UNIVERSITÄT LEIPZIG
Wirtschaftswissenschaftliche Fakultät
Institut für Wirtschaftsinformatik

Versionskontrollsysteme

Versionskontrollsysteme (1)

- Als **Versionsverwaltung** (VCS) auch (Source Code Mngt.) wird ein System bezeichnet, welches zur Erfassung von Änderungen an Dokumenten oder Dateien Verwendung findet
- Dient der Sicherungen von meist textbasierten Dateien
- **Jede** erstellte **Version wird** mit eindeutiger *ID*, *Zeitstempel* und *Benutzerkennung des Autors* **gesichert**
- VCS enthalten Werkzeuge, um verschiedene Versionen mit einander zu vergleichen, zusammenzufassen, etc.
- Bekannte Vertreter:
  - CVS
  - SVN
  - **GIT**
  - Mercurial

Versionskontrollsysteme (2)

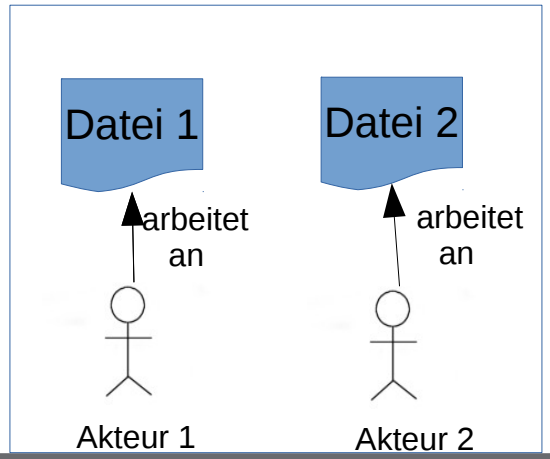
- Ein System mit dem Änderungen an Dateien verfolgt werden kann
  - „Eine Art Zeitmaschine“
  - Werkzeug, welches (nicht nur) in der Softwareentwicklung eingesetzt wird
  - Benutzer und Zeitstempel werden hinterlegt
- Anforderungen an das Versionsmanagement:
  - Änderungen am Produkt müssen über den gesamten Lebenszyklus nachvollziehbar sein
  - Konsistenz der Entwicklungsartefakte sicherstellen
  - Produkte (Versionen) müssen jederzeit reproduzierbar sein

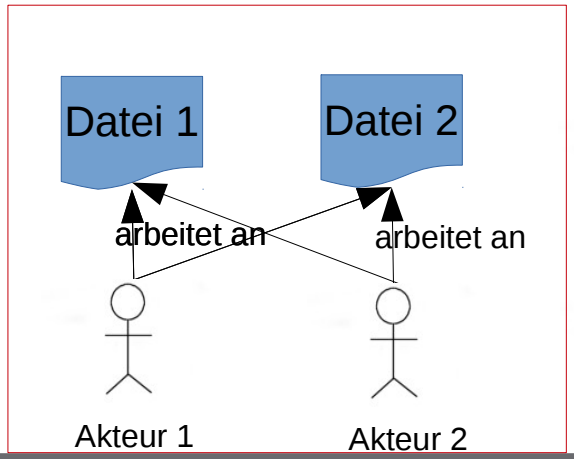
Arten von Versionsverwaltungen

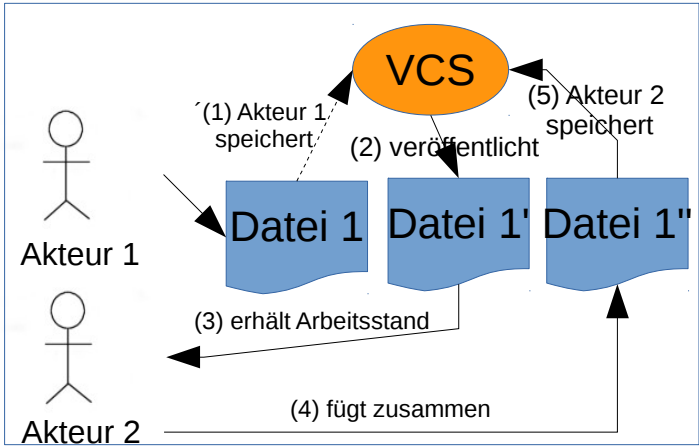
- Lokale Versionsverwaltung
  - Bswp. innerhalb von Eclipse „Show Local History“
- Zentrale Versionsverwaltung
  - Client-Server-Prinzip
  - Rechteverwaltung, Versionsgeschichte liegt beim Server
  - Beispiele: Concurrent Versions System (CVS) und Subversion (SVN), MS Team Foundation Server, …
- Verteilte Versionsverwaltung
  - Es gibt kein zentrales Repository (zumind. nicht zwangsweise)
  - Jeder Client hat sein eigenes Repository und kann dieses jedem anderen abgleichen
  - Verbreitet: GIT, Mercurial, Fossil, …

