Versionskontrolle mit Subversion

Eine Einführung

Jörg Faschingbauer

Table of Contents

FASCHINGBAUER

1. Einleitung
2. Basic Workflow
3. Branches
4. Merging
5. Verschiedenes
6. Administration
7. Schlusswort

Overview

FASCHINGBAUER

1. Einleitung
2. Basic Workflow
3. Branches
4. Merging
5. Verschiedenes
6. Administration
7. Schlusswort

Workflow: Draufhauen

FASCHINGBAUER

Keine Versionskontrolle: die einfachste Lösung

- Es gibt genau eine Kopie des Source
- An ihr wird gearbeitet
- Zu jeder Zeit nur *ein* Stand verfügbar

Nachteile (... sofern sie nicht auf der Hand liegen)

- Man kann nicht zurück, wenn man sich in einer Sackgasse befindet
- Parallelentwicklung (mehrer Leute am gleichen Code) *undenkbar* und unmöglich bzw. verboten
- (Vorteil: Konflikte von vorneherein ausgeschlossen)

Workflow: Sicherungskopien

FASCHINGBAUER

Versionskontrolle per Sicherungskopie

- Bevor man was einbaut, macht man eine Sicherungskopie
- Dann haut man drauf, und irgendwann geht es und ist “fertig”

Nachteile

- Grobgranular: die “Arbeitskopie” (aktueller Stand) bleibt lange in der Schwebe
- Keine “Zwischenstände”
- Kollegen wissen nichts vom Draufhauen → Parallelentwicklung ist Unfall

```
.
|-- 2014-01-04
|   |-- a.txt
|   `-- b.txt
|-- 2014-01-07
|   |-- a.txt
|   `-- b.txt
`-- current
    |-- a.txt
    `-- b.txt
```

diff: Vergleichen von Textfiles

FASCHINGBAUER

Unterschied (Informatisch: “Diff”) zwischen zwei Files

Frage: was hat sich von 2014-01-04 auf 2014-01-07 geändert?

```
$ diff -u 2014-01-04/a.txt 2014-01-07/a.txt
$ diff -u 2014-01-04/b.txt 2014-01-07/b.txt
--- 2014-01-04/b.txt 2014-02-13 15:06:41.756888053 +0100
+++ 2014-01-07/b.txt 2014-02-13 15:08:18.405885831 +0100
@@ -1 +1 @@
-Das ist ein dummer Text namens B
+Das ist ein gscheiter Text namens B
```

- formale - mathematische - Änderungsvorschrift, um von *Version* 2014-01-04 auf *Version* 2014-01-07 zu kommen
- Basis von Versionskontrollsystemen

diff: Vergleichen von Directories

FASCHINGBAUER

Rekursives Vergleichen

```
$ diff -ur 2014-01-04/ 2014-01-07/
diff -ur 2014-01-04/b.txt 2014-01-07/b.txt
--- 2014-01-04/b.txt 2014-02-13 15:06:41.756888053 +0100
+++ 2014-01-07/b.txt 2014-02-13 15:08:18.405885831 +0100
@@ -1 +1 @@
-Das ist ein dummer Text namens B
+Das ist ein gscheiter Text namens B
```

- Änderungen in allen Files in beliebiger Tiefe
- Schwachstelle: Umbenennen und Löschen von Files → Files werden "leergemacht" (verschwinden nicht)

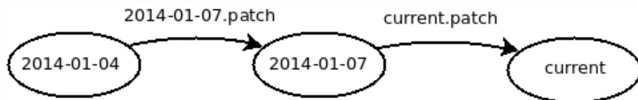
`diff`: Patches

FASCHINGBAUER

Patch: File, das einen Diff enthält

```
$ diff -ur 2014-01-04/ 2014-01-07/ > 2014-01-07.patch
$ diff -ur 2014-01-04/ current/ > current.patch
```

