

Programmierkonzepte in der Physikalischen Chemie

6. Versionsverwaltung

Albert-Ludwigs-Universität Freiburg



**UNI
FREIBURG**

Dr. Till Biskup

Institut für Physikalische Chemie
Albert-Ludwigs-Universität Freiburg
Wintersemester 2018/19



- ❏ Versionsverwaltung wirkt sich befreiend auf die Softwareentwicklung aus.
- ❏ Versionsverwaltung ist essentiell, um jederzeit eindeutig die verwendete Version einer Routine zu identifizieren.
- ❏ Verteilte Versionsverwaltungssysteme erleichtern die Verwendung und verringern externe Abhängigkeiten.
- ❏ Für ein größeres Projekt sollte ein klarer Arbeitsablauf festgelegt und konsequent befolgt werden.
- ❏ Versionsverwaltung hat (neben automatisierten Tests) den größten Einfluss auf die Art zu programmieren.

Motivation: Warum Versionsverwaltung?

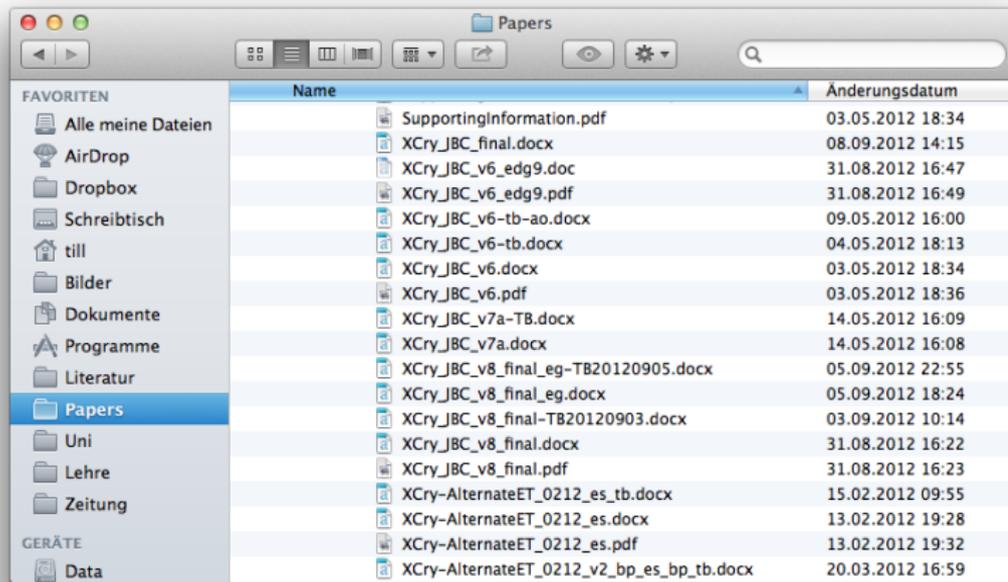
Übersicht über verschiedene Versionsverwaltungssysteme

Grundlegendes Arbeiten mit einer Versionsverwaltung

Strukturierung von Arbeitsabläufen

Motivation: Warum Versionsverwaltung?

Ein Beispiel aus dem realen Leben



The screenshot shows a Mac OS Finder window titled 'Papers'. The left sidebar shows the 'FAVORITEN' (Favorites) section with 'Papers' selected. The main pane displays a list of files with columns for 'Name' and 'Änderungsdatum' (Modification Date). The files listed are:

Name	Änderungsdatum
SupportingInformation.pdf	03.05.2012 18:34
XCry_JBC_final.docx	08.09.2012 14:15
XCry_JBC_v6_edg9.doc	31.08.2012 16:47
XCry_JBC_v6_edg9.pdf	31.08.2012 16:49
XCry_JBC_v6-tb-ao.docx	09.05.2012 16:00
XCry_JBC_v6-tb.docx	04.05.2012 18:13
XCry_JBC_v6.docx	03.05.2012 18:34
XCry_JBC_v6.pdf	03.05.2012 18:36
XCry_JBC_v7a-TB.docx	14.05.2012 16:09
XCry_JBC_v7a.docx	14.05.2012 16:08
XCry_JBC_v8_final_eg-TB20120905.docx	05.09.2012 22:55
XCry_JBC_v8_final_eg.docx	05.09.2012 18:24
XCry_JBC_v8_final-TB20120903.docx	03.09.2012 10:14
XCry_JBC_v8_final.docx	31.08.2012 16:22
XCry_JBC_v8_final.pdf	31.08.2012 16:23
XCry-AlternateET_0212_es_tb.docx	15.02.2012 09:55
XCry-AlternateET_0212_es.docx	13.02.2012 19:28
XCry-AlternateET_0212_es.pdf	13.02.2012 19:32
XCry-AlternateET_0212_v2_bp_es_bp_tb.docx	20.03.2012 16:59

