Mercurial

Verteilte Versionskontrolle

Reinhard Wobst

r.wobst@gmx.de

UNIX-Stammtisch Dresden, 5.5.2010

Inhaltsverzeichnis

1. Einführung

Versionskontrollsystem =

- Revision Control system (RCS)
- Software Configuration Management (SCM)
- Source Code Management
- Source Code Control
- Version Control System (VCS)
- ... alles dasselbe

Wichtigste Vertreter verteilter Systeme:

Mercurial, Git, Bazaar

1.1. Was ist ein DVCS (distributed version control system)?

- neue Generation von VCS, wird allgemein als die Zukunft angesehen (obwohl es aktuell nur genutzte 2-3 Vertreter gibt)
- kein zentrales Repository
- ohne äußere Vorgaben sind alle Repositories gleichberechtigt, keine zentrale Nummerierung!
- Repositories werden kopiert oder abgeglichen

1.2. Vorteile

- Man ist unabhängig von Netzverbindungen (Notebook im Hotel ...)

- keine Überlastung zentraler Server wegen ständiger kleiner Patches
- Sandbox-Entwicklung bei Experimenten, kann wieder verworfen werden, bei Erfolg aber eingecheckt
- neue Entwicklungen können auch problemlos in Sandbox von anderen getestet werden

1.3. Nachteile

- merge hat prinzipielle Gefahren (vgl. 4.3.), auch wenn es in Mercurial leichter als in früheren Systemen ist
- Infrastruktur für Nutzung muss vorgegeben werden, sonst Chaos (gilt aber generell für Softwareentwicklung ☺)

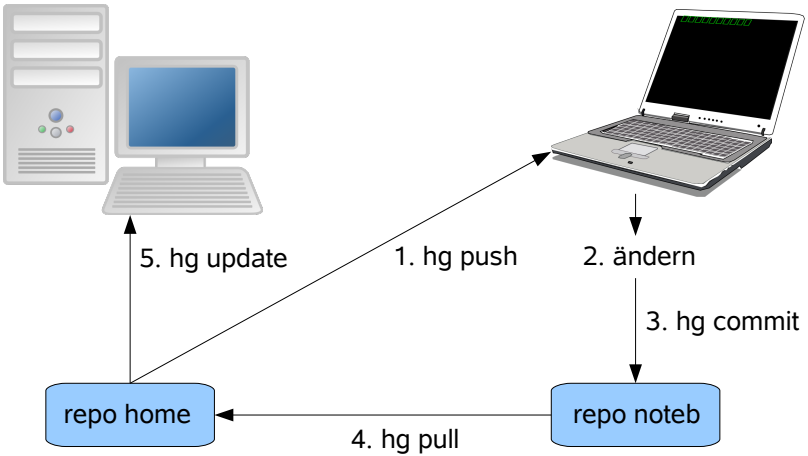
- Absichtlich wird standardmäßig nur die gesamte History kopiert (nur bedingt Nachteil, ist so gewollt und kann vermieden werden - vgl. 7. , ConvertExtension und rebase)

1.4. Arbeit mit DVCS

- Repository clonen oder aktualisieren (pull), oder neu einrichten (s.u.)
- lokal lineare Entwicklung oder mit Zweigen
- backport: push (Entwickler auf Server) or pull (durch zentrale Instanz), merge (Verschmelzen verschiedener, parallel entstandener Zweige)
- "pull" und "branch/merge" sind Grundoperationen!

1.5. Anwendungs-Szenarien

- **Abgleich** verschiedener Rechner, z.B.
  - Folien von Vortrag werden auf Notebook in Pausen und auf Heimweg korrigiert und erweitert
  - im Büro/zu Hause aktualisiert man das Repository auf dem Hauptrechner sowie dessen Arbeitsverzeichnis
  - Entwickelt Vortrag auf Hauptrechner weiter
  - man aktualisiert anschließend das Repository auf dem Notebook automatisch nur mit den während des Vortrags wirklich benötigten Files (interessant auch bei USB-Sticks!):


5. hg update
1. hg push
2. ändern
3. hg commit
repo home
repo noteb
4. hg pull

- **Softwareentwicklung für externen Partner:**
  - branch for debugging: kleinen Seitenzweig einrichten mit Release, das Debug-Ausgaben enthält. Danach Fortsetzung im Hauptzweig. Vorteil: Syntax der Debug-Ausgaben jederzeit abrufbar, kann auf andere Quellen übertragen werden; Ausgaben können im Seitenzweig mit abgelegt werden.