- Versionskette $\Longleftrightarrow$ Folge von *Patches* auf die Originalversion
- $\rightarrow$ Versionsdatenbank mit Relationen liegt (einigermassen) nahe



patch: Anwenden von Patches

FASCHINGBAUER

Frage:

- Ich habe Version 2014-01-04/ und einen Patch 2014-01-07.patch
- Wie komme ich auf Version 2014-01-07/?

```
$ cp -r 2014-01-04/ 2014-01-07/
$ (cd 2014-01-07 && patch -p 1) < 2014-01-07.patch
patching file b.txt
```

- Alles klar?
- Auch egal, dafür gibt es Versionskontrollsysteme

Resümee: Versionskontrolle

FASCHINGBAUER

Lineare Versionskontrolle

- Abfolge von Kopien
- Von mir aus formalisiert mit Patches
- Änderungen von Version zu Version sind definiert und klar
- Programme:
  - `SCCS`: Source Code Control System
  - `RCS`: Revision Control System (streng genommen *nichtlinear*, aber selten so verwendet, weil kompliziert → CVS)

Aber:

- Was passiert, wenn mehrere Entwickler gleichzeitig eine Kopie anfertigen?
- → Entwicklungszweig ("Branch")

Ausblick: Versionskontrolle mit Subversion

FASCHINGBAUER

Konzepte

- *Gleichzeitigkeit*
  - Branch — Paralleler Entwicklungszweig
  - Merge — Zusammenführen von Branches
- *Release-Management*
  - Tags — Markieren von ausgelieferten Versionen
  - Bugfix- und Releasebranches
  - Entwicklungsbranches
- *Repository*
  - Zentraler Speicher von Versionen
  - Auschecken, Einchecken
  - Zugriffsmethoden
  - Administration: Backup, Restore

Overview

1. Einleitung
2. Basic Workflow
3. Branches
4. Merging
5. Verschiedenes
6. Administration
7. Schlusswort

`svn checkout` — "Working Copy" (1)

FASCHINGBAUER

Am Anfang ist Nichts — ausser das Repository

- Subversion ist ein zentrales Versionskontrollsystem → genau ein Repository
- Beliebig viele Benutzer können daran arbeiten → *gleichzeitig entwickeln*
- Vorausgesetzt, sie haben eine "Working Copy"
- → *Checkout*

`svn checkout` — "Working Copy" (2)

FASCHINGBAUER

Zugriffsmethoden

- gibt es viele
- Für das Verstehen nicht so wichtig
- → später
- Einstweilen: `svn+ssh://user@svn.repository.com/path/to/repo/trunk`
- `trunk`: "Stamm", Hauptentwicklungszweig (zum Unterschied von Branches → später)

svn checkout — "Working Copy" (3)

FASCHINGBAUER

Working Copy

- By Default wie der Basename des ausgecheckten Pfades
- Die jüngste Version am Hauptentwicklungszweig

```
$ svn checkout svn+ssh://user@svn.repository.com/path/to/repo/
$ tree trunk/
trunk/
|-- a.txt
`-- b.txt
```

Endlich Arbeiten

FASCHINGBAUER

Working Copy — *Arbeits*kopie → los geht's

- Änderungen werden lokal angebracht → Working *Copy*
- ... gleich wie bei der Sicherungskopie-Analogie von vorhin

Was hab ich geändert und noch nicht committed?

```
$ svn diff
Index: b.txt
==================================================================
--- b.txt (revision 2)
+++ b.txt (working copy)
@@ -1 +1 @@
-Das ist ein dummer Text namens B
+Das ist ein gscheiter Text namens B
```

Working Copy Ändern

FASCHINGBAUER

Änderungen anbringen

- File/Directory hinzufügen: `svn add NAME`
- File/Directory löschen: `svn rm NAME`
- File/Directory umbenennen: `svn mv NAME`

svn commit — Eine neue Version erzeugen

FASCHINGBAUER

Bin fertig! Wenn man fertig ist, wird *eingecheckt* → *eine neue Version erzeugt*

```
$ svn commit -m 'dumm -> gscheit'
Sending        b.txt
Transmitting file data .
Committed revision 3.
```

- `svn commit` ohne `-m`: Editor (`$EDITOR`) poppt auf
- Vulgonamen: `svn ci`, `svn checkin`

Konfliktpotential: Gleichzeitigkeit

FASCHINGBAUER

Konflikte sind normal und alltäglich ...

