

Git versus SVN

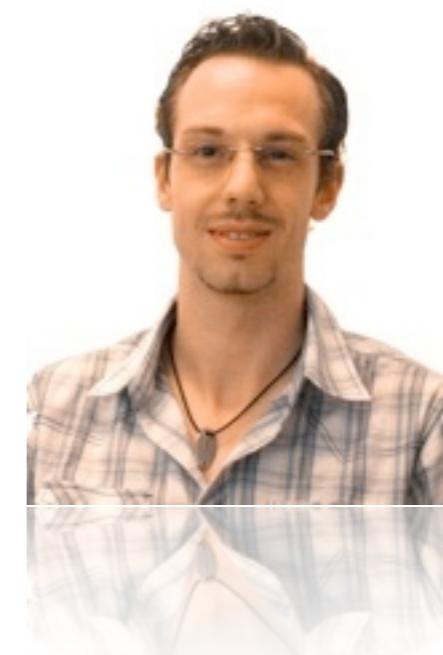
Eine vergleichende Einführung in verteilte Versionskontrollsysteme (VCS) anhand von „Git“ und dem zentralisierten VCS „Subversion“ (SVN)

http://bit.ly/PHPUG_JUN_GITvsSVN



Wer bin ich?

- ⬢ Mario Müller
- ⬢ TWT Interactive GmbH - Düsseldorf
- ⬢ PHP (ZCE), Javascript, Python, Java
- ⬢ Webservices, verteilte Systeme, Build Systeme, Frameworks
- ⬢ MySQL, Postgresql, CouchDb, MongoDb
- ⬢ Mac-Head & Linux Enthusiast
- ⬢ Xing: <http://bit.ly/mariomueller>
- ⬢ Twitter: <http://twitter.com/xenji>
- ⬢ Github: <http://github.com/xenji>





Agenda

- ⬢ Versionierung
- ⬢ Warum?
- ⬢ Lokal
- ⬢ Zentral
- ⬢ Dezentral





Agenda

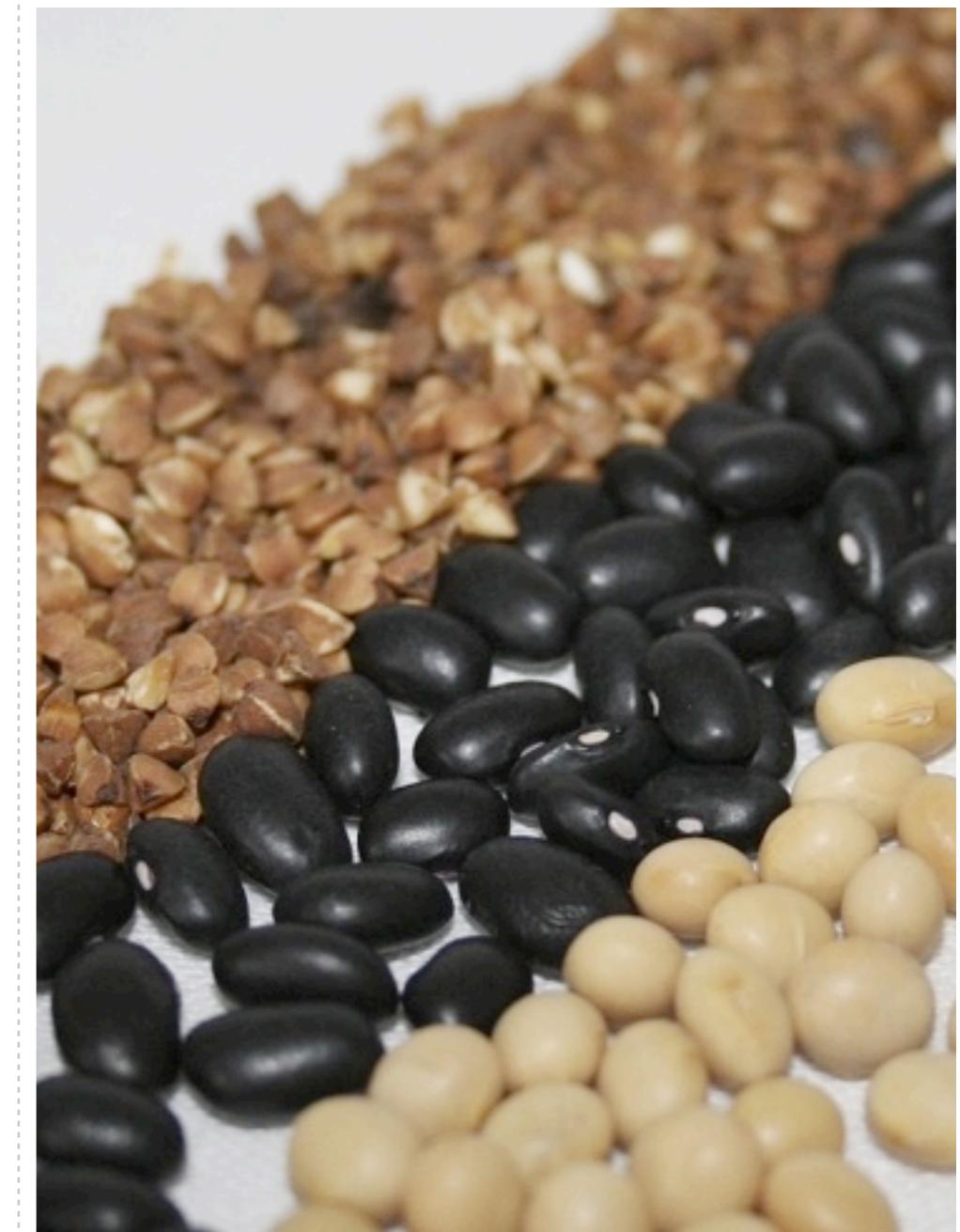
- ➊ SVN
 - ☰ Geschichte
 - ☰ Begriffe
 - ☰ Funktionsweise
 - ☰ Vorteile & Einschränkungen
 - ☰ Alleinstellungsmerkmale





Agenda

- Git
 - Geschichte
 - Begriffe
 - Funktionsweise
 - Vorteile & Einschränkungen
 - Workflow & Team Organisation





Agenda

- ⬢ Gegenüberstellung Vorteile & Nachteile
- ⬢ Fazit
- ⬢ Fragen
- ⬢ Quellen







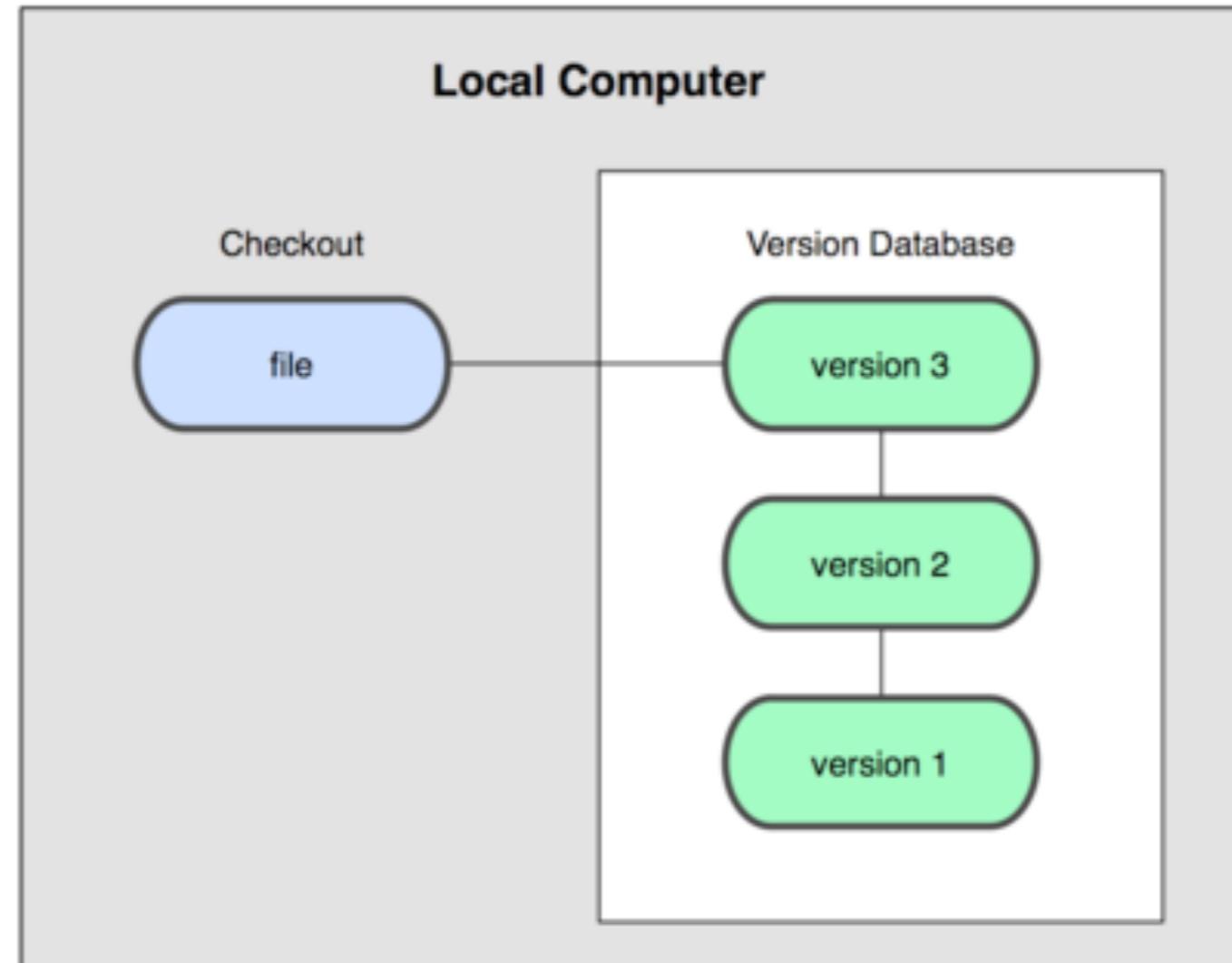
#1 Versionierung



Warum Versionierung?

- ❖ Protokollierung
- ❖ Archivierung
- ❖ Wiederherstellung





Lokale Versionierung



Merkmale

- ⬢ Alle Arbeiten sind lokal
- ⬢ Es gibt immer nur eine Realität
- ⬢ Die Versionshistorie ist lokal
- ⬢ Dateien werden als Ganzes versioniert
- ⬢ Vergleiche sind möglich
- ⬢ Performanz ist gut

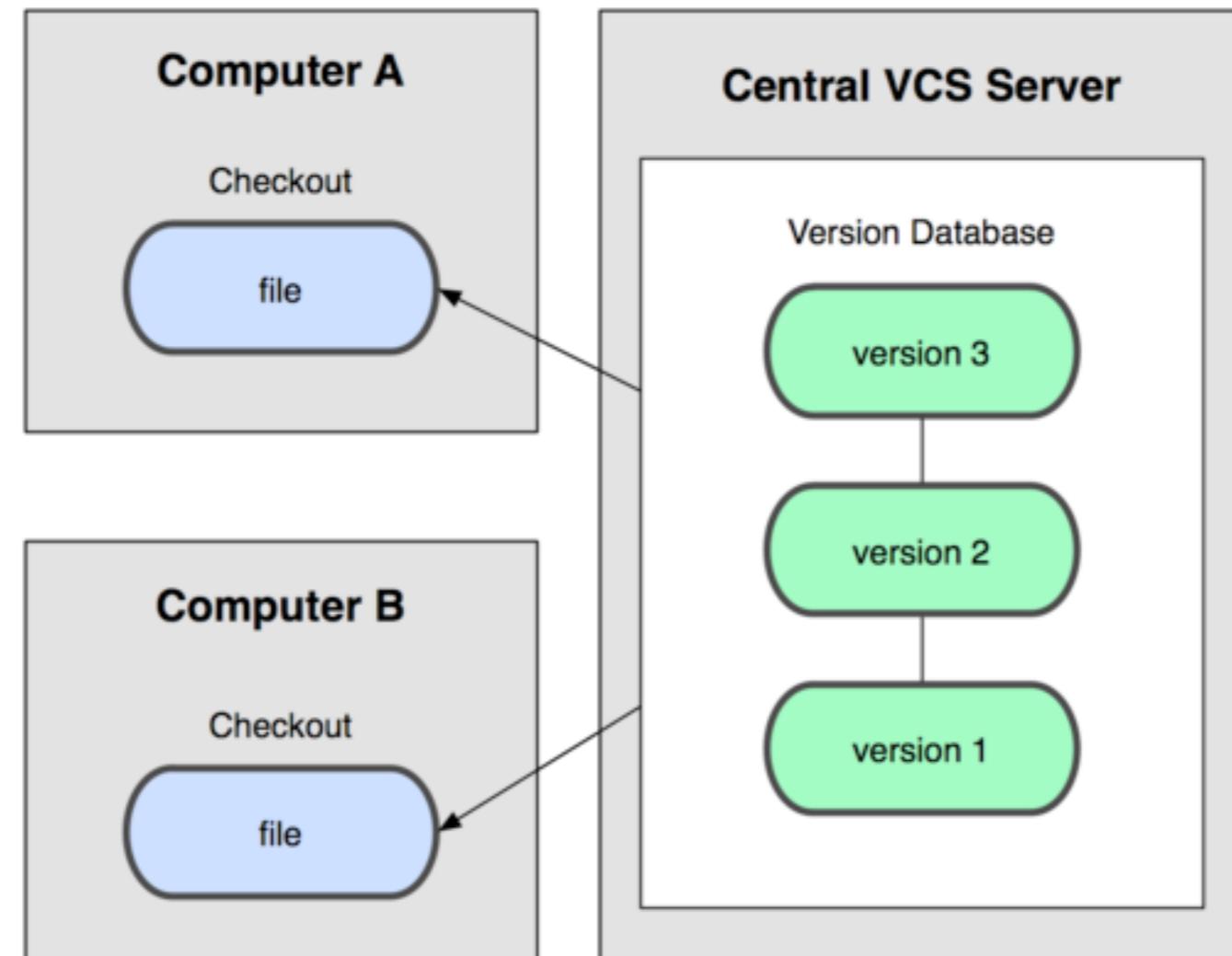


RCS - Revision Control System

- ❖ Dateibasiertes Unix & Linux VCS
- ❖ Sehr alt (Anfang der 1980er) und Vorgänger von CVS, welches immer noch die gleiche Methode zur Dateiverwaltung verwendet
- ❖ Wird heute teilweise noch von Sys-Admins eingesetzt um Konfigurationen zu sichern
- ❖ Speichert immer den gesamten Dateistand (kein Delta).

```
$ ci -u PyChart-1.26.1.tar.gz
PyChart-1.26.1.tar.gz,v <-- PyChart-1.26.1.tar.gz
enter description, terminated with single '.' or
end of file:
NOTE: This is NOT the log message!
>> Init Checkin
>> .
initial revision: 1.1
done
```