- **gemeinsame Softwareentwicklung:**
  - pull-only-Modell (Linux-Kernel, man bietet Update an, führt nie "push" aus)
  - konservatives Modell mit zentralem Repository:
    - hg pull
    - hack, hack ...
    - hg push
    - hg merge in zentralem Repository!
    - File-Locking über commit-Hooks möglich (können zentral erzwungen werden)

- bei falscher Entwicklung: zurück zu funktionierendem Release (Hilfe dabei mit `hg bisect`, vgl. 7.), ab dort default-Zweig branchen

2. Über Mercurial

- geboren 19. April 2005 von Matt Mackall (männlich)
- etwa so alt wie Git
- geschrieben in Python, etwas C

2.1. Projekte in Mercurial

- *Mozilla*:
  - Firefox
  - Thunderbird
- *Google*:
  - Google Chrome
  - Google Code
- *Sun*:
  - OpenOffice
  - OpenSolaris
  - Xen
  - OpenJDK
- *Microsoft:* CodePlex
- *Python, vim* und natürlich *Mercurial* selbst ☺
- einige Entwickler des Linux-Kerns

(Quelle: Wikipedia)

2.2. Besonderheiten von Mercurial

- portabel (!) - Unterschied zu Git (Git ist problematisch unter Windows und sehr auf Linux-Kernelentwicklung orientiert)
- kleiner als Git, (vergessenes) Toolbox-Prinzip von UNIX: Eine Applikation muss nicht alles können!
- rename/remove/copy Tracking (Git/Bazaar: kein Copy; Git: rename "nicht explizit"): `hg rename`
- history by default append-only

2.3. Vergleiche mit anderen DVCS

- http://rg03.wordpress.com/2009/04/07/mercurial-vs-git/
- https://developer.mozilla.org/en/Mercurial_FAQ (derzeit tot)
- http://sayspy.blogspot.com/2006/11/bazaar-vs-mercurial-unscientific.html

Derzeit ringen offenbar nur noch Mercurial und Git um Vorherrschaft (soweit nötig), Bazaar scheint schon abgeschlagen zu sein.
In Funktionsumfang, Geschwindigkeit und Skalierbarkeit unterscheiden sich Git wie Mercurial offenbar nicht wesentlich, man

sagt aber Mercurial bessere Hilfe und Dokumentation und vor allem leichteren Einstieg voraus. Git versucht alles zu integrieren, Mercurial setzt eher auf Extensions.

Git oder Mercurial?

Zitat aus einem Forum: *"To paraphrase Colin Wheeler, it's OK to proselytize to those who have not switched to a distrubuted VCS yet, but trying to convert a git user to Mercurial (or vice-versa) is a waste of everyone's time and energy."*

Also:

```
Git ^ Mercurial == vim ^ emacs == perl ^ python
    == documentation ^ read_the_source,luke
```

3. Grundbefehle

3.1. Einstieg

- `hg help command`

  Damit steigt man ein, braucht man laufend!

  Hinweis: Bei der Hilfe müssen leider immer vollständige Kommandonamen angegeben werden, bei Befehlen reichen Kürzel. show

3.1.1. Neues Repository anlegen

Dateien einrichten:

- `.hgignore`

  File mit Ausnahmeliste (nicht zum Committen bestimmt), Shellmetazeichen und reguläre Ausdrücke können gemischt werden (Negation wie bei Git ist mit Absicht nicht vorgesehen);

  show

- `.hgrc`

  Konfigurationsfile mit Aliases, Extensions u.a., s.u.; global und Repository-lokal möglich (vgl. 6.).

Startbefehle:

- `hg init`

  Erzeugt ein neues Repository (andere Methode: `clone`, s.u.), noch keine Files enthalten! Damit ist unter **.hg/** das Repository eingerichtet (und nur dort; durch Löschen des Baumes .hg/ deaktiviert man Mercurial für das Arbeitsverzeichnis) - der "Baum drumherum" heißt etwas ungenau **Arbeitsverzeichnis**.

- `hg add filename[s] directory[s]`
  `hg addremove`
  `# löscht zusätzlich fehlende Dateien`

  Merkt Files für das Committen vor (Gegenteil: `hg forget`); beachte Schalter `-X` (exclude patterns) und `-I` (include

patterns); **hg add** allein merkt alles für commit vor (wobei **.hgignore** berücksichtigt wird). **Beispiel**:

```
hg add *.txt 'glob:**.{ch}'
hg add 're:.*\.c$'
hg add .hgignore  # !!
```

- **"Positivliste":** In `.hgignore` nur ".*" angeben (alles ignorieren), dann `hg add` mit Script-generierter Liste (kann jederzeit wiederholt werden, erneutes `hg add` des gleichen Files ist unwirksam).

