Versionsverwaltungssysteme

Severin Neumann

IEEE Student Branch Passau

24. Mai 2012

Motivation

Wir möchten Versionsverwaltungssysteme verwenden, um folgende Ziele zu erreichen:

Motivation

Wir möchten Versionsverwaltungssysteme verwenden, um folgende Ziele zu erreichen:

1. Gemeinsames Arbeiten vereinfachen.

Motivation

Wir möchten Versionsverwaltungssysteme verwenden, um folgende Ziele zu erreichen:

1. Gemeinsames Arbeiten vereinfachen.
2. Archivierung alter Versionen.

Motivation

Wir möchten Versionsverwaltungssysteme verwenden, um folgende Ziele zu erreichen:

1. Gemeinsames Arbeiten vereinfachen.
2. Archivierung alter Versionen.
3. Aktuelle Version einer Datei mit einer alten vergleichen.

Motivation

Wir möchten Versionsverwaltungssysteme verwenden, um folgende Ziele zu erreichen:

1. Gemeinsames Arbeiten vereinfachen.
2. Archivierung alter Versionen.
3. Aktuelle Version einer Datei mit einer alten vergleichen.
4. Protokollierung aller Änderungen.

Theorie

Das Arbeiten mit einem Versionierungssystem läuft beispielsweise folgendermassen ab:

1. Man erstellt ein Projekt in einem Versionsverwaltungssystem. (`create`)

Theorie

Das Arbeiten mit einem Versionierungssystem läuft beispielsweise folgendermassen ab:

1. Man erstellt ein Projekt in einem Versionsverwaltungssystem. (`create`)
2. **oder** man holt eine Arbeitskopie eines Projekts aus dem *Repository* eines Versionsverwaltungssystem (`checkout`).

Theorie

Das Arbeiten mit einem Versionierungssystem läuft beispielsweise folgendermassen ab:

1. Man erstellt ein Projekt in einem Versionsverwaltungssystem. (`create`)
2. **oder** man holt eine Arbeitskopie eines Projekts aus dem *Repository* eines Versionsverwaltungssystem (`checkout`).
3. Man bearbeitet die Dateien in seiner Arbeitskopie.

Theorie

Das Arbeiten mit einem Versionierungssystem läuft beispielsweise folgendermassen ab:

1. Man erstellt ein Projekt in einem Versionsverwaltungssystem. (`create`)
2. **oder** man holt eine Arbeitskopie eines Projekts aus dem *Repository* eines Versionsverwaltungssystem (`checkout`).
3. Man bearbeitet die Dateien in seiner Arbeitskopie.
4. Man überträgt die Änderungen in das Repository und gibt dazu eine Logmeldung mit.

Theorie

Das Arbeiten mit einem Versionierungssystem läuft beispielsweise folgendermassen ab:

1. Man erstellt ein Projekt in einem Versionsverwaltungssystem. (`create`)
2. **oder** man holt eine Arbeitskopie eines Projekts aus dem *Repository* eines Versionsverwaltungssystem (`checkout`).
3. Man bearbeitet die Dateien in seiner Arbeitskopie.
4. Man überträgt die Änderungen in das Repository und gibt dazu eine Logmeldung mit.
5. Man vergleicht Versionen des Projekts untereinander (`diff`)

Theorie

Das Arbeiten mit einem Versionierungssystem läuft beispielsweise folgendermassen ab:

1. Man erstellt ein Projekt in einem Versionsverwaltungssystem. (`create`)
2. **oder** man holt eine Arbeitskopie eines Projekts aus dem *Repository* eines Versionsverwaltungssystem (`checkout`).
3. Man bearbeitet die Dateien in seiner Arbeitskopie.
4. Man überträgt die Änderungen in das Repository und gibt dazu eine Logmeldung mit.
5. Man vergleicht Versionen des Projekts untereinander (`diff`)
6. Man stellt eine alte Version wieder her. (`revert`)