Nutzen im Arbeitsprozess

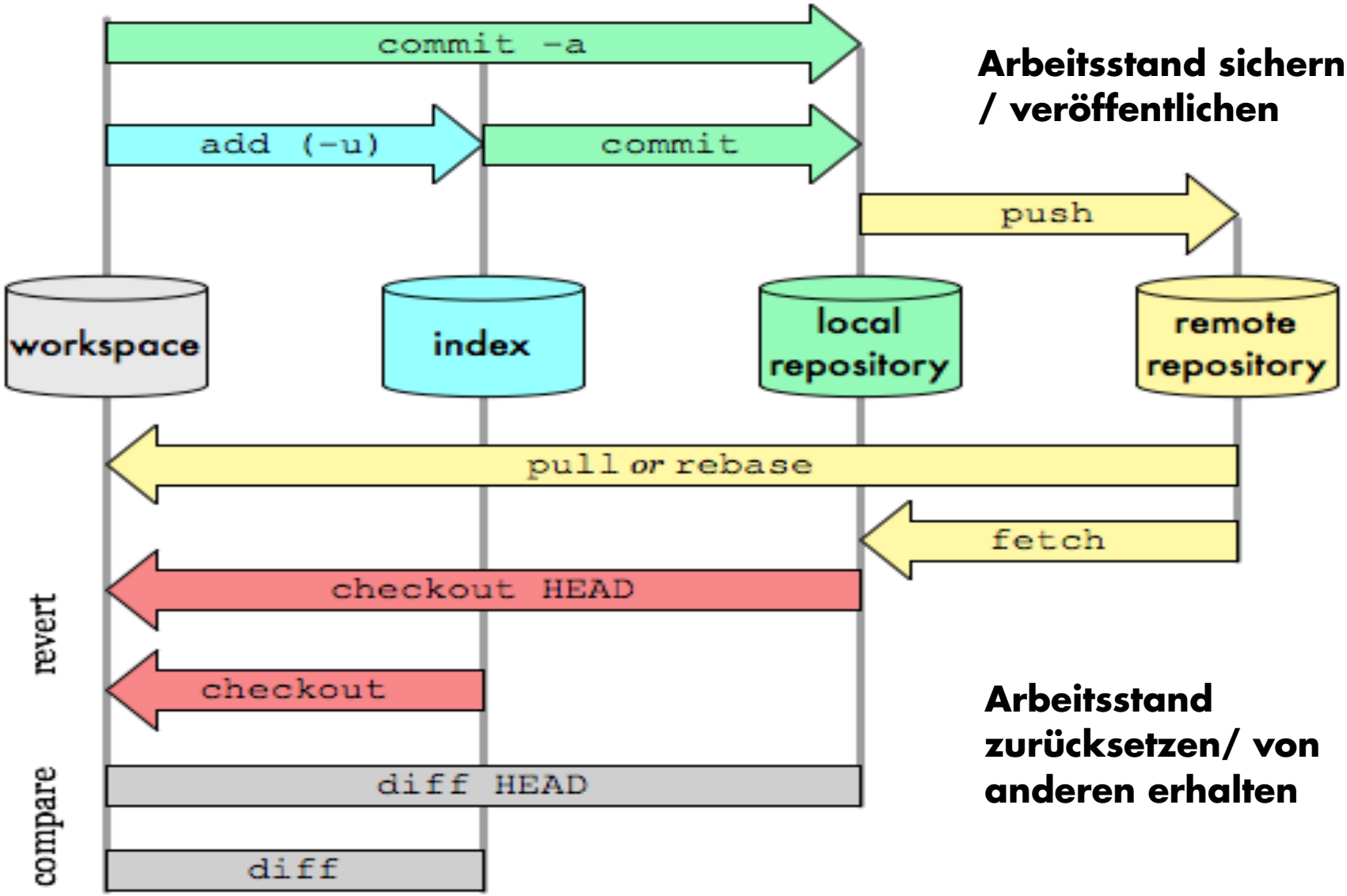
- Einfaches Wechseln zwischen verschiedenen Arbeitsständen
- Nutzer muss keinen (meist fehlerbehafteten) Versionierungsprozess selbst durchführen, d.h. Verhinderung von Änderungensanomalien
- Ablage aller Arbeitskopien durch zentrale Instanz (Server)
  → Sicherung
  → Verfügbarkeit
- Ermöglicht verlustfreie Parallelisierung im Arbeitsprozess:







Git: Architektur



Arbeit mit Git

- **Git repository** → Verwahrungsort für Commits (Lokal & Zentral)
- Repo initial „auschecken“ → **git clone REPOSITORY_URL**
- Auf neue Änderungen Prüfen → **git fetch**
- Änderungen machen
- Änderungen zur Sicherung auswählen → **git add DATEINAME**
- Arbeitsstand sichern (mit einer Nachricht) → **git commit**
- Arbeitsstand veröffentlichen → **git push**

  → Wenn nicht möglich, durch neue Änderungen im Remote-Repo:
  - **git pull**

    → Wenn Änderungen Kollidieren → Konflikt Manuell auflösen → **git commit** der Zusammenführung
  - **git push**
- **Bei großen Teams**: Arbeit in Featurebranches (git flow)

Git: Befehlsreferenz

- **clone** → erstmaliges fetchen eines Repositories
- **commit** → (lokale) Speicherung eines Arbeitsstandes
- **add** → Dateien zum Commit auswählen
- **branch** → Aneinanderreihung von Commits verwalten
- **push** → Upload von branches zu einem zentralem Repository
- **fetch** → Download von commits vom zentralen Repository
- **rebase** → Anfügung von Commits an Branch mit neuen Commits
- **merge** → Zusammenführung von branches / commits
- **pull** → fetch + merge
- **tag** → meta Information, um commit noch genauer auszuzeichnen
- **checkout** → Zu einem Branch/Commit/Tag wechseln
- **diff** → Vergleich zweier Commits/Branches

Git: Demo