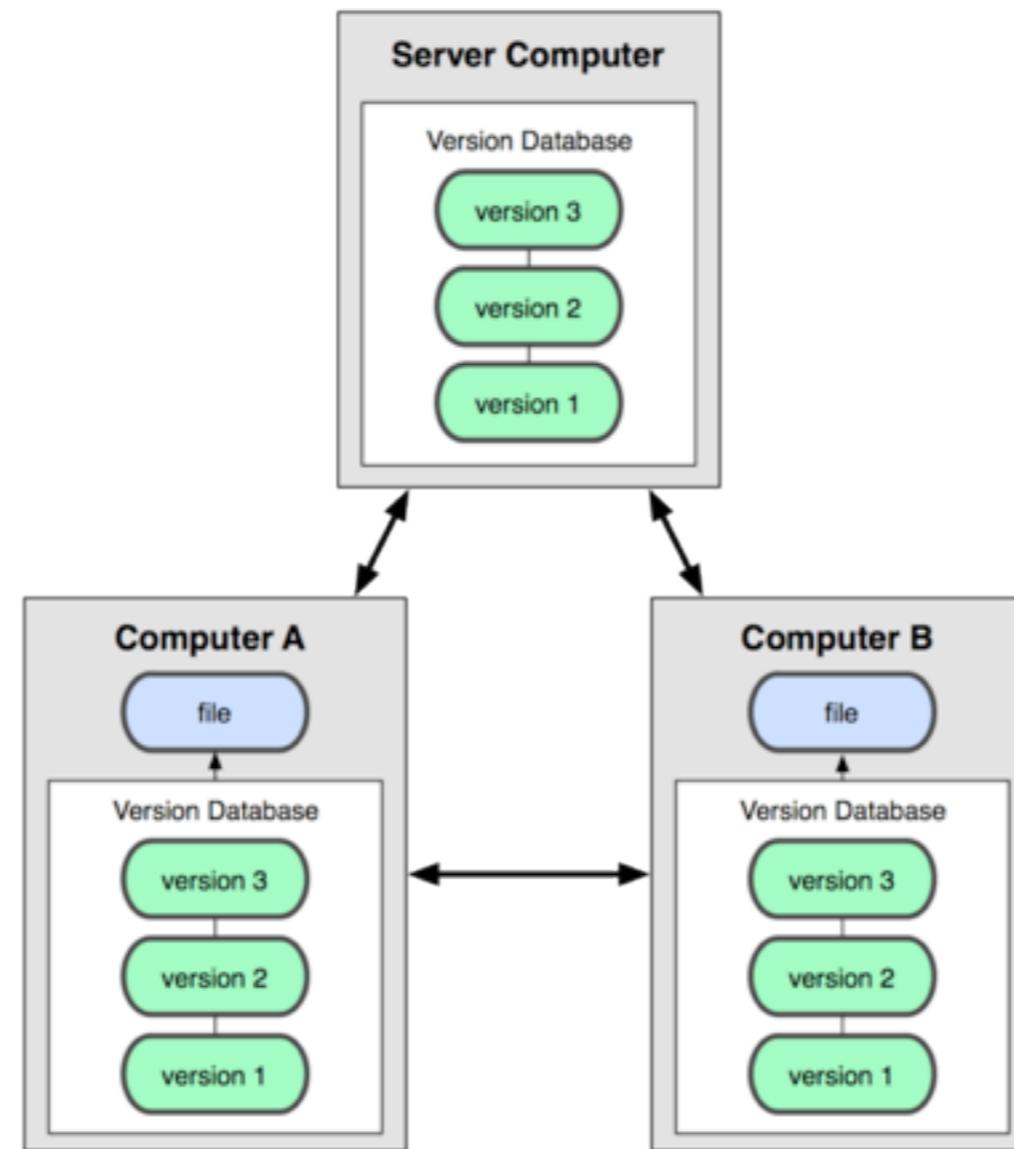
Zentrale Versionierung



Merkmale

- ⬢ Es gibt mehr als eine Realität (ein Server, n Workingcopies)
- ⬢ Revisionen werden zentral verwaltet, Versionsnummern zentral vergeben
- ⬢ Vergleiche sind nur direkt mit dem Server möglich
- ⬢ Häufig wird ein Delta-basiertes Speicherverfahren verwendet, so bleiben die zu übertragenden Mengen gering
- ⬢ Die Versionshistorie ist nur auf dem Server verfügbar
- ⬢ Die Zentralisierung ermöglicht ein Zugriffs- und Rechtemanagement





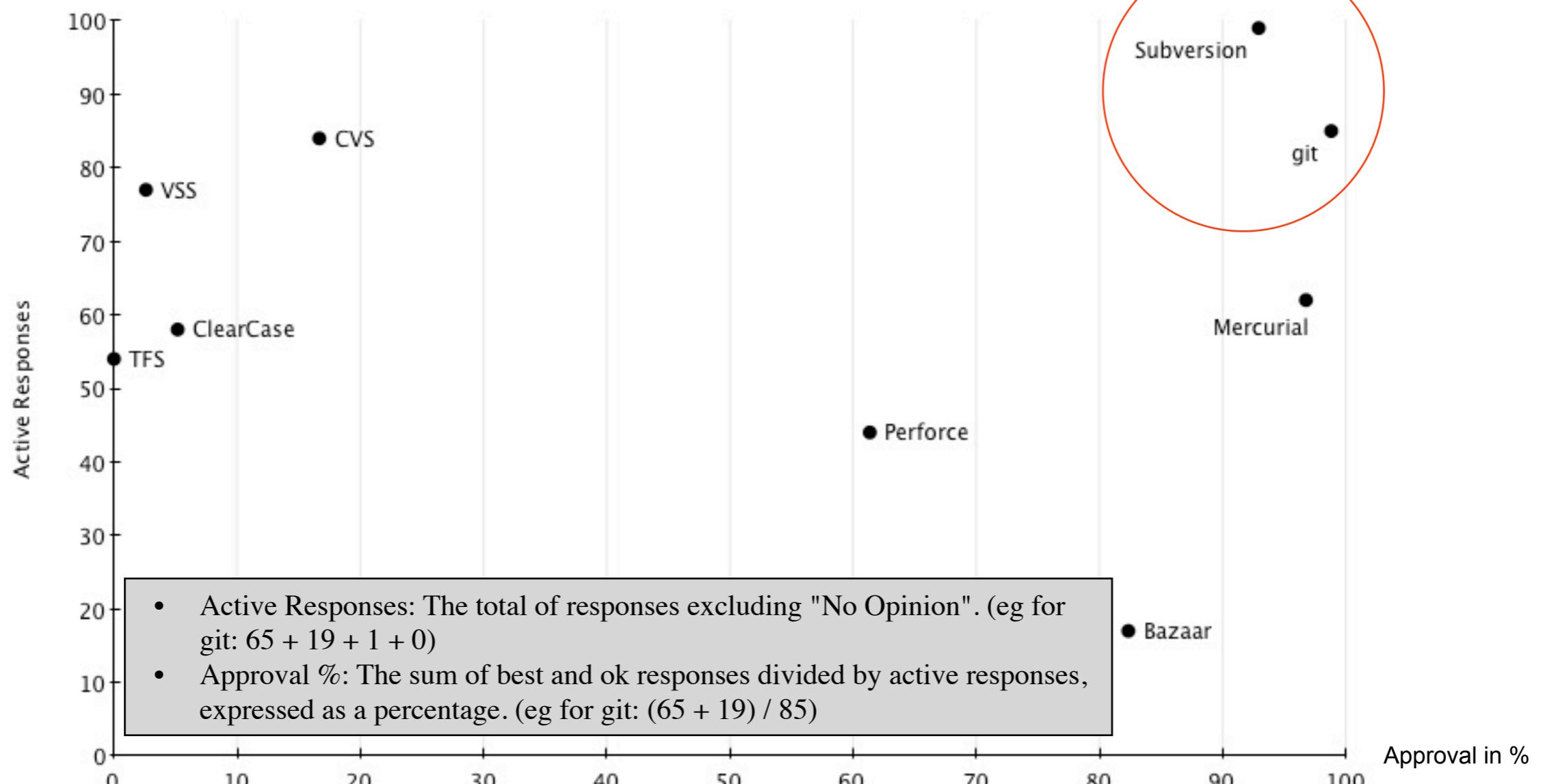
Dezentrale Versionierung



Merkmale

- ⬢ Es gibt viele, mehrdimensionale Realitäten (Multi-Master, Multi-Workingcopy)
- ⬢ Jede Workingcopy ist ein kompletter Klon mit allen Versionen
- ⬢ Theoretisch gibt es keinen zentralen Server
- ⬢ Das Repository ist lokal und unabhängig
- ⬢ Alle Operationen sind lokal
- ⬢ Es ist ein Mechanismus zur Synchronisierung mit einer entfernten Instanz vorhanden





VCS Survey (von M. Fowler)



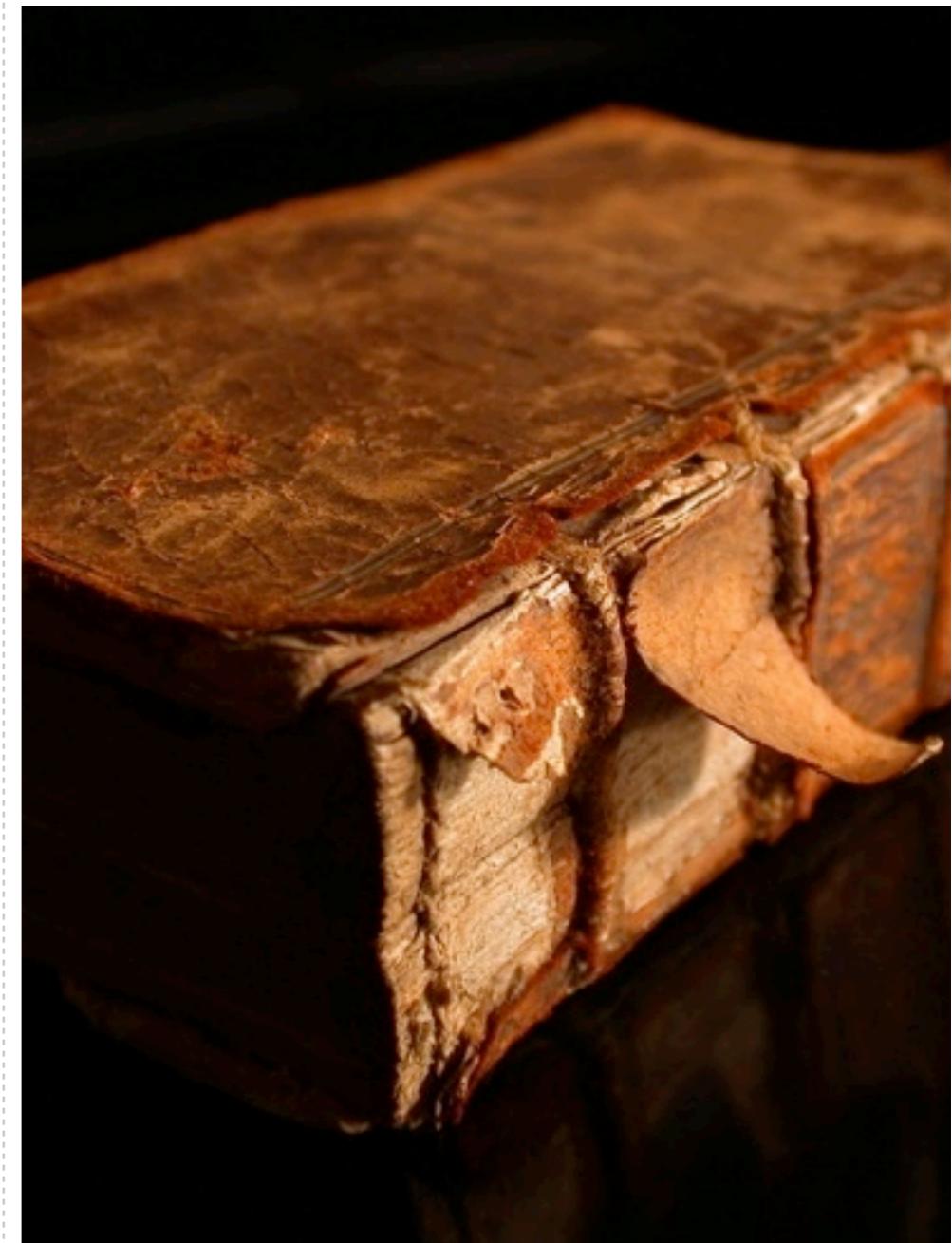


#2 SVN - Subversion



Geschichte

- ⬢ 4 Jahre Entwicklung - Version 1.0 am 23. Feb. 2004
- ⬢ Entwickelt von CollabNet
- ⬢ Seit dem 10. Feb. 2010 ein Apache Top-Level Projekt
- ⬢ Weiterentwicklung vom ebenfalls zentralen Versionierungstool „CVS“