3.1.2. Arbeit mit lokalem Repository

Bemerkung: hg-Kommandos können von jedem Unterverzeichnis aus gestartet werden und suchen dann das root-Verzeichnis (wo `.hg/` liegt).

Einige Kommandos:

- `hg com(mit) [files]`
  `hg ci    # eingebautes Alias`

  Committen (check in); i.a. committed man alle geänderten Files auf einmal (ein **changeset**)

- `hg log`
  `hg glog    # extension`

  Zeigt Revisionshistory an **(show)**

  - `hg serve`

  Webserver - kann per default mit localhost:8000 bedient werden **(show):**


/home1/wobst/tmp/tmp/condfield
graph
log
graph
tags
branches
changeset
browse
merged default tip
91 seconds ago, by Reinhard Wobst
description extended
10 minutes ago, by Reinhard Wobst
Mailadr to netic
9 minutes ago, by Reinhard Wobst
Version 0.7
14 minutes ago, by Reinhard Wobst
Version 0.7
18 minutes ago, by Reinhard Wobst
Version 6
17 minutes ago, by Reinhard Wobst
Version 0.4
20 minutes ago, by Reinhard Wobst
new version 0.3
23 minutes ago, by Reinhard Wobst
Added tag cond_OK? for changeset 70dd3f587ea1
11 days ago, by Reinhard Wobst
_run variable in condfield2.py - temporary reset default value, works here cond_OK?
11 days ago, by Reinhard Wobst
Added tag debug_base for changeset bd076242e4d9
13 days ago, by Reinhard Wobst
In condfield2 self.env: default_type auf ticket_type gesetzt
2 weeks ago, by Reinhard Wobst
traceback und pdb in condfield2.py:99ff
2 weeks ago, by Reinhard Wobst
with debug commands, 17.2. debug_base
2 weeks ago, by Reinhard Wobst
base state, before type-problem
2 weeks ago, by Reinhard Wobst

- **`hg up(date) [-r revision]`**
  `hg checkout, hg co` **`# eingebaute Aliase`**

  Auschecken der benannten Revision (oder des **tip** = neueste Revision), nützliche Schalter **(show)**. **`hg revert:`** Nur zurück zur Revision, tip bleibt Eltern-Release.
- **`hg stat(us)`**

  Zeigt Zustand der Dateien im Arbeitsverzeichnis (Kontrolle, ob man Files vergessen hat ...)
- **`hg sum(mary)`**

  Information über aktuelle Revision und Arbeitsverzeichnis **(show)**

3.2. Befehle zum verteilten Arbeiten

- **`hg clone`**

  Repository clonen; typisch für Sandbox-Entwicklung:

```
$ pwd
/home/westerwelle/steuersenkung
$ mkdir ../sandbox
$ cd ../sandbox
$ hg clone ../steuersenkung
$ cd steuersenkung # !
```

  Bei lokalem Clonen nur **Hardlinks** von Repositorydaten, wenn möglich! Quelle wird in `.hg/hgrc` eingetragen. Es können auch einzelne Zweige/Tags/Releases ausgecheckt werden, vgl. `hg help clone`.

- **`hg pull` *`URL`***

  Update von lokalem oder entfernten Repository:

  `hg pull ssh://[user[:pass]@]host[:port]/[path]`

  (**`hg push -r rev`** = nur bis zur Revision *rev)*

  weitere Protokolle: **http, https**

  Danach immer "hg up"! (http ist schnell)

- **`hg push` *`URL`***

  Zurückspielen der Änderungen auf entferntes Repository

  (vorher **`hg commit`**!).

Es wird standardmäßig immer mit der gesamten History gearbeitet

(**`hg push -r rev`** = nur bis zur Revision *rev):*

"Given that having all history available in all repos is the premier advantage of distributed version control, there is little reason to selectively copy a part of the history."