Theorie

Das Arbeiten mit einem Versionierungssystem läuft beispielsweise folgendermassen ab:

1. Man erstellt ein Projekt in einem Versionsverwaltungssystem. (`create`)
2. **oder** man holt eine Arbeitskopie eines Projekts aus dem *Repository* eines Versionsverwaltungssystem (`checkout`).
3. Man bearbeitet die Dateien in seiner Arbeitskopie.
4. Man überträgt die Änderungen in das Repository und gibt dazu eine Logmeldung mit.
5. Man vergleicht Versionen des Projekts untereinander (`diff`)
6. Man stellt eine alte Version wieder her. (`revert`)
7. ...

Theorie

Das Arbeiten mit einem Versionierungssystem läuft beispielsweise folgendermassen ab:

1. Man erstellt ein Projekt in einem Versionsverwaltungssystem. (`create`)
2. **oder** man holt eine Arbeitskopie eines Projekts aus dem *Repository* eines Versionsverwaltungssystem (`checkout`).
3. Man bearbeitet die Dateien in seiner Arbeitskopie.
4. Man überträgt die Änderungen in das Repository und gibt dazu eine Logmeldung mit.
5. Man vergleicht Versionen des Projekts untereinander (`diff`)
6. Man stellt eine alte Version wieder her. (`revert`)
7. ...

Es folgt ein Praxisbeispiel (mit Subversion).

Subversion – Gelernte Befehle

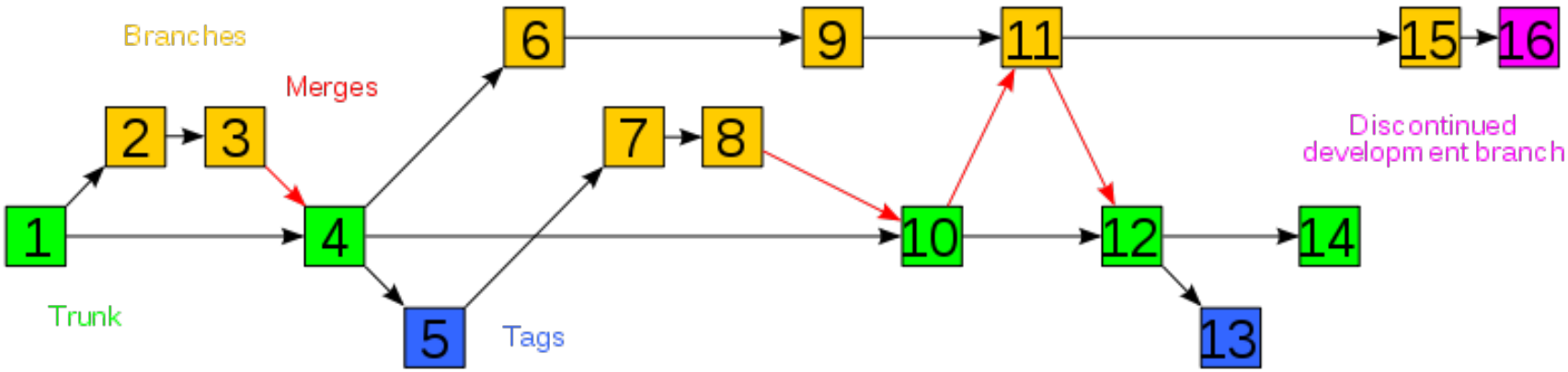
1. `svn co https://svn.example.com/svn/project .` erstellt eine Arbeitskopie des Projekts *project*.
2. `svn up` aktualisiert die Arbeitskopie.
3. `svn add` fügt eine Datei zur Arbeitskopie hinzu.
4. `svn del` löscht eine Datei aus der Arbeitskopie.
5. `svn mv` verschiebt eine Datei in der Arbeitskopie.
6. `svn ci` sendet die Änderungen an das Versionsverwaltungssystem.
7. `svn diff` zeigt die Unterschiede zwischen Versionen an.