Begriffe

Changeset

Revision

Delta / Diff

Merge

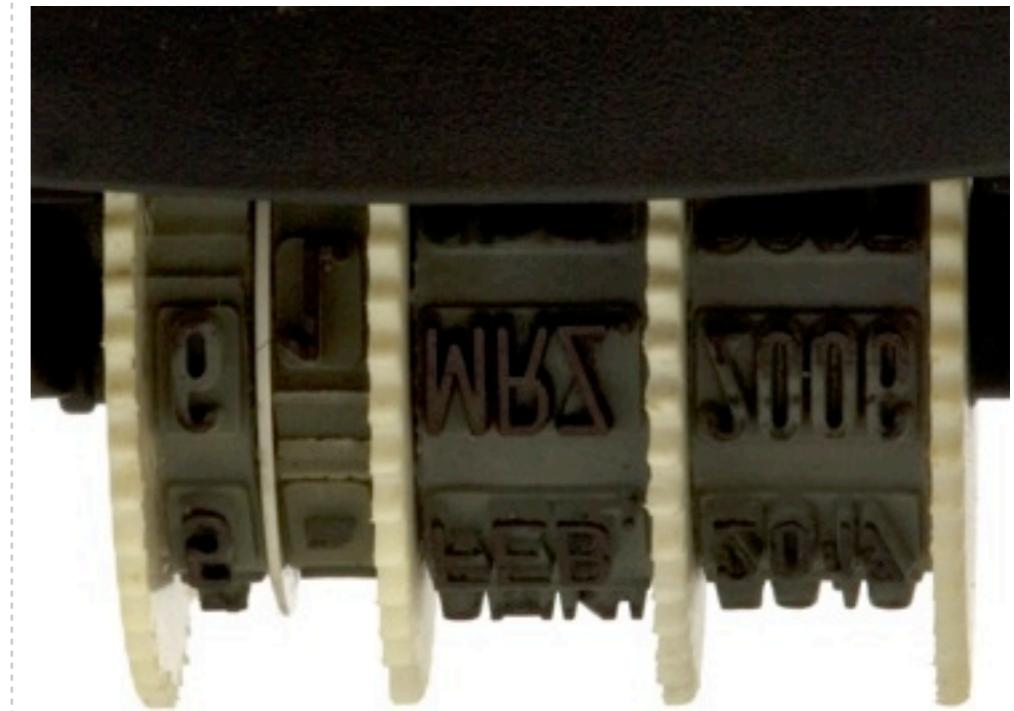
Branch

Tag



Revision

- im Bereich der elektronischen Archivierung, insbesondere im Rahmen der Archivierung kaufmännischer Daten, für Nachprüfbarkeit, Unveränderbarkeit, Nachvollziehbarkeit (Wikipedia)



Changeset

- ⬢ Eine Zusammenfassung von Änderungen einer Version. Häufig wird eine Notation verwendet, die die Operation mit / auf dem Bestandteil des Changesets erklärt:
 - ⬢ U = Updated
 - ⬢ D = Deleted
 - ⬢ A = Added
- ⬢ Beispiel für ein Changeset:

U foo.txt
A bar.sh
D baz.php



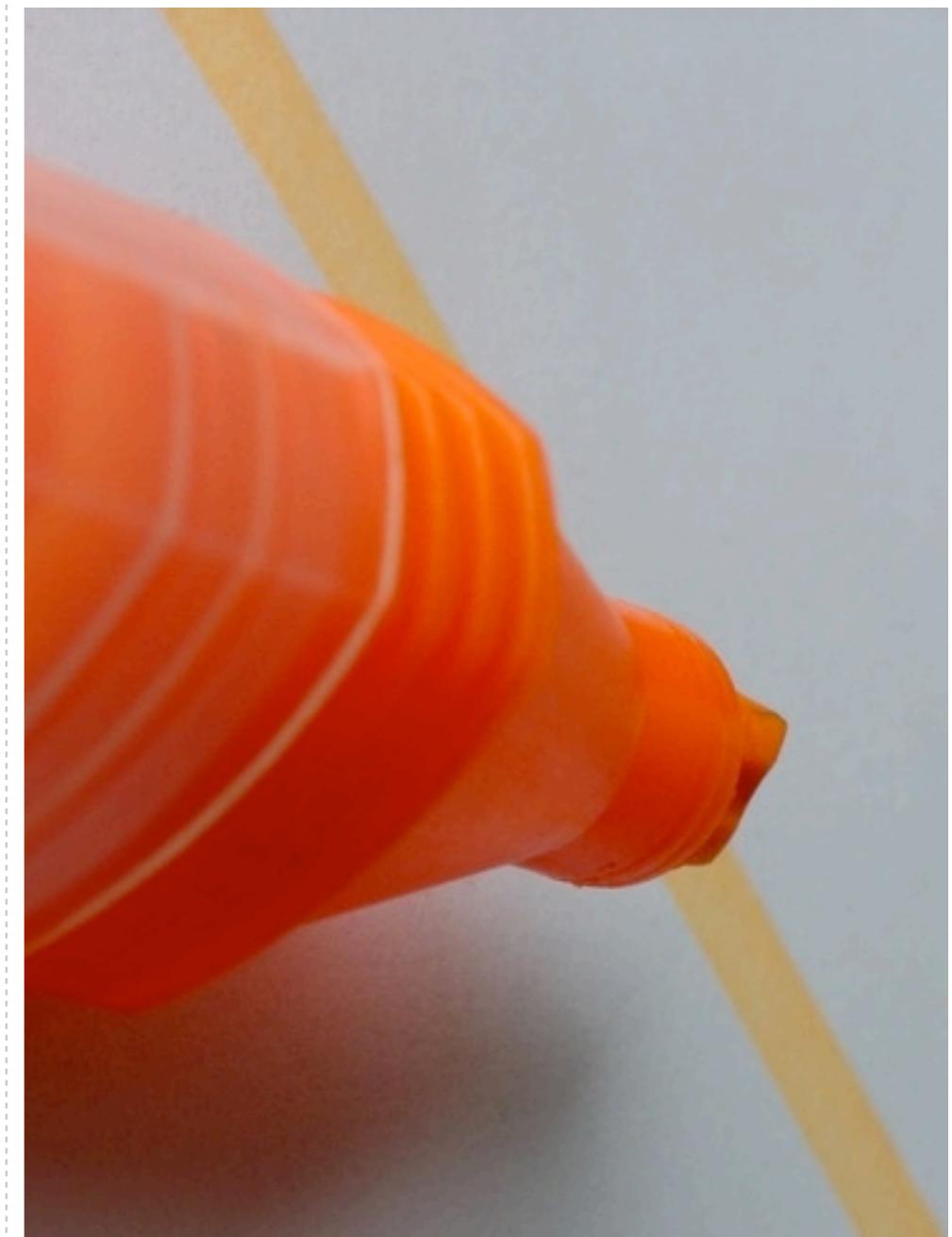
Delta / Diff

- ➊ Das griechische Delta (Δ) wird häufig für die Benennung von Differenzen verwendet.
- ➋ Im Fall von SVN handelt es sich sogar um das angewandte Speicherverfahren.
- ➌ Es werden immer nur die Unterschiede zwischen zwei Versionen festgehalten.



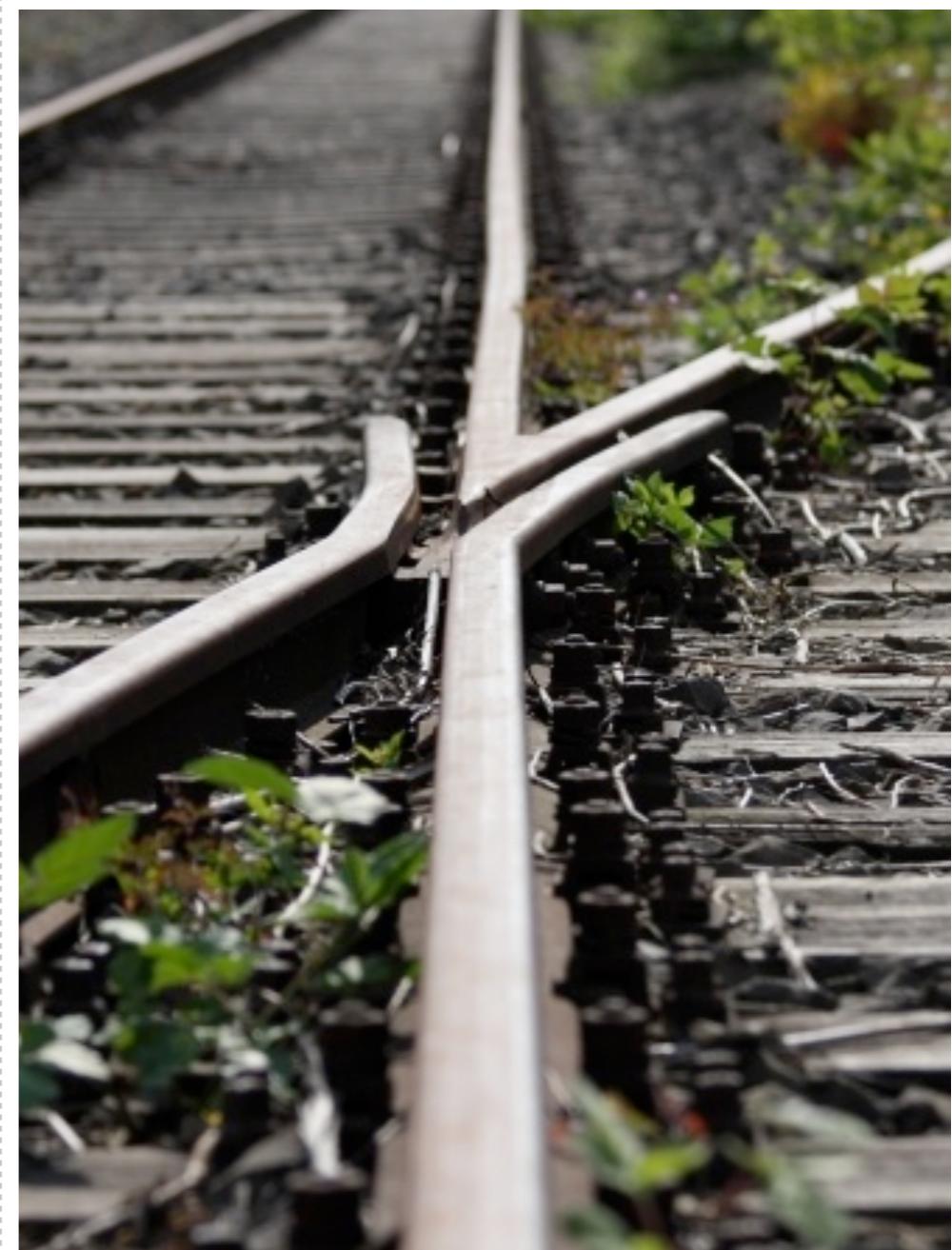
Tag

- ◉ Ein Tag ist eine Beschriftung einer bestimmten, einzelnen Revision. Man markiert einen definierten Stand mit einem Zeiger. Das Taggen eine „günstige“, wenig Ressourcen - verbrauchende Operation.
- ◉ Häufig wird das Taggen für die Definition von Versionen verwendet. Tags werden als unveränderlich betrachtet, sind es aber de-facto in SVN nicht.