Ausnahmen:

- einzelne Zweige pullen/pushen (s.o., vgl.a. 4.)
- siehe folgender Abschnitt (bundle)

3.3. Exportieren

- `hg archive`

  Umwandeln der aktuellen bzw. angegebenen Revision in ein Archiv (Filebaum, tar, tgz, tbz2, zip ...) - Exportmöglichkeit,

wenn Empfänger kein Mercurial hat oder nur per Mail/USB-Stick/CD verkehrt

- **hg bundle**
- **hg unbundle**

Bündelung von Changesets zu kompakten Archiven bzw. Einspielen

3.4. Befehlsüberblick

add	Fügt die angegebenen Dateien der nächsten Version hinzu: hg add -I '**.py'
addremove	Fügt alle neuen Dateien hinzu, löscht alle fehlenden Dateien: ... -I '**.py'
annotate	Zeigt Informationen über Änderungssätze pro Dateizeile an
archive	create an unversioned archive of a repository revision

```
backout  Macht einen vorangegangen Änderungssatzes rückgängig
bisect   Binäre Suche von Änderungssätzen
branch   Setzt oder zeigt den Namen des aktuellen Zweigs
branches Zeigt alle benannten Zweige des Projektarchiv an
bundle   Erzeugt eine Datei mit Änderungsgruppen
cat      output the current or given revision of files
clone    make a copy of an existing repository
commit   Übernimmt Änderungen der angegebenen Dateien oder
         alle ausstehenden Änderungen ins Archiv: hg commit -A
         -I '**.py' == addremove + commit
copy     Markiert Dateien als Kopien bereits versionierter
         Dateien
diff     Zeigt Änderungen des Projektarchiv oder angegebener
         Dateien an: hg diff | kompare -
export   dump the header and diffs for one or more changesets
forget   forget the specified files on the next commit
grep     Sucht ein Muster in angegebenen Dateien und
         Revisionen
heads    Zeigt die Köpfe des Archivs oder von
         Entwicklungszweigen
help     Zeigt die Hilfe für ein gegebenes Thema oder eine
```

```
          Hilfsübersicht
identify  Beschreibt die Arbeitskopie oder die angegebene
          Revision
import    Importiert eine Liste von Patches
incoming  Zeigt neue Revisionen in einer externen Quelle an
init      Erzeugt ein neues Projektarchiv im angegebenen
          Verzeichnis
locate    Suche Dateien mit bestimmtem Namen
log       Zeigt die Revisionshistorie des Archivs oder von
          Dateien an: hg log -v
manifest  Gibt das Manifest (komplette Dateiliste) der
          angegebenen oder aktuellen Revision aus.
merge     Führt das Arbeitsverzeichnis mit einer anderen
          Revision zusammen
outgoing  Zeigt Änderungssätze, die nicht im Zielarchiv sind
parents   Zeigt die Vorgänger des Arbeitsverzeichnisses oder
          einer Revision
paths     Zeigt Adresse für Aliasnamen von entfernten
          Projektarchiven an
pull      Holt Änderungen aus dem angegebenen Projektarchiv
push      Überträgt lokale Änderungen in das angegebene Ziel
```

```
recover  Setzt eine unterbrochene Transaktion zurück
remove   Entfernt die angegebenen Dateien in der nächsten
         Version
rename   Benennt Dateien um; äquivalent zu "copy" und "remove"
resolve  retry file merges from a merge or update
revert   Setzt gegebene Dateien oder Verzeichnisse auf frühere
         Version zurück: hg rev -r 0 name
rollback roll back the last transaction
root     Gibt die Wurzel (top) des aktuellen
         Arbeitsverzeichnisses aus
serve    Exportiert das Projektarchiv via HTTP
showconf Zeigt die kombinierten Konfigurationswerte aller
ig       hgrc-Dateien an
status   Zeigt geänderte Dateien im Arbeitsverzeichnis
summary  summarize working directory state
tag      Setze ein oder mehrere Etiketten für die aktuelle
         oder gegebene Revision
tags     Liste alle Etiketten des Archivs auf
tip      Zeigt die zuletzt übernommene Revision
unbundle Wendet eine oder mehrere Änderungsgruppendateien an
update   Aktualisiert das Arbeitsverzeichnis oder setzt es
```

```
          zurück: hg up [-r rev]
verify    Prüft die Integrität des Projektarchivs
version   Gibt Version und Copyright Information aus
```

Zusätzliche Hilfethemen:

config	Konfigurationsdateien	multirevs	Angabe Mehrerer Revisionen
dates	Datumsformate	diffs	Diff-Formate
patterns	Dateimuster	templating	Nutzung von Vorlagen
environment	Umgebungsvariablen	urls	URL-Pfade
revisions	Angabe Einzelner Revisionen	extensions	Benutzung erweiterter Funktionen

4. Tagging, Branchen, Mergen

4.1. Tags

Tags sind einfach Namen für Releases, die intern als extra Revision verwaltet werden:

```
hg tag "name for my release"
```

Im Logfile sieht das dann z.B. so aus:

```
o  Änderung:        16:7a5a76e884bf
|  Nutzer:         Reinhard Wobst <r.wobst@gmx.de>
|  Datum:          Thu Apr 01 14:49:34 2010 +0200
|  Zusammenfassung: Added tag Atmel1_final for changeset fe43113f54cc
```

Tags können an Stelle der Revision-ID angegeben werden (vgl. 5.).