Versionsverwaltung

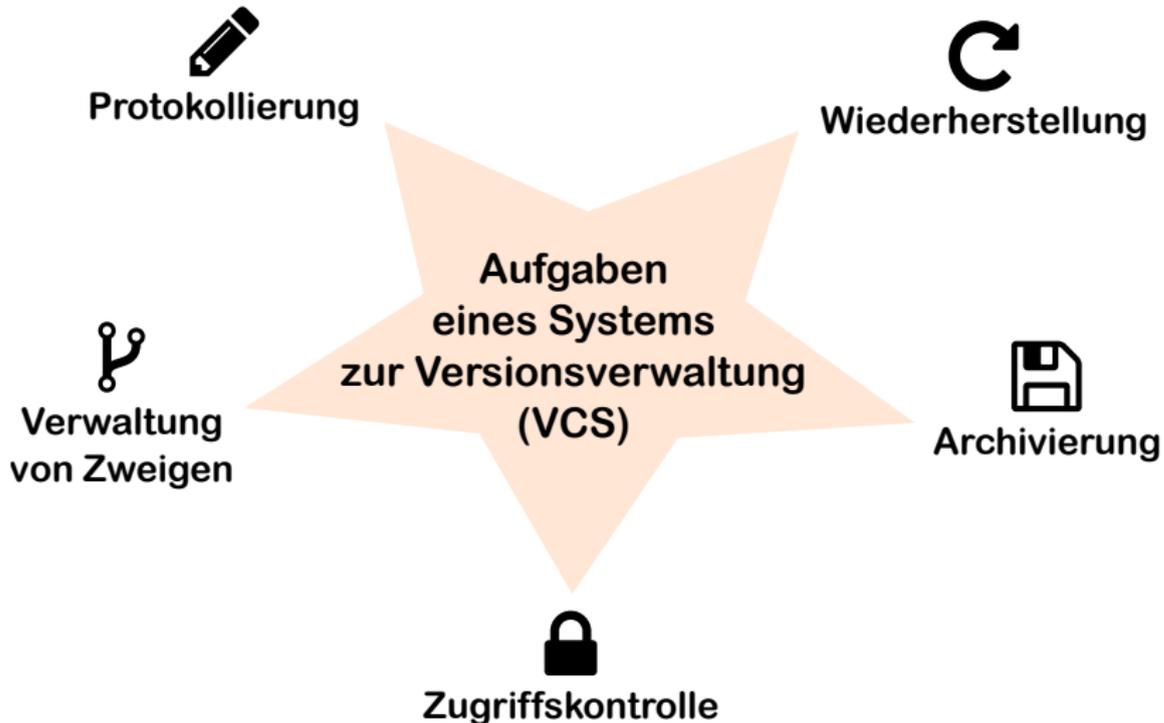
engl. *version control system* (VCS), System zur Erfassung von Änderungen an Dokumenten oder Dateien.

- ▶ System: automatisiert und strukturiert
 - ▶ Alle Versionen werden in einem Archiv gesichert.
 - ▶ jeweils mit Zeitstempel und Benutzerkennung
 - ▶ Jede Version kann später wiederhergestellt werden.
-
- ☞ typischer Einsatz: Softwareentwicklung
 - ☞ VCS hat erst einmal nichts mit Versionsnummern zu tun.

These

Rechnergestützte Arbeit im wissenschaftlichen Kontext ohne eine funktionierende Versionsverwaltung ist keine wissenschaftliche Arbeitsweise.

- ▶ Versionsverwaltung ersetzt das Laborbuch
- ▶ Zwingende Voraussetzung für die Nachvollziehbarkeit und Reproduzierbarkeit
- 👉 Versionsverwaltung ist nicht optional.