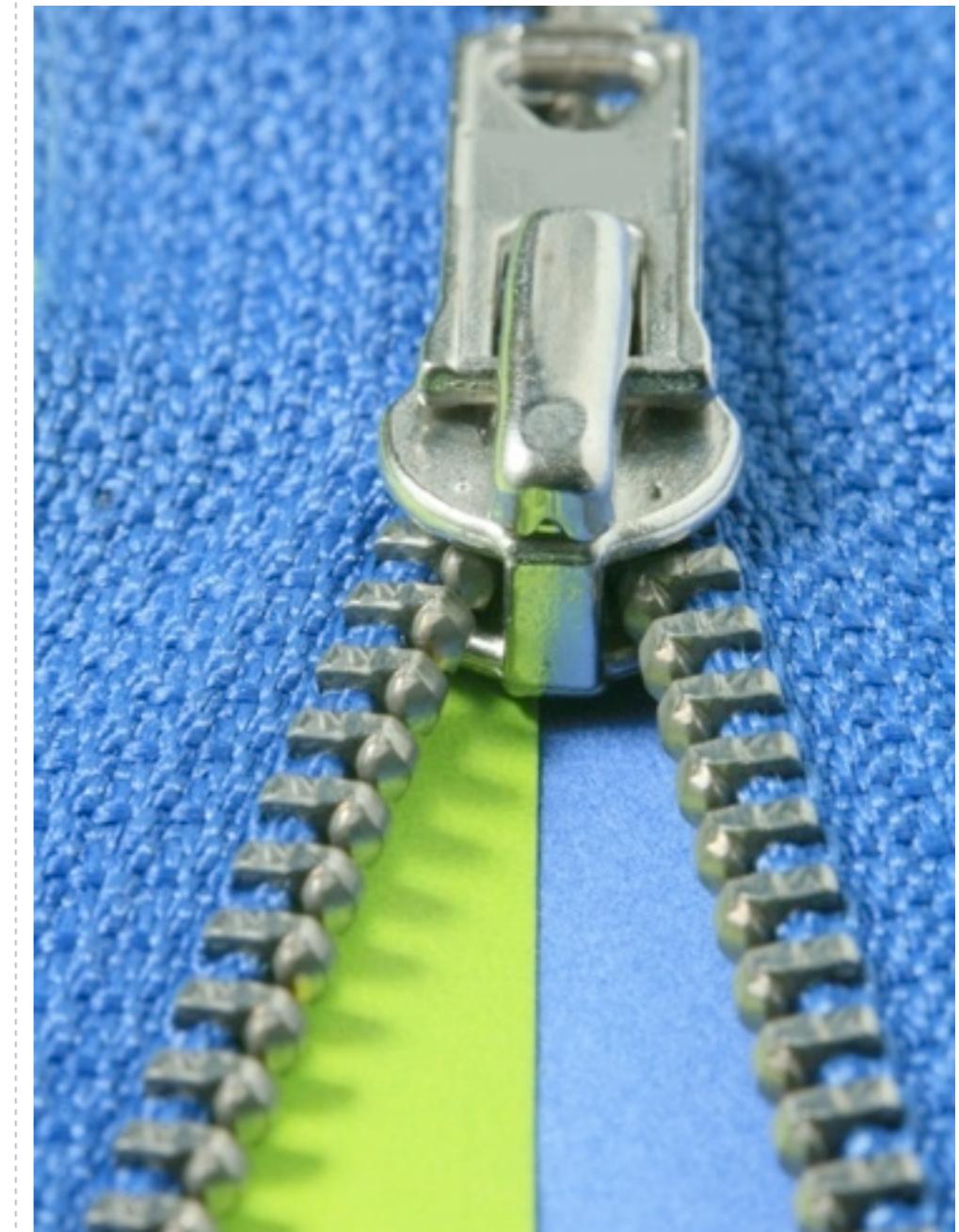
Branch

- ⬢ Ein Branch (= Ast) ist meist eine Ab-/Verzweigung einer Hauptentwicklungsline (meist „trunk“ genannt).
- ⬢ In SVN sind Branches Kopien von einer Ursprungsversion (In SVN über sog. „Cheap Copies“ realisiert).
- ⬢ Häufig werden Branches dazu verwendet um verschiedene Versionsstände von einander zu trennen oder um Arbeitsabläufe zu parallelisieren.
- ⬢ Bsp: trunk -> latest-test oder latest-test -> latest-production



Merge

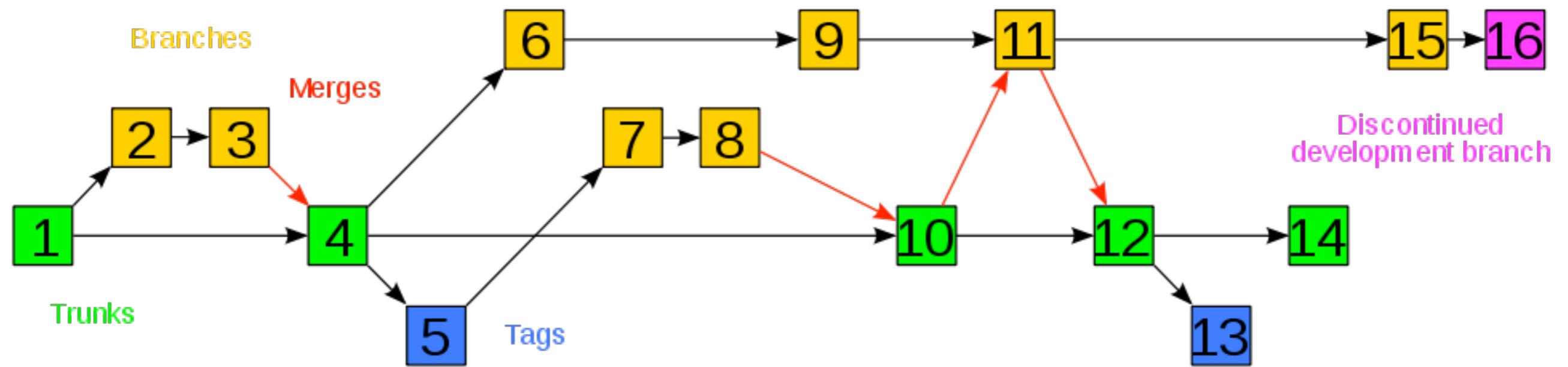
- ➊ Das Mergen oder Verschmelzen ist die Zusammenführung von verschiedenen Änderungen in zwei Versionen einer Datei oder auch eines Dateibaumes.
- ➋ Bekanntes Beispiel ist das Mergen von zwei Branches



Funktionsweise (Beispiel)

- ➊ touch foo.bar
- ➋ svn add foo.bar
- ➌ svn commit foo.bar -m „Initial checkin“
- ➍ echo „baz“ > foo.bar
- ➎ svn commit foo.bar -m „Added baz“





Eine SVN Timeline



Vorteile

- ⬢ Kostenfrei erhältlich
- ⬢ Erprobт im Alltag
- ⬢ Modern durch stetige Entwicklung als Apache Project
- ⬢ Akzeptiert von den meisten Entwicklern
- ⬢ Unterstützt von vielen IDEs, Clients und Shared Hosting Anbietern (z. B. SourceForge)
- ⬢ Einfach in der Handhabung
- ⬢ Komplexe Szenarien sind abbildbar.



Einschränkungen

- ⬢ Viele Operationen sind aus Datenhaltungssicht „teuer“
- ⬢ Ohne Server geht nichts
- ⬢ Es werden nur Deltas verwaltet
- ⬢ Automatisches Mergen ist in vielen Fällen keine schöne Erfahrung (= viele Konflikte bei offensichtlich eindeutigen Situationen)



Alleinstellungsmerkmale

- Properties auf Datei / Verzeichnisebene
- svn:externals um entfernte Repositories „hineinzulinken“





#3 Git





Wer hat's erfunden?



Linus Torvalds

- Initiator der Linux - Bewegung
- Wahrscheinlich der berühmteste Entwickler der heutigen Zeit
- Entwickelt aktiv am Linux Kernel
- Ist Erfinder von Git, jedoch nicht (mehr) der Hauptentwickler



Geschichte

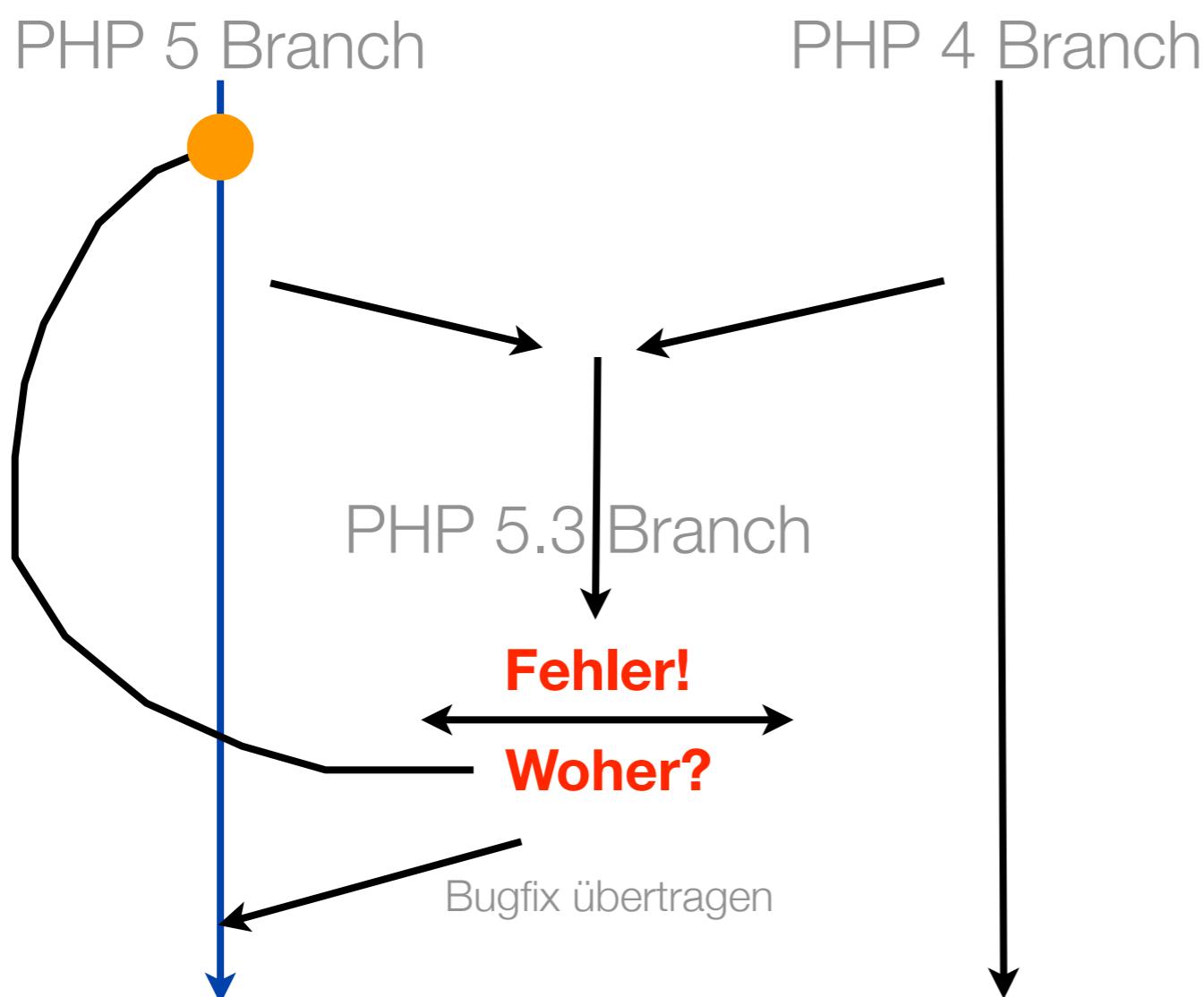
- ⬢ Gestartet im April 2005 um den damals verwendeten BitKeeper zu ersetzen von Linus Torvalds
- ⬢ Aktuell in der Version 1.7 verfügbar
- ⬢ Sehr junges Projekt
- ⬢ Schnell adaptiert worden (GitHub, Gitorious)



Besonderheit: Merge Gedächtnis

Nach dem Merge weiss Git wer „seine Eltern sind“





*Einige Zeit nach
Erstellung des PHP
5.3 Branches wird ein
Fehler festgestellt.*

*Dieser Fehler wird im
5.3er Branch gefixt,
wo muss er denn
noch gefixt werden?*

*Für Git kein Problem,
da es den Ursprung
der Datei und deren
Einzelteile kennt.*