Entwickler A	Rev	Entwickler B	Rev
checkout	42	checkout	42
modify		modify	
checkin	43		
		checkin	??

- Commit von Entwickler B (Revision 44) würde die Änderungen von Entwickler A *überschreiben*
- B muss sich drum kümmern → *Update* (und evtl. *Merge*) auf die neueste Version

Zusammenarbeit und Konflikte (1)

FASCHINGBAUER

Kollegen sind nicht böse:

- Sie machen ihre Arbeit
- Sie arbeiten unter Umständen an den gleichen Files/Modulen/...
- Man redet miteinander
- Man kommt sich nicht in die Quere
- Konflikte sind immer menschlicher Natur
- $\rightarrow$ kein Versionskontrollsystem kann da helfen

Zusammenarbeit und Konflikte (2)

FASCHINGBAUER

Arbeitsweise

- Subversion hat keine Ahnung von Programmiersprachen
- → zeilenbasiert (mit gewissen Heuristiken)
- → Konflikt ist, wenn mehrere Entwickler die gleiche Zeile ändern

Achtung

- Kein Versionskontrollsystem kann ein Ersatz für Kommunikation sein
- Es gibt auch Konflikte, die Subversion nicht erkennt
  - Widersprüchliche Änderungen in verschiedenen Modulen, die voneinander abhängen
  - Bei dynamischen Sprachen hilft auch kein Compiler

svn update — Änderungen holen

FASCHINGBAUER

Änderungen holen, die inzwischen passiert sind ...

```
$ svn update
Updating '.':
U    a.txt
Updated to revision 4.
```

- Keine lokale Modifikation in der Working Copy → geht butterweich rein
- Lokale Modifikation → komplizierter

Ernsthafte Konflikte (1)

FASCHINGBAUER

Auffassungsunterschiede zwischen Individuen. Entwickler A will folgendes committen:

```
$ svn diff
Index: b.txt
==============================================================
--- b.txt (revision 4)
+++ b.txt (working copy)
@@ -1 +1 @@
-Das ist ein gscheiter Text namens B
+Das ist ein dummer Text namens B
```

Ernsthafte Konflikte (2)

FASCHINGBAUER

Commit geht schief: im Repository ist eine neuere Version

```
$ svn commit -m 'B ist dumm'
Sending          b.txt
svn: E155011: Commit failed (details follow):
svn: E155011: File '/home/jfasch/checkout-trunk/b.txt' is out
svn: E160028: File '/trunk/b.txt' is out of date
```

- **Keine Panik**: Das ist normal
- Lösbar durch `svn update`
- Ausser ...

Ernsthafte Konflikte (3)

FASCHINGBAUER

Wie befohlen: wir bringen unsere Working Copy auf die neueste Version *und mischen sie unter unsere Modifikation* ...

```
$ svn update
Updating '.':
Conflict discovered in '/home/jfasch/checkout-trunk/b.txt'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options:
```

Alarm!

- Man muss sich einigen!

Ernsthafte Konflikte (4)

FASCHINGBAUER

Was ist der Streitfall?

```
Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: df
--- /home/jfasch/checkout-trunk/.svn/pristine/c8/c825ba772f72
Fri Feb 14 09:07:16 2014
+++ /home/jfasch/checkout-trunk/.svn/tmp/b.txt.tmp
Fri Feb 14 10:08:59 2014
@@ -1 +1,5 @@
-Das ist ein gscheiter Text namens B
+<<<<<<< .mine
+Das ist ein dummer Text namens B
+=======
+Das ist ein sehr gscheiter Text namens B
+>>>>>>> .r5
```

Ernsthafte Konflikte (5)