- ✎ Protokollierung der Änderungen
 - Wer hat wann was (warum) geändert?
- ↻ Wiederherstellung
 - beliebige alte Zustände einer Datei wiederherstellbar
 - (versehentliche) Änderungen rücknehmbar
- 📁 Archivierung
 - Zustände eines Projektes archiviert
 - bestimmter Zustand jederzeit wiederherstellbar
- 🔒 Zugriffskontrolle und -koordination
 - wichtig bei gemeinsamer Arbeit an einem Projekt
 - Zusammenführung unterschiedlicher Änderungen
- 🔗 Verwaltung mehrerer Zweige

Motivation: Warum Versionsverwaltung?

Notwendige Grundlage zentraler Programmierkonzepte

Warum Versionsverwaltung?

- ▶ Auswirkungen auf die Art des Programmierens
 - befreiend: funktionierende Vorversion immer erreichbar
 - strukturierend: klarer Verweis auf eine Version möglich
- ▶ Unumgänglich für verteilte Programmierung
 - Zusammenführung unterschiedlicher Änderungen
 - Verantwortliche für Änderungen nachvollziehbar

Warum Versionsverwaltung als einzelner Nutzer?

- ▶ Voraussetzung für die Nachvollziehbarkeit und Reproduzierbarkeit (Dokumentationspflicht)
- ▶ Historie einer Entwicklung verfügbar
- ▶ Auswirkungen auf die Art des Programmierens (s.o.)
- 👉 Voraussetzung für zentrale Programmierkonzepte

Motivation: Warum Versionsverwaltung?

Positiver Einfluss auf die Art des Programmierens



These

Versionsverwaltung kann – richtig eingesetzt – die Art, wie wir programmieren, nachhaltig (positiv) verändern.

- ▶ immer in kleinen Abschnitten programmieren
- ▶ Version sichern, bevor Änderungen vorgenommen werden
- ▶ Keine neue Version, wenn noch auskommentierte Teile im Code stehen (Ausnahmen nur gut begründet)
- ☛ Arbeiten mit Versionskontrolle ist eine Frage der Einstellung und der Routine.

Lokale Versionsverwaltung

- ▶ oft Versionierung nur einer Datei
- ▶ heute noch in Büroanwendungen (z.B. Word & Co.)

Zentrale Versionsverwaltung

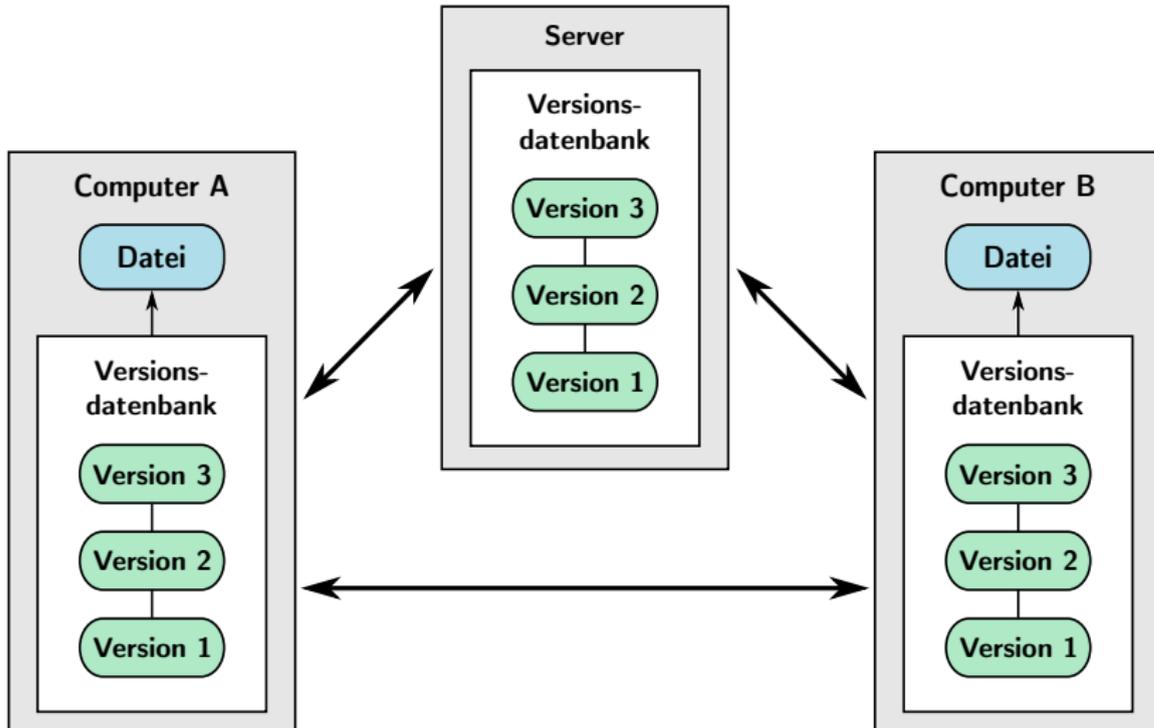
- ▶ Server-Client-Konzept
- ▶ Rechteverwaltung beschränkt Zugriff
- ▶ Versionsgeschichte nur an einem Speicherort