*Die Historie verrät wo
der Fehler herkommt.*

Begriffe

Rebase

Staging

Dirty

Pull

Remote

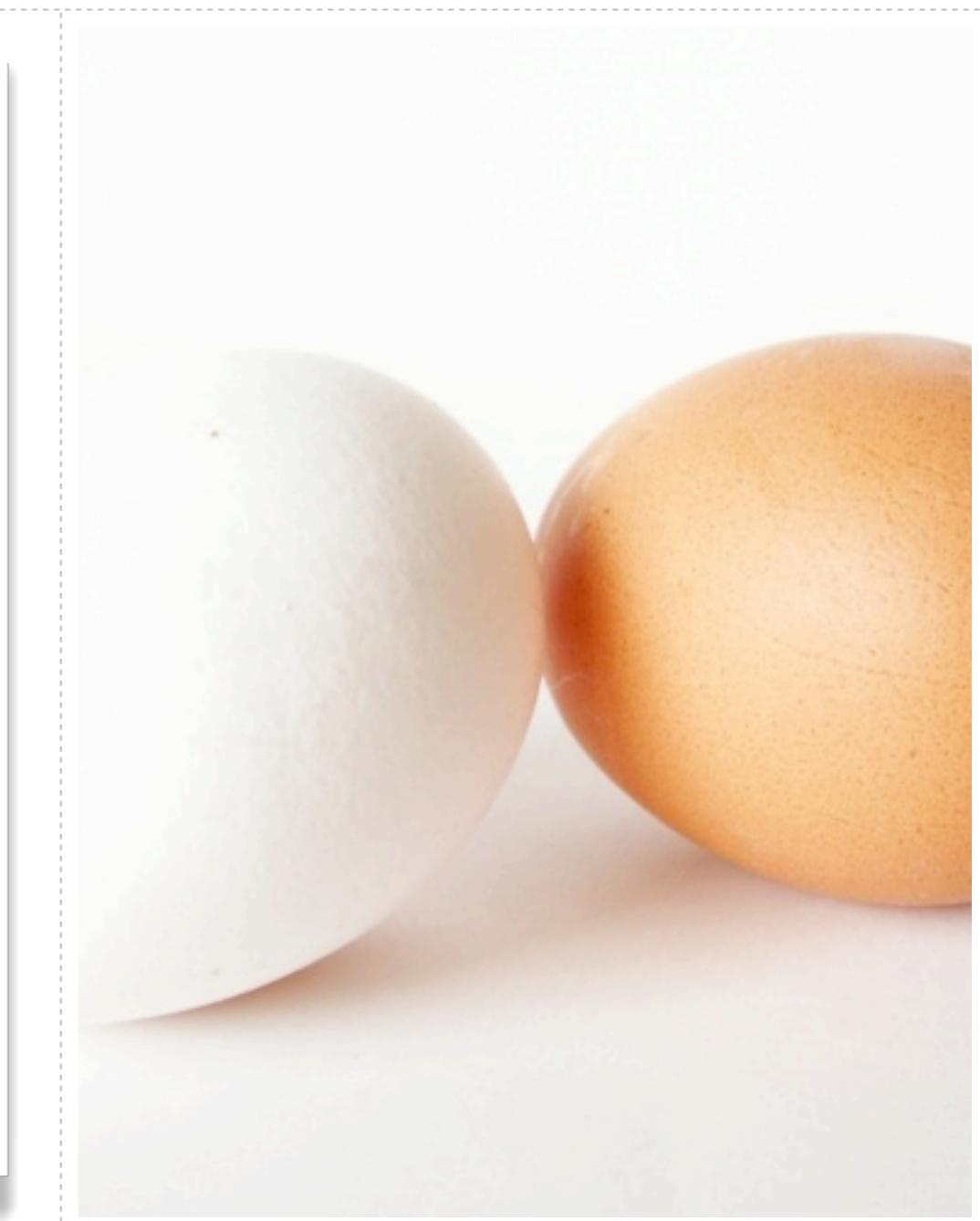
Clone

Push



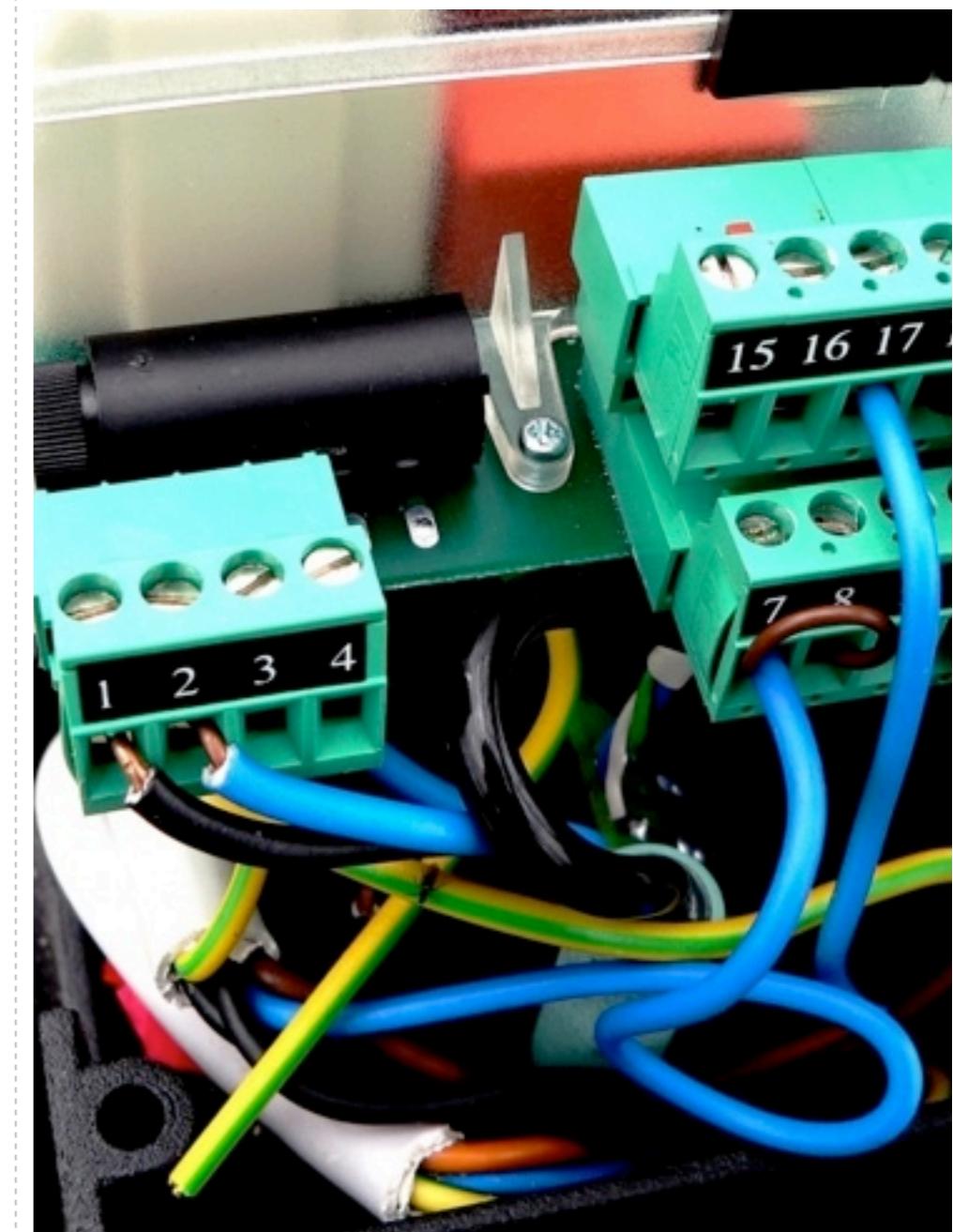
Clone

- ⬢ Unter einem „Clone“ versteht man das Spiegeln einer vollständigen Historie in ein (lokales) Repository.
- ⬢ Dabei wird jeder Commit, jeder Tag und jeder Branch mit einbezogen.



Remotes

- ⬢ Branches werden in Git in zwei Zuständen verwaltet. Lokal und Remote. Ein Remote Branch ist eine Referenz auf einen lokalen Branch in einem entfernten Repository.
- ⬢ Remotes werden interessant, wenn mehrere Entwickler am selben Branch arbeiten und den entwickelten Quellcode verteilen wollen.



Staging

- ❖ Hinzufügen von Dateien in einen virtuellen Bereich
- ❖ Alle Daten im Stage kommen in den nächsten Commit
- ❖ Commits sind dadurch auf CLI Ebene „zusammenbaubar“





Push

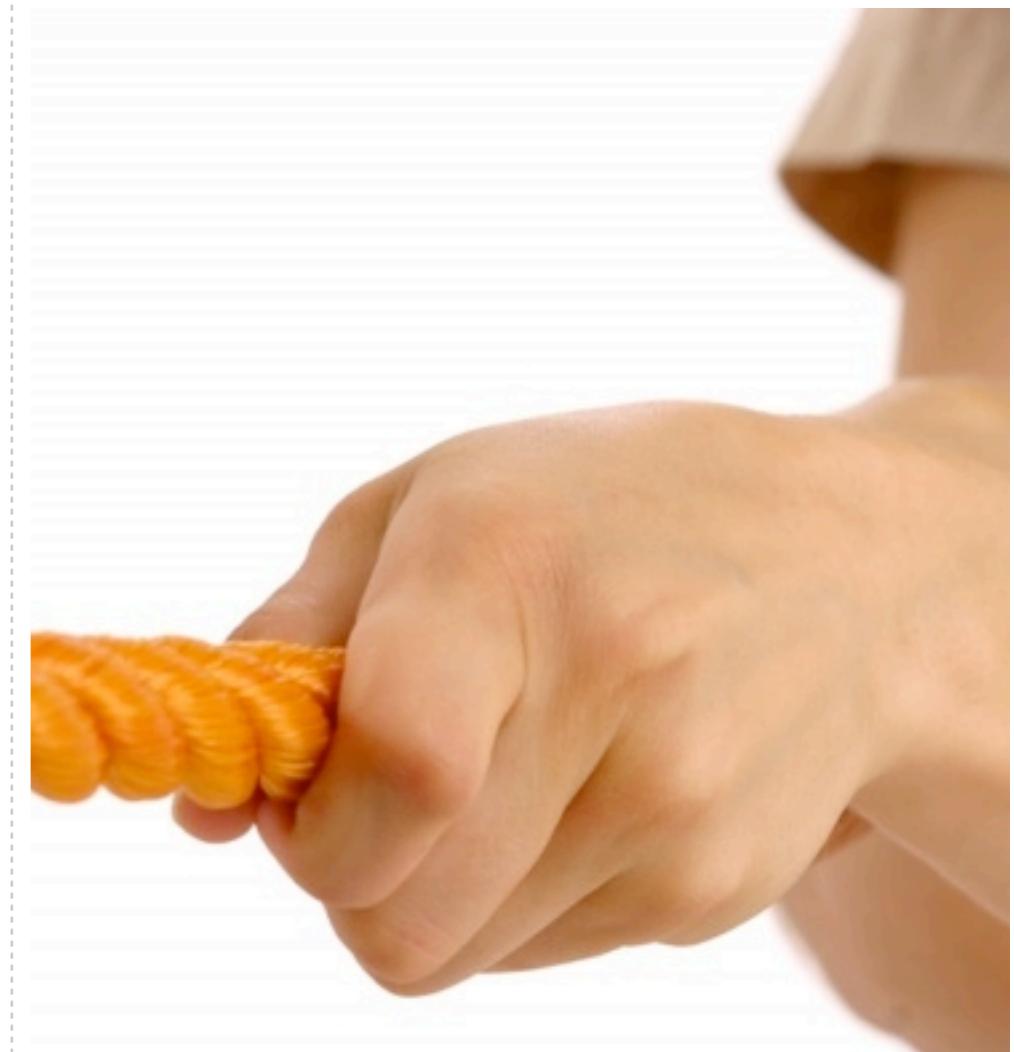
- ➊ Übermittelt den Inhalt eines Branches aus einem lokalen Repository an ein Remote Repository
- ➋ Transferiert Commit-By-Commit
- ➌ Aus Sicht des Remote Repositories sieht es aus, als hätte die Person lokal Committed





Pull

- ☰ „Zieht“ Änderungen aus einem Remote Repository
- ☰ Wenn mehrere Branches aus einem Remote existieren (z. B. nach einem Clone eines Repositories mit mehreren Branches), werden diese ebenfalls „gezogen“



Rebase

- ➊ Ähnelt dem Pull
- ➋ Zieht alle entfernten Änderungen in das lokale Repository
- ➌ läuft bis zu dem Punkt, in der Timeline, ab dem man selbst Veränderungen vorgenommen hat
- ➍ Wiederholt ab diesem Zeitpunkt alle Commits bis zum aktuellen Stand.
- ➎ Klarer Unterschied zum Merge: Rebase erzeugt keine neue Revision!



Dirty

- ☰ Als „dirty“ bezeichnet man Branches, die
 - ☰ Änderungen gegenüber dem letzten Commit besitzen
 - ☰ ihre Änderungen noch nicht (vollständig) im Staging haben
- ☰ Beispielkonstellation für den Zustand „dirty“
 - ☰ Foo.txt (tracked, unchanged)
 - ☰ Bar.php (tracked, changed, unstaged)
 - ☰ Baz.css (tracked, changed, staged)

Grund für den Zustand „dirty“



Funktionsweise

- ➊ Git Repositories bestehen aus drei Komponenten:
 - ➌ Tree Objekte
 - ➌ Commit Objekte
 - ➌ Blobs (*Binary Large OBjects*)
- ➋ Jedes Objekt bekommt eine repository-weit eindeutige ID in Form einer SHA-1 Prüfsumme

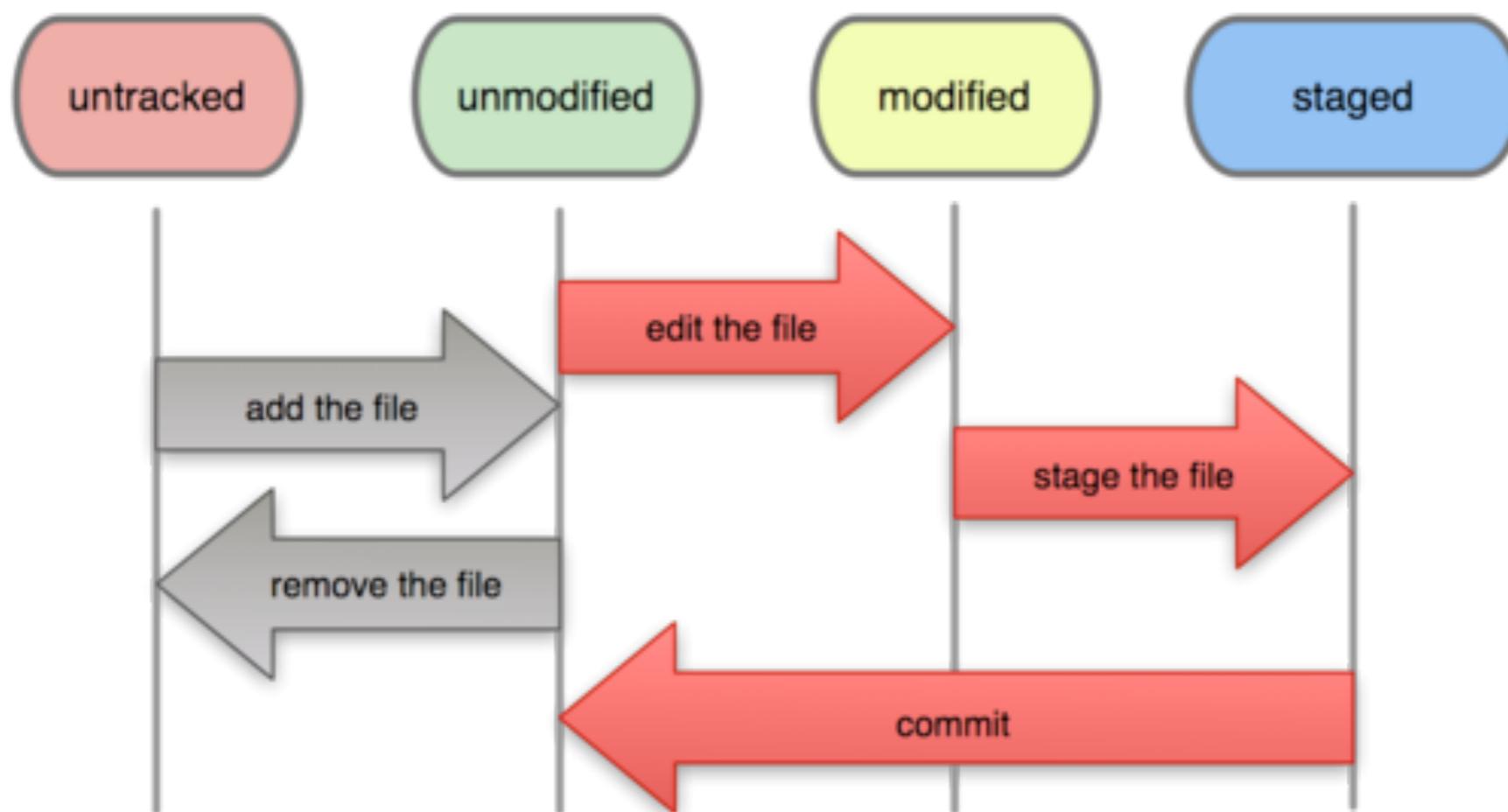


Funktionsweise

- ⬢ Alles wird in einem .git Ordner im Wurzelverzeichnis des Repositories gesammelt. (Keine verstreuten .svn Ordner mehr)
- ⬢ Dateien werden nach ihrem Inhalt beurteilt, nicht nach ihrem Namen
 - ⬢ „Renaming-Detection“ ist also eingebaut
 - ⬢ Kein Linux/Windows „Conflict State“ Problem bei Groß- und Kleinschreibung



