung“, Informationszentrum Sozialwissenschaften



## Optional - Zentrale Versionsverwaltung in Git

### HowTo: Zentrale Versionsverwaltung

- Nutze ein Repository ohne Arbeitskopie
  - `git init --bare`
- Definiere das zentrale Repository
  - `git clone <bare> <Ziel>` oder
  - `git remote add origin <bare>`
- Synchronisiere Daten
  - `git push <origin [branch]>`
  - `git pull`