Verteilte Versionsverwaltung

- ▶ kein zentraler Speicher, dezentrale Versionsgeschichte
- ▶ Jeder pflegt eigenen Speicherort mit ei(ge)ner Historie.

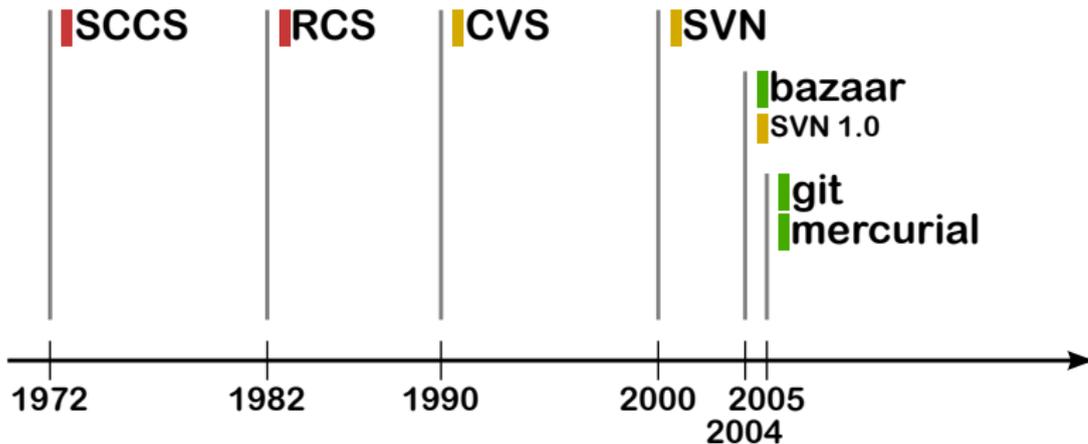
Verschiedene Versionsverwaltungssysteme

Arten von Versionsverwaltungssystemen





Einige freie Versionsverwaltungssysteme



lokal



zentral (Server-Client)



verteilt

Grundbegriffe der Versionsverwaltung

Repository (zentraler) Speicherort der versionierten Dateien

Arbeitskopie lokale Kopie einer Revision eines Repositorys

Revision einzelner der Versionsverwaltung bekannter Stand

Branch Zweig, Abspaltung von einer anderen Version

Branches können parallel weiterentwickelt werden

Zusammenführen von Branches möglich

Tag frei wählbarer Bezeichner für eine Revision

z.B. nach außen kommunizierte Versionsnummer

Grundlegende Operationen

- checkout Holen einer Version aus dem Repository
- update Aktualisierung der lokalen Arbeitskopie
- commit Übertragen einer Version in das Repository
- diff Vergleich zweier Versionen
- merge Zusammenführen unterschiedlicher Versionen

Operationen bei verteilten Versionsverwaltungssystemen

- push Übertragen einer Version in ein anderes (ggf. entferntes) Repository
- pull Holen einer Version aus einem anderen (ggf. entfernten) Repository

Grundlegender Arbeitsablauf

- 1 auf Änderungen überprüfen (update/fetch/pull)
- 2 lokal Änderungen vornehmen
- 3 neue Version speichern (add/commit/push)

Anmerkungen

- ▶ erster Schritt entfällt bei rein lokaler Nutzung
 - ansonsten erster Schritt vor dem Ändern von Code
 - bei zentralen VCS hilfreich vor jedem commit
- ▶ neue Version spätestens am Ende des Arbeitstages
 - „Commit early, commit often“
 - idealerweise immer nur funktionierenden Code committen