File Status Lifecycle

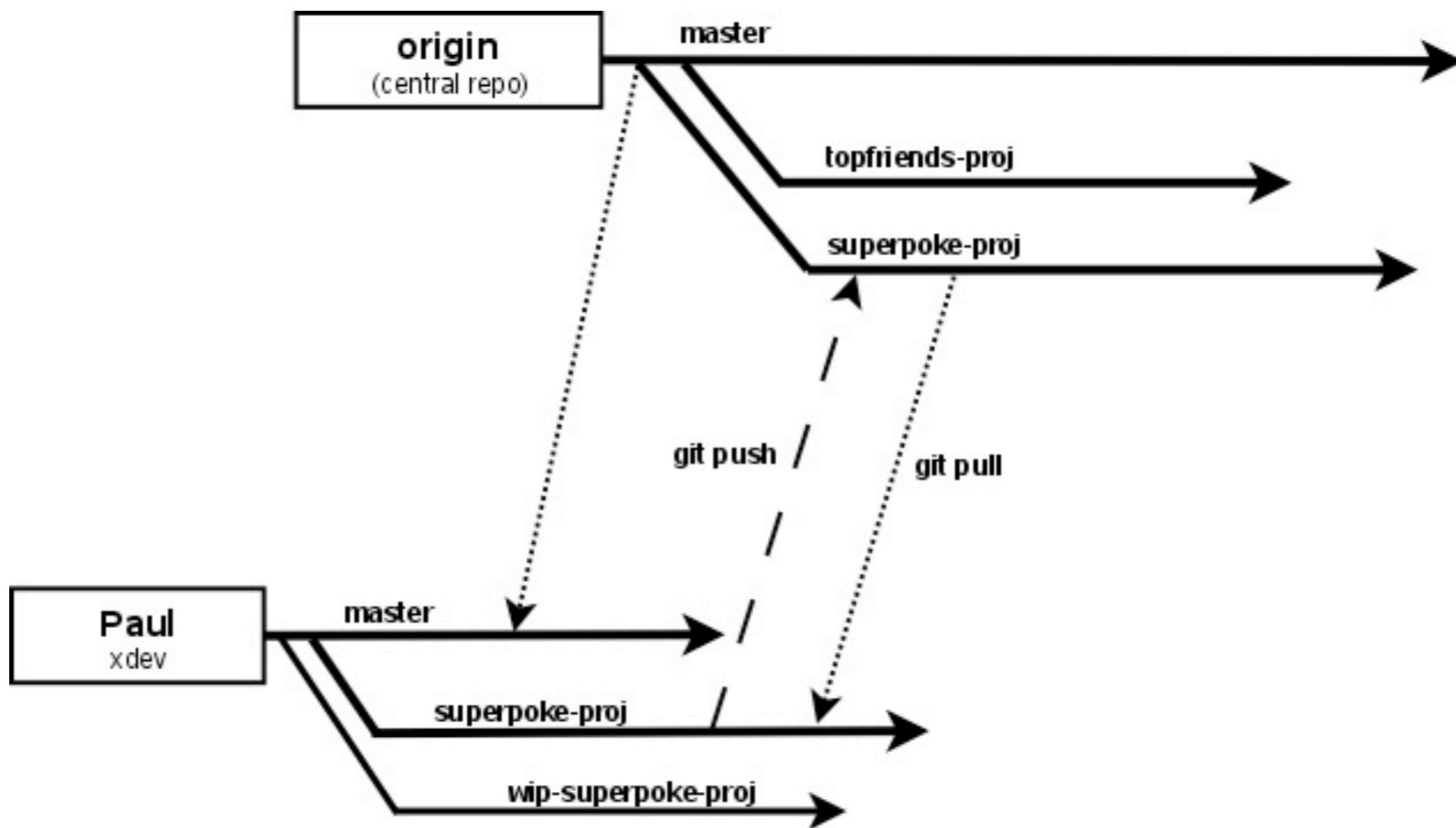


Arbeitsschritte

4-faltigkeit

- ⬢ Eine Blob kann aus Sicht von Git vier Zustände annehmen
 - ⬢ untracked (nicht versioniert)
 - ⬢ unmodified (versioniert, aber nicht verändert)
 - ⬢ modified (versioniert, verändert, nicht im Stage)
 - ⬢ staged (versioniert, verändert und im Stage, aber nicht committed)

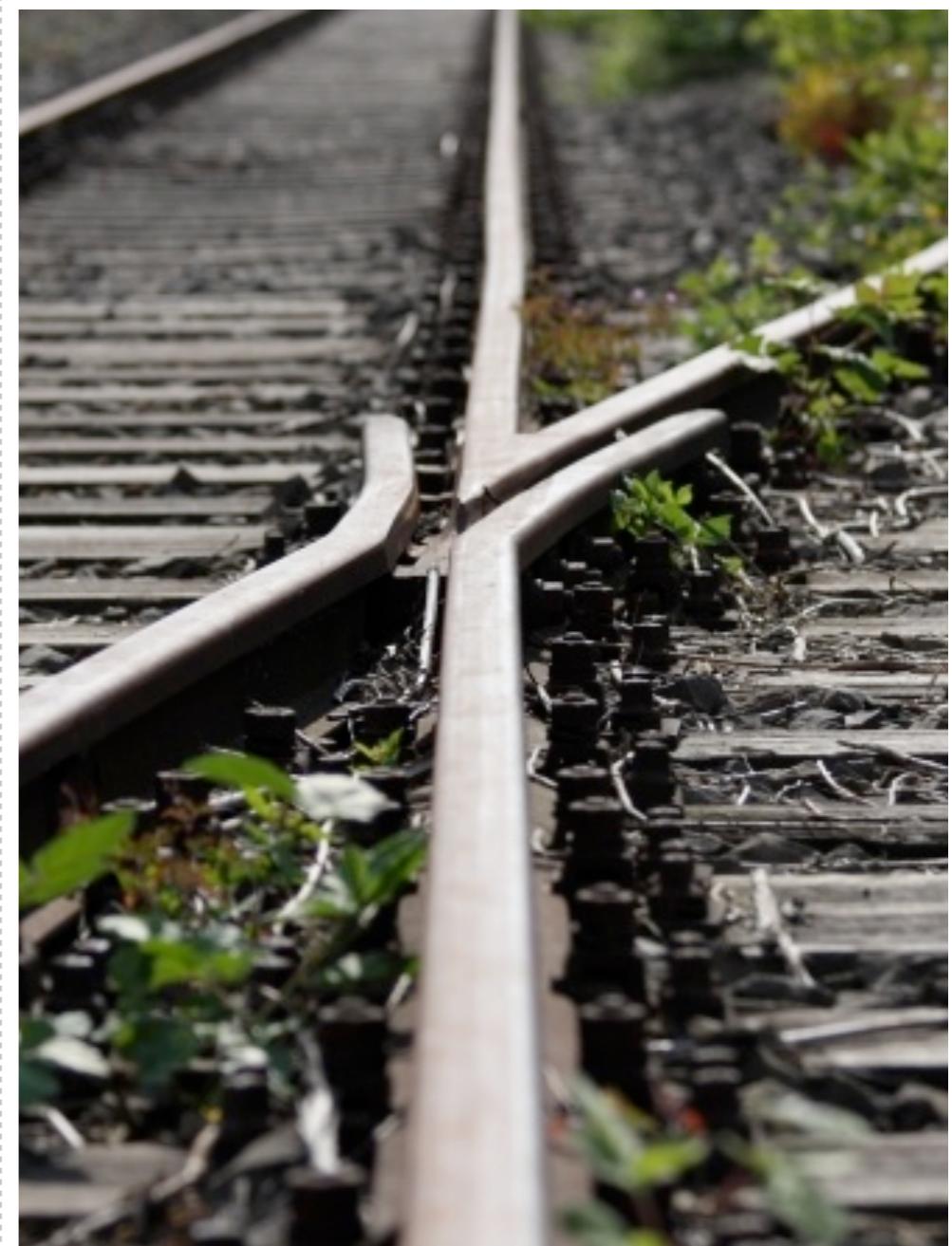




Multi-Branch Modell

Branching

- ❖ Gehört zum täglichen Arbeiten
- ❖ Ein Branch pro Feature / Bugfix / Change
- ❖ Lokales und entferntes Branchen möglich
- ❖ Saubere Fallunterscheidung
- ❖ Sichere Code-Basis, da definierter Stand und Kenntnis über den Ursprung des Stands (jeder Branch kennt den Punkt ab dem er divergiert (abgewichen) ist



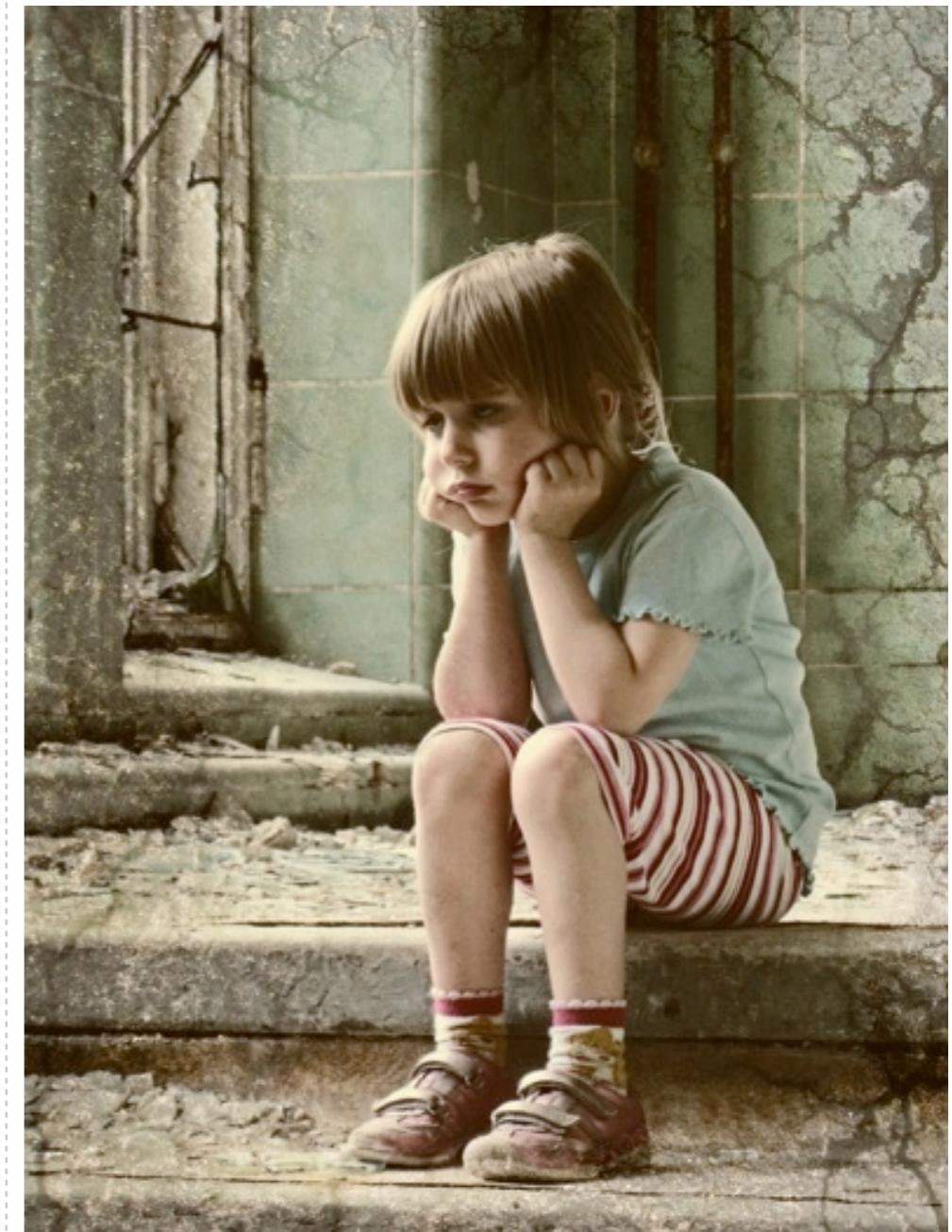
Vorteile

- ⬢ Schnell, da eine Vielzahl der Operationen lokal ist
- ⬢ Unabhängig, da kein Server benötigt wird
- ⬢ Sicher, da jeder alles besitzt (= verteiltes Backup)
- ⬢ Modern, da Objekt-orientierte Sichtweise auf die Teilstücke des Versionsbaumes
- ⬢ Vollkommene Freiheit, da jeder sich selbst organisieren kann.