FASCHINGBAUER

Konfliktlösung

- “Postpone” ist eine gute Wahl
- Besprechung beim Kaffee
- … und dann …

```
vi b.txt
<<<<<<< .mine
Das ist ein dummer Text namens B
=======
Das ist ein sehr gscheiter Text namens B
>>>>>>> .r5
```

Ernsthafte Konflikte (6)

FASCHINGBAUER

Konflikt gelöst

- … nur weiss Subversion noch nichts davon
- Wie gehts weiter?
- Committen darf man noch nicht

svn status

```
$ svn status
C       b.txt
Summary of conflicts:
  Text conflicts: 1
```

- C: “conflicted” Status

Ernsthafte Konflikte (7)

FASCHINGBAUER

Wie sag ich's Subversion?

- Als "resolved" markieren

svn resolved

```
$ svn resolved b.txt
Resolved conflicted state of 'b.txt'
$ svn status
M       b.txt
```

svn commit

```
$ svn commit -m 'Wohlfuehlen nach Konflikt'
Sending        b.txt
Transmitting file data .
Committed revision 6.
```

Basic Workflow: Zusammenfassung (1)

FASCHINGBAUER

Was bisher geschah …

- Zentrales Repository, erreichbar … irgendwie
- `svn checkout`: Checkout einer *Working Copy* von der Entwicklungshauptlinie (`trunk`)
- `svn commit`: Rückführen von Änderungen ins Repository → sichtbar für andere
- `svn update`: Holen der Änderungen vom Repository
- Kaffee und `svn resolved`: Konfliktmanagement
- `svn diff`: Besichtigen meiner lokalen Änderungen
- `svn status`: Status der Working Copy
  - Viele Buchstaben → `svn status --help`

Basic Workflow: Zusammenfassung (2)

FASCHINGBAUER

Was fehlt, um den Workflow perfekt zu machen?

- Gleichzeitige Entwicklung nur kurzzeitig
  - Lokale Änderungen hängen in der Luft
  - Grosse Änderung → *ein* Riesencommit
  - → *Branches*
- Konfliktmanagement nur mit (zeilenbasierten) Textfiles
  - Binärfiles (und generierte Textfiles) schwierig
  - → *Locking* (halbherzig)

Overview

FASCHINGBAUER

1 Einleitung
2 Basic Workflow
3 Branches
4 Merging
5 Verschiedenes
6 Administration
7 Schlusswort

Branches: warum und wie (1)

FASCHINGBAUER

Probleme:

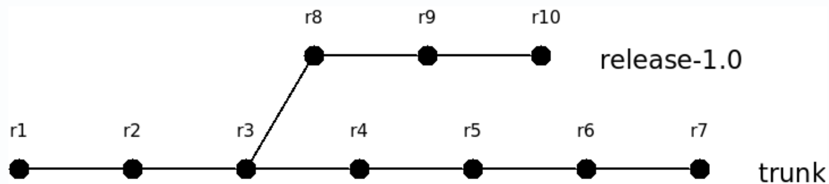
- Mehrere Entwickler arbeiten an einer Änderung
- ... und/oder die Änderung ist zu gross für einen einzigen Commit
- Man hat eine stabile Version, die nur mehr Bugfixes bekommt. Weiterentwicklung erfolgt *parallel*.

Lösung: Branches

- Parellel zum `trunk` im Repository
- Ansonsten gleichberechtigt → Entwickler checken dezidierte Working Copy aus
- Ein Branch kommt von woher → üblicherweise auch dahin zurückgeführt (*Merge*)

Branches: warum und wie (2)

FASCHINGBAUER



svn copy — Branch anlegen

FASCHINGBAUER

Branches sind Kopien

- Perfekte Analogie zur “manuellen” Versionskontrolle
- Branch vom Trunk: Kopiere den Trunk und arbeite dort weiter
- Branch vom Branch (Achtung: man kanns auch übertreiben): ...