Subversion – Gelernte Befehle

1. `svn co https://svn.example.com/svn/project .` erstellt eine Arbeitskopie des Projekts *project*.
2. `svn up` aktualisiert die Arbeitskopie.
3. `svn add` fügt eine Datei zur Arbeitskopie hinzu.
4. `svn del` löscht eine Datei aus der Arbeitskopie.
5. `svn mv` verschiebt eine Datei in der Arbeitskopie.
6. `svn ci` sendet die Änderungen an das Versionsverwaltungssystem.
7. `svn diff` zeigt die Unterschiede zwischen Versionen an.

Hinweis: Natürlich gibt es noch viel mehr Möglichkeiten. Diese Funktionen decken für einen *Einsteiger* 90% ab, für alles andere gibt es `svn help`.

Subversion – Ein Ausblick



Subversion – Grafische Benutzeroberflächen

1. Subclipse und Subversive für Eclipse
2. Netbeans
3. TortoiseSVN für Windows
4. KDESvn
5. rabbitvcs (Nautilus)
6. thunar-vcs-plugin

svn oder nicht svn?

Subversion ist nur eines von vielen Versionsverwaltungssystemen, eine interessante Alternative ist `git`:

svn oder nicht svn?

Subversion ist nur eines von vielen Versionsverwaltungssystemen, eine interessante Alternative ist `git`:

1. `git` ist dezentral.

svn oder nicht svn?

Subversion ist nur eines von vielen Versionsverwaltungssystemen, eine interessante Alternative ist `git`:

1. `git` ist dezentral.
2. `git` bietet kryptographische Sicherheit für die Projektgeschichte.

svn oder nicht svn?

Subversion ist nur eines von vielen Versionsverwaltungssystemen, eine interessante Alternative ist `git`:

1. `git` ist dezentral.
2. `git` bietet kryptographische Sicherheit für die Projektgeschichte.
3. `git` ist (subjektiv!, für Umsteiger?) gewöhnungsbedürtiger in der Bedienung.

svn oder nicht svn?

Subversion ist nur eines von vielen Versionsverwaltungssystemen, eine interessante Alternative ist `git`:

1. `git` ist dezentral.
2. `git` bietet kryptographische Sicherheit für die Projektgeschichte.
3. `git` ist (subjektiv!, für Umsteiger?) gewöhnungsbedürtiger in der Bedienung.
4. Auch für `git` gibt es verschiedene grafische Oberflächen und Integrationen in IDEs

svn oder nicht svn?

Subversion ist nur eines von vielen Versionsverwaltungssystemen, eine interessante Alternative ist `git`:

1. `git` ist dezentral.
2. `git` bietet kryptographische Sicherheit für die Projektgeschichte.
3. `git` ist (subjektiv!, für Umsteiger?) gewöhnungsbedürtiger in der Bedienung.
4. Auch für `git` gibt es verschiedene grafische Oberflächen und Integrationen in IDEs

Empfehlung (auch subjektiv): Wer schnell und einfach einsteigen will, sollte `svn` verwenden. Wer jedoch die oben genannten Features verwenden will oder größere Projekte plant, sollte `git` probieren.

Repositories

1. Ein Repository für eure Projekte, Übungsblätter, Abschlußarbeiten oder das SEP könnt ihr bei unseren CIP Admins beantragen.

Repositories

1. Ein Repository für eure Projekte, Übungsblätter, Abschlußarbeiten oder das SEP könnt ihr bei unseren CIP Admins beantragen.
2. Öffentliche Projekte können z.B. bei `gitorious` oder `github` angelegt werden.

Repositories

1. Ein Repository für eure Projekte, Übungsblätter, Abschlußarbeiten oder das SEP könnt ihr bei unseren CIP Admins beantragen.
2. Öffentliche Projekte können z.B. bei `gitorious` oder `github` angelegt werden.
3. Um eigene Repositories zu betreiben, befragt das Internet.