Umgang mit Veränderungen: diff und merge

- ▶ automatisches Zusammenführen
 - bei sich nicht überschneidenden Änderungen am Code
- ▶ manuelle Zusammenführung der Änderungen
 - bei parallelen inkompatiblen Änderungen
 - Unterstützung durch entsprechende Werkzeuge

Zweige: unabhängige Parallelentwicklungen

- ▶ Entwicklung neuer Features
- ▶ Pflege veröffentlichter Versionen
- ☞ Änderungen über Zweige hinweg übernehmbar

Gründe für Branching (und Merging)

- ▶ größere Änderungen in eigenen Branches
 - entkoppelt die Entwicklung neuer Features
- ▶ Trennung von Entwicklung und Produktion
 - stabile Produktivversion jederzeit verfügbar
 - Entwicklung befreit vom Zwang funktionierender Versionen
 - Datenauswertung nicht mit Entwicklerversionen
- ▶ Pflege von Versionen nach Veröffentlichung
 - Software ist selten fehlerfrei
 - evtl. mehrere Versionen parallel in aktiver Nutzung
- ▶ Zugriffs- und Qualitätskontrolle
 - Schreibrechte auf Hauptzweig ggf. eingeschränkt
 - klare Kriterien für Codequalität in bestimmten Zweigen

unabhängig von zentraler Infrastruktur

- ▶ kein zentraler Server notwendig
- ▶ keine Internetverbindung notwendig

Commits/Branches sind „billig“

- ▶ „Commit early, commit often“
 - befreiend für die Entwicklung
 - je häufiger, desto enghemmer das Sicherheitsnetz
 - fördert schrittweises Vorgehen bei der Entwicklung
- ▶ einfaches Branching und Merging
 - neue Features in eigenem Branch
 - gefahrloses „Rumspielen“, ohne etwas kaputt zu machen

Argumente für die Strukturierung

- ▶ Projekte von Einzelkämpfern
 - automatisches Setzen von Tags (z.B. mit Versionsnummer)
 - automatisches Inkrementieren der Versionsnummer
 - Feature-Branched auch hier hilfreich
 - „Commit early, commit often“

- ▶ Projekte mit mehreren Entwicklern
 - klare Verantwortlichkeiten

- ▶ Projekte mit einer größeren Nutzerbasis
 - Schema für die Verwendung von Branches
 - klare Verantwortlichkeiten (Zugriff auf Branches, ...)
 - parallele Pflege mehrerer Programmversionen

Bausteine einer Strukturierung und Automatisierung

- ▶ Aufbau von Commit-Kommentaren
 - kurz und prägnant formulieren
 - ggf. Aufteilung in Betreffzeile und (längere) Beschreibung
- ▶ Nutzung von Feature-Branches für Entwicklungen
 - kurzlebig und nur für *ein* Feature
- ▶ „Commit early, commit often“
 - mindestens am Ende eines jeden Arbeitstages
- ▶ nur lauffähigen und getesteten Code committen
 - besonders bei zentralen Branches und Produktivversionen
- ▶ Schema zur Inkrementierung von Versionsnummern
 - Versionsnummer als Tag ablegen

git flow und Alternativen

- ▶ bekannte(re) Schemata:
 - git flow, github flow, gitlab flow
 - zumindest git flow mit Kommandozeilen-Unterstützung
- ▶ strukturierte/automatisierte Arbeitsabläufe
 - Erzeugen/Zusammenführen von Feature-Branches
 - Veröffentlichung von Releases (inkl. Versionsnummern)
 - Hotfixes
- ☞ Diskussion von Vor-/Nachteilen führte zu weit
- ☞ Wichtig: konsistent einem Schema folgen
- ☞ Automatisieren, was automatisierbar ist



- ❏ Versionsverwaltung wirkt sich befreiend auf die Softwareentwicklung aus.
- ❏ Versionsverwaltung ist essentiell, um jederzeit eindeutig die verwendete Version einer Routine zu identifizieren.
- ❏ Verteilte Versionsverwaltungssysteme erleichtern die Verwendung und verringern externe Abhängigkeiten.
- ❏ Für ein größeres Projekt sollte ein klarer Arbeitsablauf festgelegt und konsequent befolgt werden.
- ❏ Versionsverwaltung hat (neben automatisierten Tests) den größten Einfluss auf die Art zu programmieren.