Achtung: wenngleich das Konzept genial klingt, lässt es doch zuviele Freiheiten

- Man kann alles kopieren (auch einzelne Files) → völliges Durcheinander
- Man muss selbst Ordnung halten → widerspricht der menschlichen Natur
- → Regelwerk

Branching: Regeln

FASCHINGBAUER

Branching-Einheiten: *WAS* kopiere ich?

- Was ist die Einheit, die ich paketiere?
- C/C++: was ist die Einheit, die ich z.B. mit CMake massiere?
- Was ist die Einheit, von der ich eine Working Copy habe?
    - Sehr wahrscheinlich: die, die ich paketiere und massiere.
- → Das ist dann sehr wahrscheinlich auch die Einheit, die ich *kopiere*/*branche* — mit anderen Worten: *warte*

Repository Layout

FASCHINGBAUER

Problem: wo kopiere ich den Branch hin?

- Unsere Working Copy stammt von `svn+ssh://user@svn.repository.com/path/to/repo/trunk`
- "Administrator" hat dann vermutlich auch `/path/to/repo/branches` angelegt ...
- ... so wie es eine weitere Regel besagt

```
$ svn ls svn+ssh://user@svn.repository.com/path/to/repo
branches/
tags/
trunk/
```

Branch anlegen (1)

FASCHINGBAUER

Working Copy ins Repository kopieren

```
$ pwd
/home/jfasch/checkout-trunk
$ svn copy . \
   svn+ssh://user@svn.repository.com/path/to/repo/branches/re
```

- Kopiert den aktuellen Stand der Working Copy — die Revision, auf der sie basiert
- Ohne die lokalen Änderungen

Branch anlegen (2)

FASCHINGBAUER

Vom Repository ins Repository kopieren

```
$ svn copy \
   svn+ssh://user@svn.repository.com/path/to/repo/trunk \
   svn+ssh://user@svn.repository.com/path/to/repo/branches/re
```

- Geht komplett ohne Working Copy (→ schöne Managementaufgabe)
- **Achtung**: man weiss im Allgemeinen nicht, welche Revision man kopiert

Branch anlegen (3)

FASCHINGBAUER

Vom Repository ins Repository kopieren, mit expliziter Revision

```
$ svn copy \
   -r 42 \
   svn+ssh://user@svn.repository.com/path/to/repo/trunk \
   svn+ssh://user@svn.repository.com/path/to/repo/branches/re
```

- Man weiss, was man hat: 42 ist die letzte gute Revision

Working Copy von Branch anlegen

FASCHINGBAUER

Working Copy vom Branch `release-1.0`

- Wir haben bis jetzt eine Working Copy, von `trunk`
- Warum sollen wir nicht gleichzeitig am Branch `release-1.0` arbeiten?
- → Zweite Working Copy, nur eben von dort

```
$ svn checkout \
   svn+ssh://user@svn.repository.com/path/to/repo/branches/re
```

Workflow am Branch

FASCHINGBAUER

Was ist jetzt anders? Ich arbeite am Branch `release-1.0` und nicht am `trunk` — was ändert sich?

- *Nichts!*
- ... ausser, man will öfter man Änderungen vom `trunk` übernehmen
- ... ausser, man will die Änderungen von `release-1.0` mal in `trunk` übernehmen

Konflikte! Wie im bisherigen `trunk` Betrieb wird parallel entwickelt

- Am `release-1.0` von mehreren Entwicklern gleichzeitig
- Änderungen am `trunk` sind *zusätzlich*
- ... und diese Änderungen sind gross und langdauernd
- → Konflikte sind sehr wahrscheinlich!

Overview

1 Einleitung
2 Basic Workflow
3 Branches
4 Merging
5 Verschiedenes
6 Administration
7 Schlusswort

Merging: was ist das?