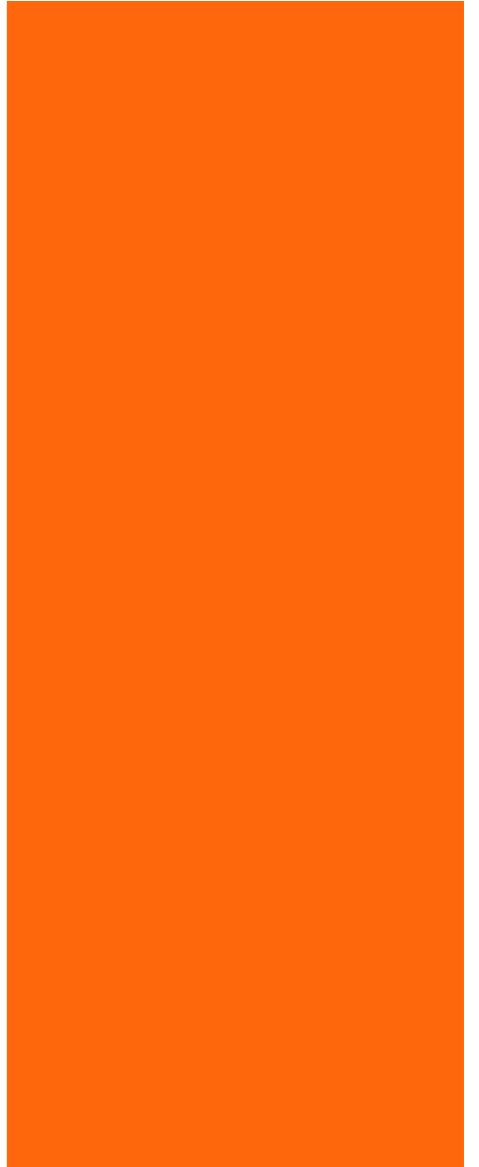
Einschränkungen

- ⬢ Kaum Möglichkeiten Teilstücke des Codes per ACL einzuschränken
- ⬢ Komplizierter Einsatz unter Windows
- ⬢ Kaum GUIs oder IDE Plugins
- ⬢ „Ungemütliche Lernkurve“
- ⬢ Vollkommene Freiheit





**ZEIT-
SPAREN**

A photograph of a book titled "ZEIT-SPAREN". The title is printed in large, bold, black capital letters. Below the title is a horizontal bar with a gradient from red on the left to yellow on the right. The book is bound in a light-colored cover with a visible vertical crease where the pages meet at the spine.

**ZEITVOR-
WAHL**

A photograph of a book titled "ZEITVOR-WAHL". The title is printed in large, bold, black capital letters. Below the title is a dark, solid horizontal bar. The book is bound in a light-colored cover with a visible vertical crease where the pages meet at the spine.

Workflow Modelle



Team Organisation

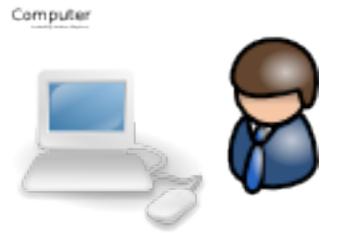
- ❖ Es sind verschiedene Ansätze zur Organisation von Teams entstanden
- ❖ Viele sind auch in der zentralisierten Welt vorhanden, aber wenig genutzt
- ❖ Canonical hat mit der Veröffentlichung von Bazaar in Verbindung mit Launchpad sehr gute Arbeit geleistet und diese möglichen Workflows dokumentiert (<http://wiki.bazaar.canonical.com/Workflows>)
- ❖ Hier stelle ich 3 Modelle beispielhaft vor



Workflow - Ein User

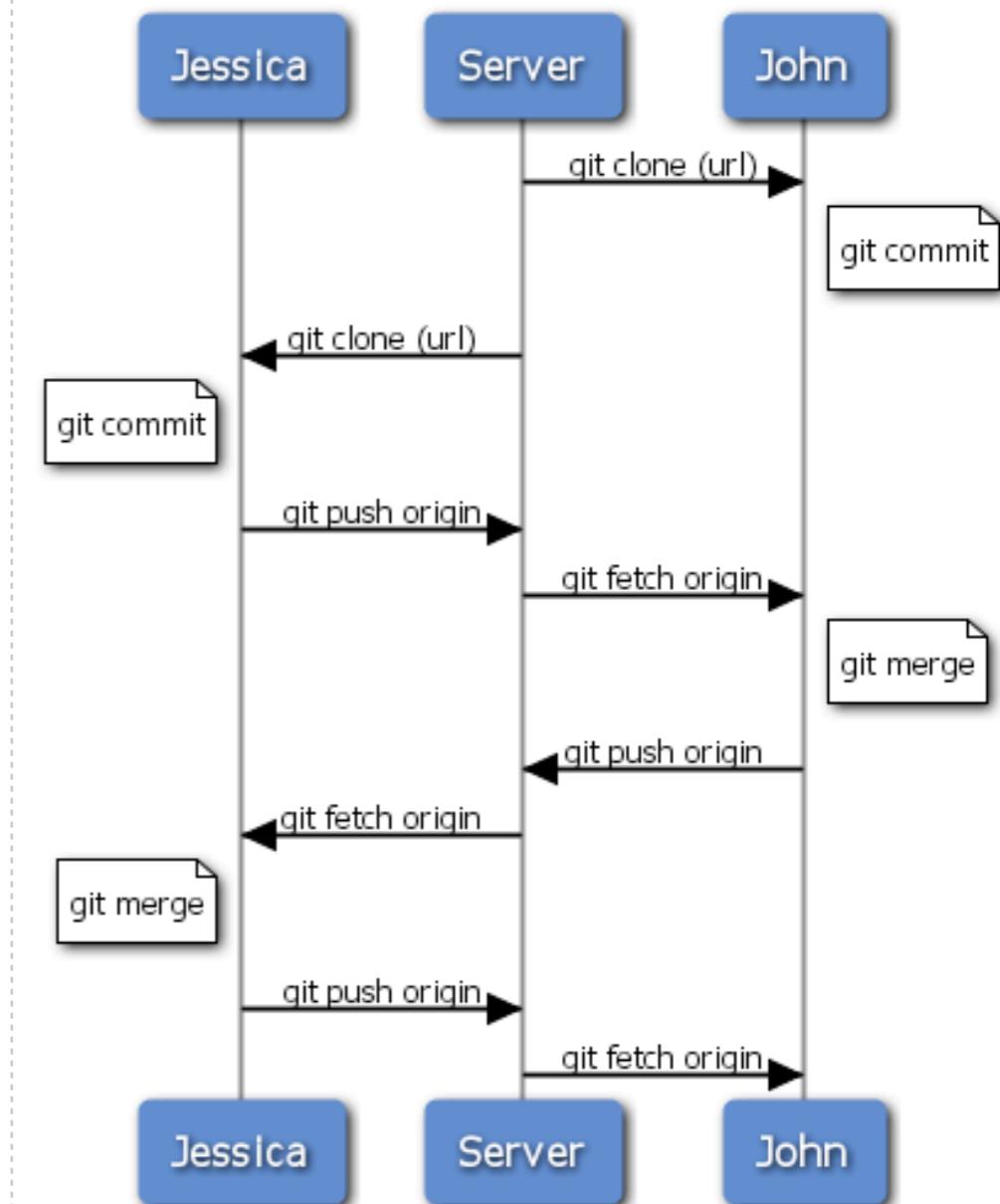
- ⬢ Der „Freelancer - Workflow“
- ⬢ Gut für einzelne Programmierer, die
 - ⬢ Weder Zeit
 - ⬢ noch Ressourcen für das Setup eines SVN Servers haben
- ⬢ Schlecht, wenn man kein Backup hat und die Festplatte / das Speichermedium verliert

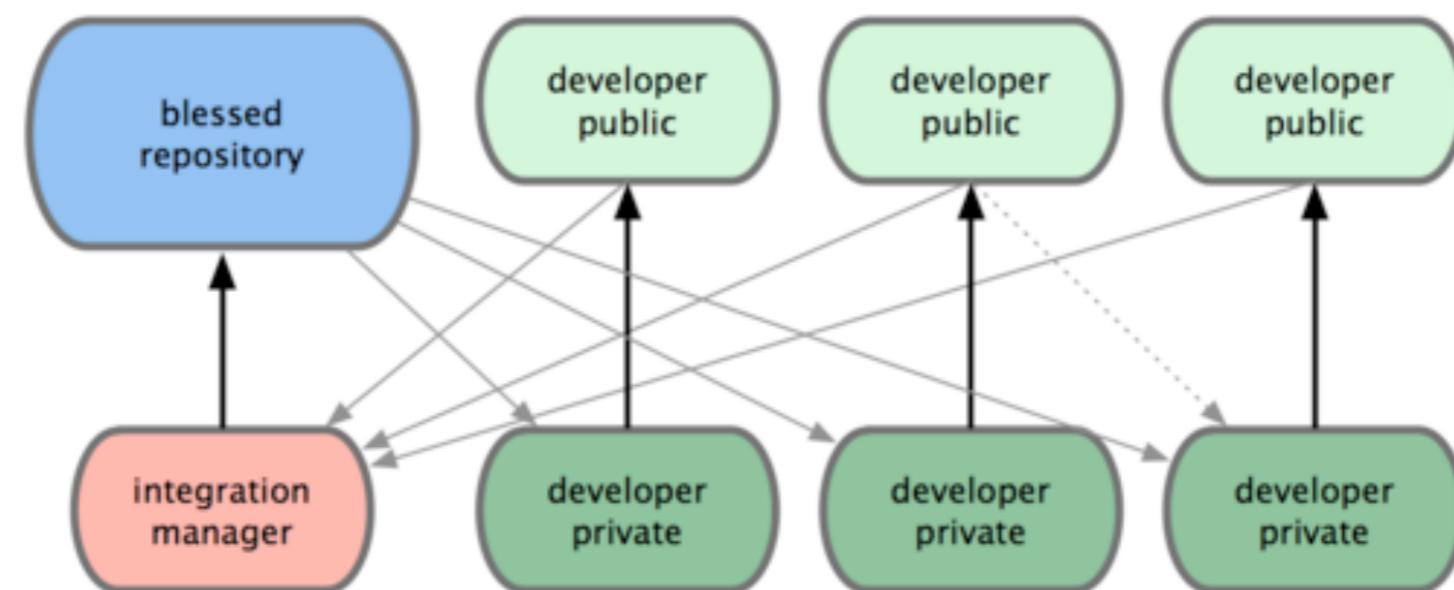
- ① create project
- ② record changes
- ③ browse history
- ④ package release



Workflow - kleines Team

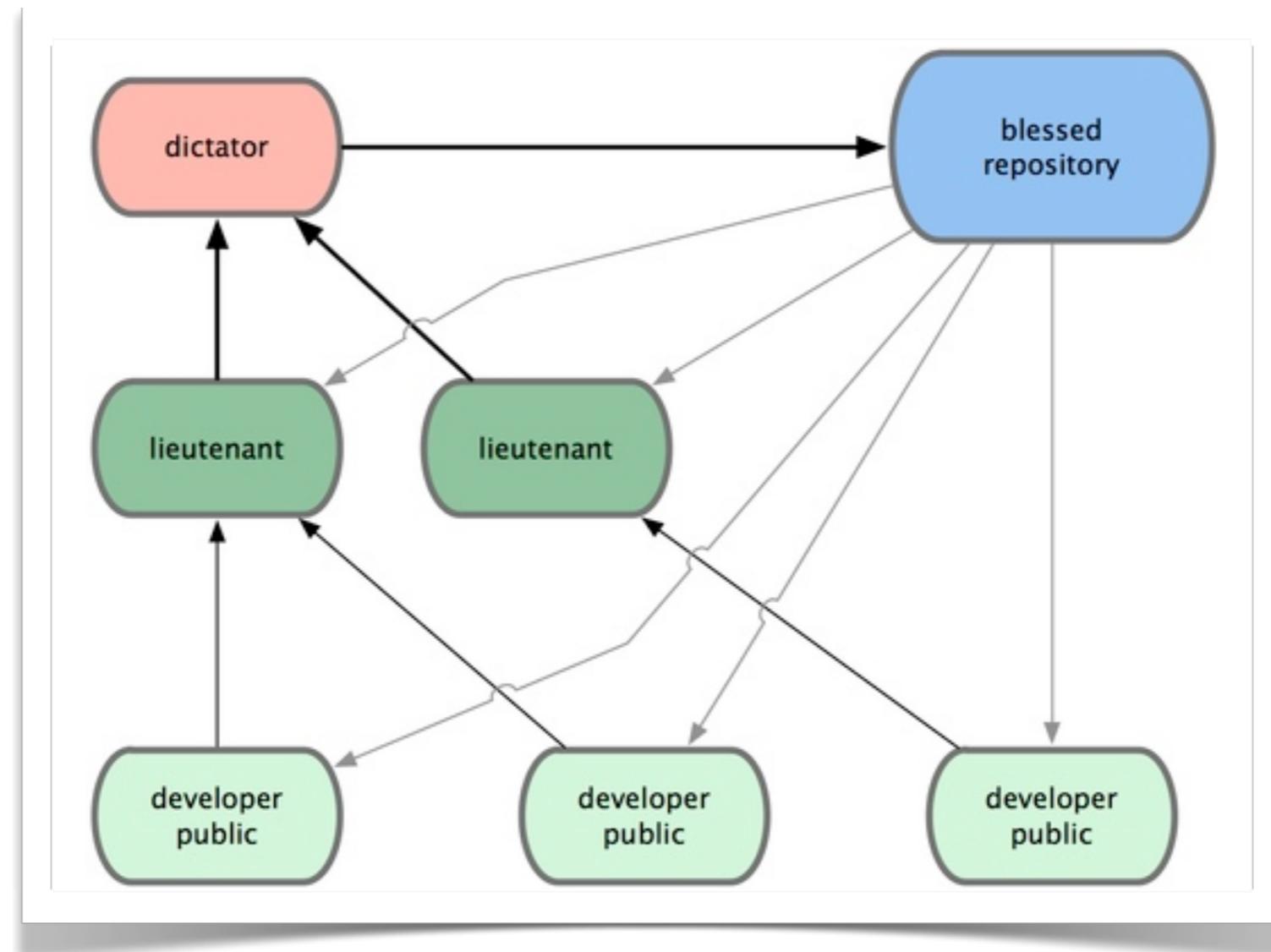
- Es gibt ein „blessed“ Repository, also ein zentrales Repository
- Jeder klont sich dieses Repository ein mal
- Ab dann werden Änderungen per push & pull verteilt
- Sinnvoll für kleine Teams (zwischen 2 und 6 Leuten) mit überschaubaren Commit-Zahlen





