Versionskontrolle mit Mercurial

Michael Gebetsroither
<michael.geb@gmx.at>

Inhalt

- Welche Systeme gibt es?
  - Genauerer Überblick.
- Mercurial:
  - Einführung
  - Grundlagen
- test

Versionskontrollsysteme

- CVS
- SVN

- Mercurial
- Git
- Darcs
- Bazaar-NG
- Arch
- SVK
- Monotone

CVS

- System: Zentral
- Programmiersprache: C
- Veröffentlicht: 1986
- Eigenschaften:
  - Verschieben / umbenennen unversioniert
  - Keine symbolischen Links
  - Eingeschränkte Unicode Unterstützung
  - Keine Transactions

SVN

- System: Zentral
- Programmiersprache: C
- Veröffentlicht: 2002
- Eigenschaften:
  + Weit verbreitet.
  + Sehr gute Toolchain
  + Transactions
  - Langsam
  - Verbindung zum Server erforderlich

Monotone

- System: dezentral
- Programmiersprache: C++
- Veröffentlicht: 2003
- Eigenschaften:
  + Sehr solides System
  - Verbesserungswürdige Usability
  - Pro Benutzer üblicherweise eine SQLite db
  - Daher schlecht zum Experimentieren

Darcs

- System: Dezentral
- Programmiersprache: Haskell
- Veröffentlicht: 2002
- Eigenschaften:
  + „Theory of patches“
  - Merges können **sehr** lange dauern oder ganz fehlschlagen

Bazaar-NG

- System: Dezentral
- Programmiersprache: Python
- Veröffentlicht: 2005
- Eigenschaften:
  + Alle nötigen Funktionen
  + Usability
  + Plattformübergreifend
  - Für große Projekte zu langsam

Mercurial

- System: Dezentral
- Programmiersprache: Python
- Veröffentlicht: 2005
- Eigenschaften:
  - + **Sehr** schnell
  - + Repositories portabel
  - + Gute Usability

Git

- System: Dezentral
- Programmiersprache: C
- Veröffentlicht: 2005
- Eigenschaften:
  + **Sehr** schnell
  - Usability
  - Nicht portabel

Fragen?

Anforderungen

- Augenmerk soll nicht am SCM liegen.
- Einfach
- Effektive/einfache Zusammenarbeit
- Effizientes Arbeiten
- Konsistenter Workflow

Mercurial ist ...

- ... ein verteiltes Versionskontrollsystem
  - Nichtlineares arbeiten und einfaches mergen.
- ... einfach
- ... effiziente Zusammenarbeit
- ... **sehr** schnell
- ... GPL

=> Für große wie auch für kleine Projekte geeignet!

Wer verwendet Mercurial

- Xen
- Mozilla
- rpm.org
- Opensolaris
- Alsa
- LinuxTV (V4L)
- Xine
- Grml

Synchronisierte Mirrors:

- Linux Kernel
- FreeBSD
- GNU Emacs
- DragonFly BSD
- pkgsrc

Einfach

- Das Konzept von Mercurial ist leicht verständlich und einfach gehalten um keine unnötige Barriere zu schaffen

- 3 Grundlagen:
  - Repository
  - Changeset
  - Working directory

Repository ?

- Your Friend!
- Gesamte Projekthistory.
- .hg/
- Schnelles dublizieren („clonen“).
- Robuste append-only Struktur.
- Jeder hat eines.

Changeset ?

- Ein Snapshot
- Wichtigste aufgezeichnete Informationen:
  - Von wem (Comitter)
  - Was geändert (welche Dateien)
  - Kurzbeschreibung
  - Auf welchem Changeset basierend
- Commit produziert einen Changeset.

Working Directory ?

- Zustand des Repositories bei einem bestimmten Changeset.
- Frei editierbar
  - Änderungen werden in den nächsten Changeset aufgezeichnet.
  - Ändern/Löschen/Umbenennen/Kopieren
- Genau ersichtlich was/wie geändert wurde.

Zentral vs. Dezentral

- Zentrales Repository
- Flaschenhals Server
- Skalierungsprobleme
- Störende Latency
- Backup unbedingt erforderlich
- Server weg und aus

- Optional
- Selten benutzt
- Viele „gratis“ Mirror
- Alle Daten Lokal
- Jedes Repository ist ein Backup
- Kein Problem

Mercurial in 60s

- Initialisieren — hg init
- Wechseln — cd foo
- Änderungen — vi bar
- Hinzufügen — hg add bar
- Aktueller Status? — hg status
  - Datei hinzugefügt — *A bar*
- Einfrieren — hg commit

Fragen?

Einfache Zusammenarbeit

- Mehr als ein Entwickler arbeitet immer parallel.
  - Die meisten SCMs erschweren das unnötig.
- Änderungen machen.
- Comitten.
- War jemand schneller?

Bekanntes Problem

- Jemand war schneller!
- Man muss mergen bevor man die eigenen Änderungen comitten kann.
- Die eigenen Änderungen hängen in der Luft.
- Ein Fehler beim Mergen und die eigenen Änderungen gehen u.u verloren.

Mercurial: Branches

- Ein Changeset hat einen Parrent.
- Zwei Changesets mit dem gleichen Parrent => Branch.
- Der Parrent ist der Branchpoint.
- kompliziert?

Mercurial: Merges

- Was also jetzt tun mit den Branches?
- Manche Changesets haben Zwei Parrents.
- Das sind Merge-Changesets.
- Er gibt an wie zwei Branches zusammengefügt werden.

Mercurial Distributed

- Unabhängiges verteiltes Arbeiten
- Freier Commitzeitpunkt
- Freier Mergezeitpunkt
- Erst comitten, dann mergen!
- Probleme beim Mergen?
  - Kein Problem, einfach nochmal versuchen.

Sharing einmal anders

- Änderungen
- Comitten
- Was geht raus?
- Vom Server
- Mergen
- Zum Server

- vi foo
- hg commit foo
- hg out
- hg pull
- hg merge
- hg push

Sharing

- Eingebauter multithreaded Webserver.
- CGI / FCGI Script
- Lieber sicher? -> ssh
- Oder lieber sshfs / nfs?
- Die letzte Rettung: email
  - Bundles (internes changeset format)
  - Exports (diff mit Zusatzinformationen)
  - Diff (der gute alte Patch)

Fragen?

Moving Target

- Upstream vorhanden
- Patches gegen Upstream
- Patches sollen versioniert sein
- Leichtes adaptieren an neue Upstream Versionen
- Bestmögliche Unterstützung

=> (M)ercurial Patch (Q)ueues, MQ

MQ ?

- Eine Serie von Patchen die auf das normale Repository sequenziell angewandt wird.
- .hg/patches/
- Auf Wunsch die Patches selbst versioniert.
- Basierend auf quilt.
- Skaliert von wenigen bis zu mehreren Tausend Patches.

Einfach MQ

- Upstream aktell
- MQ initialisieren
- Der erste Patch
- Änderungen
- Patch anpassen
- Patch einfrieren

- hg pull -u
- hg qinit -c
- hg qnew
- vi foo
- hg qrefresh
- hg qcommit

Fragen?

Links

- Mercurial: http://www.selenic.com/mercurial
- Grml: http://grml.org
- Git: http://git.or.cz
-