FASCHINGBAUER

Problem

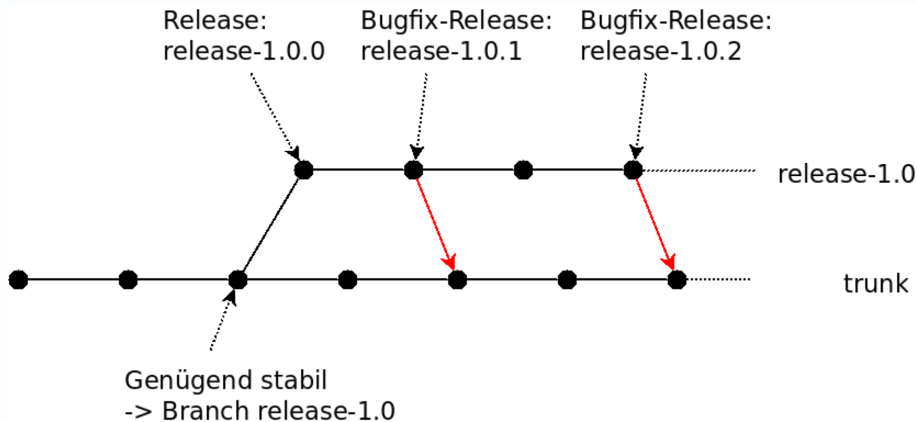
- Branch: Kopie mit eigenen Commits
- Ist sie noch mit dem Original verwandt?
  - $\rightarrow$ klar, dafür gibts Subversion
  - *Merging*: Kopie und Original abgleichen

Use Cases

- *Releasebranch*: Weiterentwicklung nur für Bugfix-Releases $\rightarrow$ Fixes müssen zurück in den `trunk`, damit die weiteren Releases davon profitieren
- *Entwicklungsbranch*: Weiterentwicklung in einer sicheren Umgebung $\rightarrow$ man will sukzessive Änderungen vom `trunk` übernehmen

Release — Lebenszyklus (1)

FASCHINGBAUER



Release — Lebenszyklus (2)

FASCHINGBAUER

Merkmale

- Release-Branch lebt ewig
- Nur Bugfixes
  - Gebugfixter Code wird auch anderswo verwendet
  - Merges passieren nur *in eine Richtung*: zu den *Interessenten* des Bugfixes
- Keine neuen Features
  - Wenn doch: Fehler im “Release-Management”

svn merge — Grundlegende Verwendung (1)

FASCHINGBAUER

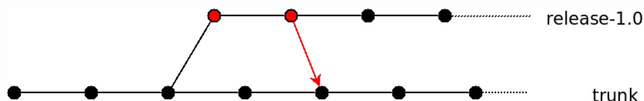
```
$ pwd
/home/jfasch/checkout-trunk
$ svn merge \
   svn+ssh://user@svn.repository.com/path/to/repo/branches/re
# kürzer ...
$ svn merge ^/branches/release-1.0
```

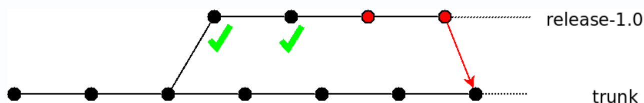
- **Konflikte** werden gleich behandelt wie schon erklärt!

`svn merge` — Grundlegende Verwendung (2)

FASCHINGBAUER

- Übernimmt alle Änderungen vom Branch `release-1.0` in die Working Copy, in der wir gerade stehen
- Mehrmals hintereinander möglich → “Merge Tracking”
- → … alle Änderungen, die noch nicht übernommen wurden





Releasemanagement — Tags

FASCHINGBAUER

"Taggen" von Releases

- Man will den Stand einer Release archivieen
- Revision alleine ist zu wenig (Zahlen sprechen nicht)
- Archivieren ist "Kopieren"

```
$ svn copy . \
   svn+ssh://user@svn.repository.com/path/to/repo/tags/release
```

Kürzer ...

```
$ svn copy . ^/tags/release-1.0.1
```

Releasemanagement — Letzte Worte

FASCHINGBAUER

Releasemanagement hat nur entfernt mit Versionskontrolle zu tun

- Prozessdefinition
- ... formal oder informal
- Versionskontrolle ist nur ein Mittel zum Zweck
- Fragen:
    - Wann wird gebranched?
    - Welchen Teil der Entwicklung mach ich am Branch weiter? (Stabilisierung)
    - Habe ich neue Features, die schon am Trunk entwickelt werden wollen?