Workflow - Integration Manager



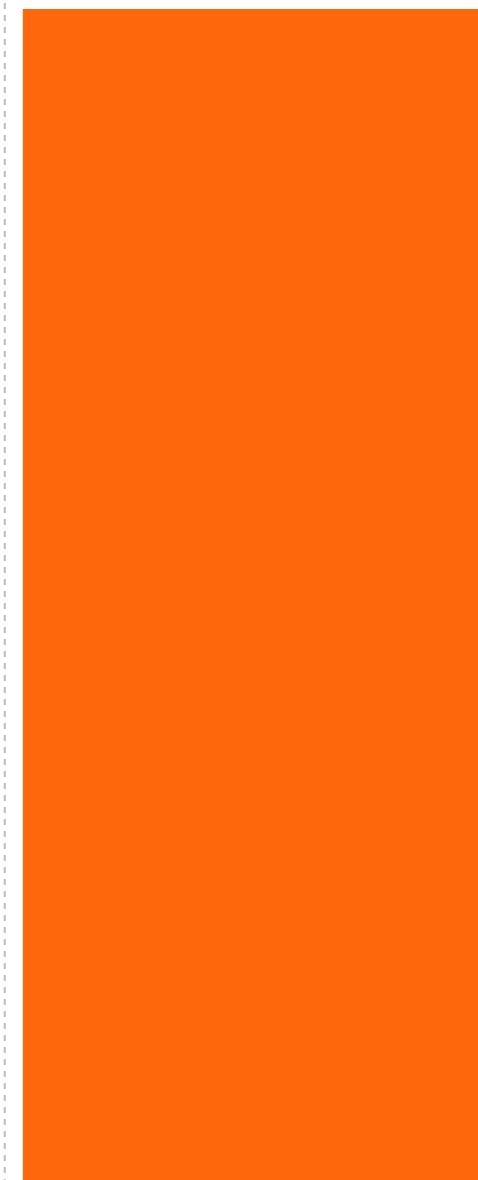


Workflow - (benevolent) Dictator

Workflow Integ. Manager / Dictator

- ❖ Für große Teams geeignet
- ❖ Hoher Management-Aufwand
- ❖ Hohe Parallelisierung
- ❖ Sehr guter Zustand des Repository
- ❖ Der Integration Manager lohnt sich ab 10-15 Personen
- ❖ Das Dictator Modell erst bei 50+ Personen





Pro & Kontra

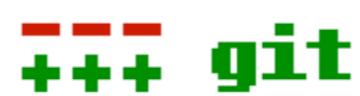




- Etabliert
- Techn. Ausgereift
- Verstanden
- Unterstützt
- Verfügbar
- Rechte-
management



- Langsam
- Historie „dumm“
- Branching ist anstrengend
- Binärdaten-
behandlung
- Ohne Server unbrauchbar

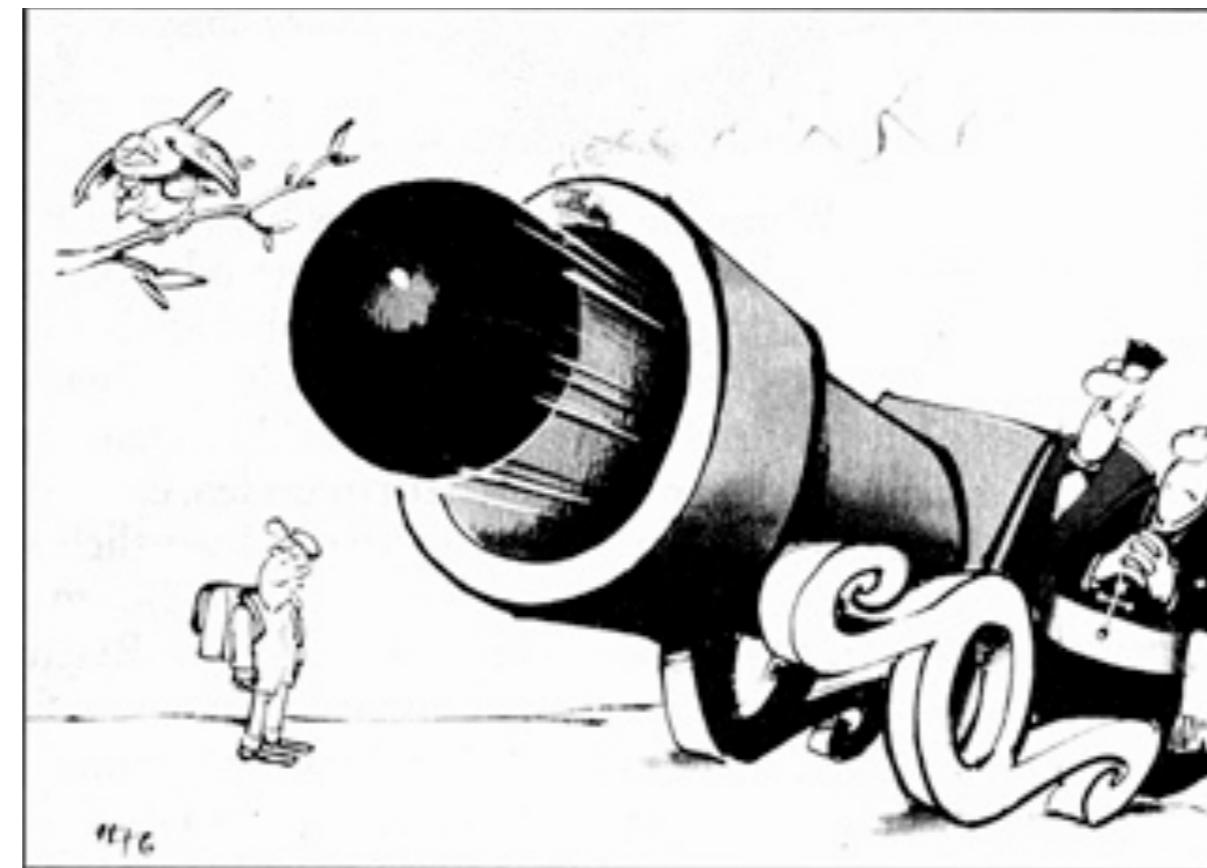


- Schnell
- Mergen ist sicher & einfach
- Branching ist erwünscht
- Ohne Server verwendbar
- Sehr intellig.
Historie



- Steiniger Einstieg
- Verlangt einen Paradigmen-
wechsel
- Nicht auf allen OS „gut“
- Fehlendes Rechte-
management





Die richtigen Mittel für den
richtigen Zweck

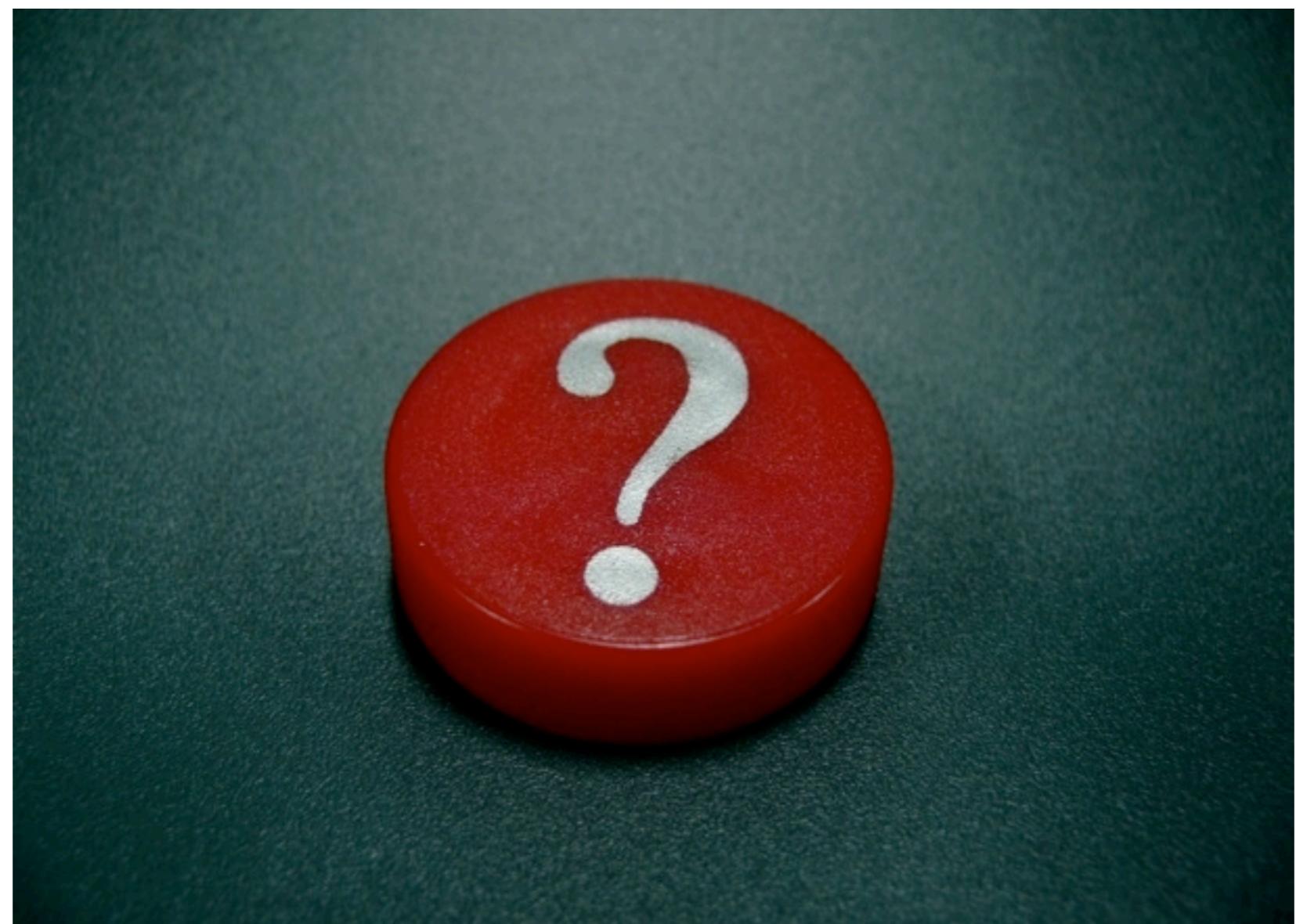
Der Umstieg / Lernaufwand lohnt sich ...

- ☰ wenn man selbst OpenSource Software schreibt und diese z. B. auf Github bereitstellen möchte
- ☰ wenn man viel im OpenSource Software Umfeld unterwegs ist, denn Git wird dort immer stärker
- ☰ wenn man große Mengen an Sourcen verwalten muss und Geschwindigkeit eine Rolle spielt
- ☰ wenn SVN mal wieder den Merge verrissen hat und man sehen will wie es auch anders geht
- ☰ wenn man die Definition von „Versionshistorie“ bei SVN auch so mager findet
- ☰ wenn man keine Versionskontrolle nutzt, die Sourcen lokal halten will und SVN Server doofe Ohren haben





Vielen Dank für die
Aufmerksamkeit!



Fragen



Quellen - lexikalisch

- ProGit - <http://progit.org/book>
- Wikipedia - <http://de.wikipedia.org/wiki/Git>
- Wikipedia - <http://de.wikipedia.org/wiki/SVN>
- Version Control with SVN - <http://svnbook.red-bean.com/>
- Bazaar Workflows - <http://wiki.bazaar.canonical.com/Workflows>
- Git-SCM - <http://git-scm.com/>
- Subversion - <http://subversion.apache.org/>



Quellen - Bilder

-  ProGit (Workflows)
-  Bazaar (Workflows)
-  Wikipedia (Logos)
-  Diverse Aboutpixel.de Fotografen (siehe Box rechts)
-  Uni-grazt.at (Kanonen auf Spatzen)

aboutpixel.de Fragezeichen © Werner Linnemann
aboutpixel.de Dialog © schwald-werbegestaltung
aboutpixel.de up to date © Bernd Boscolo
aboutpixel.de RISIKO! © Braun Alexander
aboutpixel.de öffnen © Rainer Sturm
aboutpixel.de Gemarkt © Simon Ledermann
aboutpixel.de Zwilling © Rosita Sellmann
aboutpixel.de Eierpappe © daylight
aboutpixel.de StrangZiehen © Sven Schneider
aboutpixel.de starkes mädel © regine schöttl
aboutpixel.de Partystimmung © Sebastian B.
aboutpixel.de Verkabelung © Werner Linnemann
aboutpixel.de Eierlei... 3 © Steve_ohne_S
aboutpixel.de starkes Blatt © Rainer Sturm
aboutpixel.de Und die Akten turmen sich... © S. Lingk
aboutpixel.de Wie - Mist? © daylight
aboutpixel.de doof hier! © Joana Virck-Alevra
aboutpixel.de EXIT © Sven Schneider
aboutpixel.de Schalenfrüchte © Nadine Dauwalter
aboutpixel.de Zeit sparen © Arnim Schindler
aboutpixel.de Tacitus 1 © Heinz Hasselberg
aboutpixel.de Can u see the difference © Freive
aboutpixel.de Pinguin © tina fritze
aboutpixel.de Netz © Walter Christ
aboutpixel.de Weiche © Tiberius K.