Entwicklungsbranches

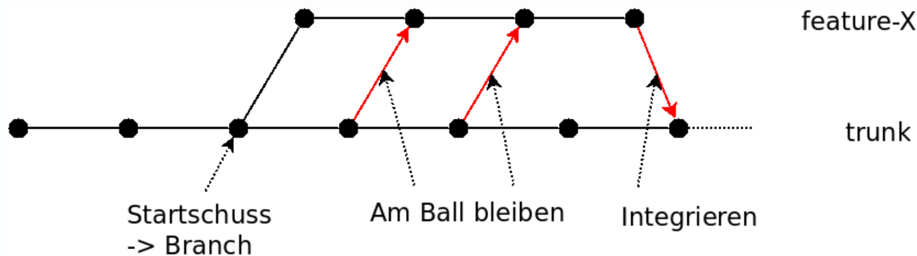
FASCHINGBAUER

Problem, wenn man Features am Trunk entwickelt

- Grosses Feature
  - Entwicklung dauert lange
  - Man will mitunter committen
  - $\rightarrow$ sofort sichtbar am Trunk
- Mehrere Entwickler arbeiten daran
  - Austausch nur über Trunk

Entwicklungsbranch — Lebenszyklus

FASCHINGBAUER



Merkmale:

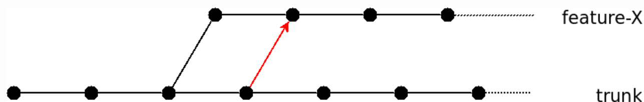
- Öfter mal werden Änderungen vom Trunk übernommen
    - Auseinanderdriften verhindert
    - → konfliktfreie(re) Integration
- Fertig → Integration → tot

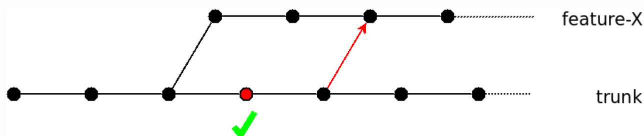
Entwicklungsbranch — Am Ball bleiben

FASCHINGBAUER

Am Ball bleiben: einfach Mergen vom Trunk

```
$ pwd
$ /home/jfasch/checkout-feature1
$ svn merge ^/trunk
```





Entwicklungsbranch — Integrieren

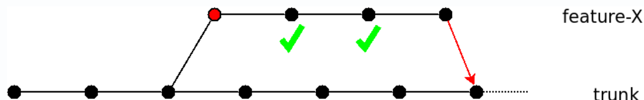
FASCHINGBAUER

Feature ist fertig: “Integrieren”

```
$ pwd
$ /home/jfasch/checkout-trunk
$ svn merge --reintegrate ^/branches/feature1
```

Warum “Reintegrate” und nicht einfach “Merge”?

- “Am Ball bleiben” hat Changes vom trunk übernommen
- Die wollen wir nicht mitintegrieren



Mini-Workflow — Cherry-Picking

FASCHINGBAUER

Rosinen rauspicken: Übernehmen von einzelnen Commits

- Gezielte Bugfixes von einem Release-Branch in den anderen (“Backporting”)
- “Überschneidungen” von Entwickler-Branches (einer fixt einen Bug, der andere übernimmt den Fix)
- ...

Cherrypicking der Revisions 1234 und 3456

```
$ pwd
$ /home/jfasch/checkout-feature2
$ svn merge -c 1234,3456 ^/branches/feature1
```

Mini-Workflow — Rückgängig machen

FASCHINGBAUER

Reverting: Rückgängig machen einer Änderung (oder einer Reihe von Änderungen)

- "Reverse Differences"
- `diff -u`: + und − umkehren
    - "negative" Revision-Nummern (umgekehrtes Cherry-Picking)
    - Verkehrte Revision-Ranges

```
$ pwd
$ /home/jfasch/checkout-trunk
$ svn merge -r -42 ^/trunk
$ svn merge -r 650:640 ^/trunk
```

Branching & Merging — Schlusswort

FASCHINGBAUER

Versionskontrolle

- Löst Probleme
- Macht Probleme
- **Ersetzt keinesfalls Kommunikation**
- Konflikte werden von Menschen gemacht, nicht von Versionskontrollsystemen

Releasemanagement

- Muss auch gemacht werden
- Ist u.a. ein formalisierter Weg zur Konfliktvermeidung

Overview

FASCHINGBAUER

1 Einleitung
2 Basic Workflow
3 Branches
4 Merging
5 Verschiedenes
6 Administration
7 Schlusswort

Geschichtsforschung — History

FASCHINGBAUER

Subversion vergisst nichts

- `svn log` Liste von Changes
  - `svn log --stop-on-copy`: Herausfinden, von wo kopiert (gebranched) wurde
- `svn diff`: Änderungen im *Unified Diff* Format (`diff -u`)
- `svn blame`: Annotieren eines Files (einer Revision davon, ...) — wer hats wann in welcher Revision verbockt?

Ignorierte Einträge

FASCHINGBAUER

Subversion weiss nichts von ...

- Editor-Backupfiles
- Compilate
- ...

Man muss ihm explizit sagen, dass es diese ignorieren soll → Properties

```
$ svn propedit svn:ignore .
```

`$EDITOR` → Eintragen der zu ignorierende Einträge

```
*~
*.o
some-generated-image.png
```

Binäre Files

FASCHINGBAUER

Subversion kann nur zeilenbasiert mergen

- Binäre Files haben unter Umständen sehr lange Zeilen
- → *gleichzeitige Änderungen sollte man vermeiden*

```
$ svn propset svn:needs-lock 1 bild.jpg
$ svn propdel svn:needs-lock bild.jpg
```

- File wird *read-only* ausgecheckt
- Explizite *Sperre* nötig → exklusiv

```
$ svn lock bild.jpg
```

Executable Files

FASCHINGBAUER

Subversion und executable Files

- Compilierte Files, die exekutierbar sind, sollten nicht eingecheckt werden ($\rightarrow$ `svn:ignore`)
- Aber was ist mit Scripts?

```
$ svn propset svn:executable 1 script.sh
$ svn propdel svn:executable script.sh
```

- File wird beim nächsten Checkout executable gemacht

Overview

FASCHINGBAUER

1 Einleitung
2 Basic Workflow
3 Branches
4 Merging
5 Verschiedenes
6 Administration
7 Schlusswort

Zugriffsmethoden

FASCHINGBAUER

Viele Methoden, um auf ein Repository zuzugreifen ...

- `file://` — Lokale Disk
- `svn://` — SVN-Protokoll ohne Authentifizierung
- `svn+ssh://` — SVN-Protokoll über SSH
- `http://` — Anonym über HTTP/WebDAV
- `https://` — HTTP/SSL/WebDAV

Zum Beispiel:

- `file:///tmp/repo/tags/REL-1.1`
- `svn+ssh://user@server/svn/proj/trunk`
- `https://server.net/repo/tags/REL-1.3`

Overview

FASCHINGBAUER

1. Einleitung
2. Basic Workflow
3. Branches
4. Merging
5. Verschiedenes
6. Administration
7. Schlusswort

Schlusswort

FASCHINGBAUER

- Subversion ist nicht perfekt
  - Branches zu machen ist leicht — Mergen hingegen schwer
  - Warum ist `--reintegrate` nötig?
  - Linus Torvalds: "… the most pointless project that has ever been started"
- Aber:
  - Unendlich leichter zu verstehen als z.B. Git
  - Zentrales Repository hat auch seine Vorteile (wenn auch sich nur das Management sicher fühlt)