4.2. Zweige (branches)

Details: vgl. 10. [3].

Es gibt genau genommen 5 Arten von Zweigen, die zwei wichtigsten zuerst:

4.2.1. *** Benannte Zweige

`hg branch _name_`

Der Name wird Bestandteil der History und bei pull/push mit übertragen (Unterschied zu Git!) und kann als Revisionsnummer verwendet werden. **Umschalten** zwischen Zweigen:

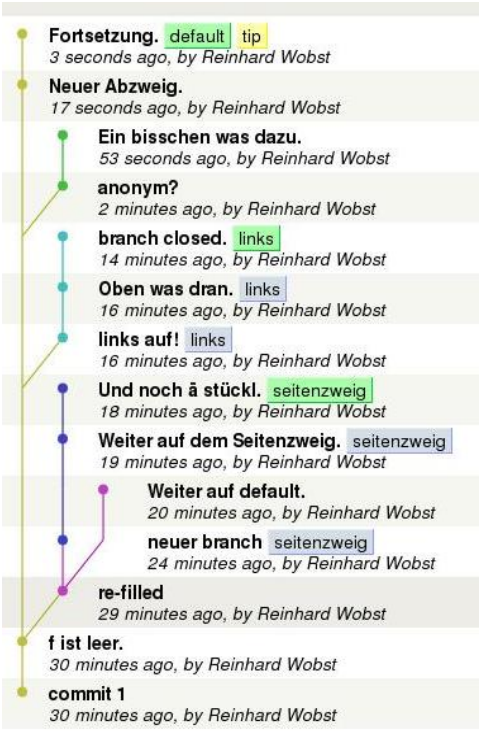
`hg up -r _name_`

Zweig abschließen:

```
hg commit --close-branch
```

Der Zweig wird danach von Kommandos nicht mehr aufgelistet (`hg heads`, `hg branches`) und im Log als "closed" markiert.

Sieht im Repository so aus:

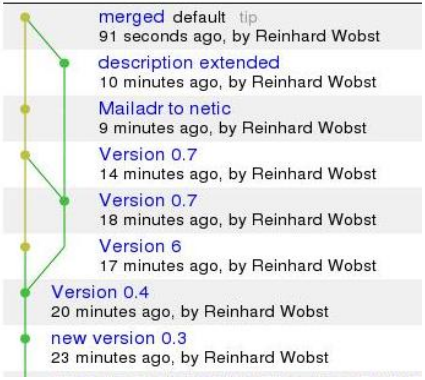


4.2.2. *** Anonyme Zweige

Entstehen als "Nebeneffekt" beim erzwungenen Update auf ältere Releases:

```
hg up -C -r 13
... hack, hack ...
hg com
```

Im Repository sieht das so aus:



Anonyme Zweige sind gut für kleine Seitenzweige (Bugfixes), die keinen extra Namen bekommen müssen.

4.2.3. Clone-Zweige

Rel. spitzfindige Definition - aber das lokale Clonen im Filesystem ist auch ein Verzweigen! Ungefährlich, nicht gut für große Repositorys.

4.2.4. Bookmarks

Extension von Mercurial (vgl. 10. , [4]): Objekte, die auf Releases zeigen, aber nicht Bestandteil dieser sind - entsprechen den Git-Zweigen und werden nicht exportiert. **Nachteil:** Wenn die

Bookmarks gelöscht sind,werden die Änderungen schleierhaft, falls nicht sauber dokumentiert!

4.2.5. Lokale Zweige

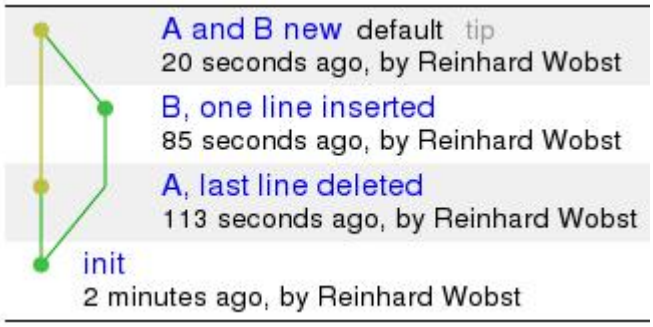
Weitere Extension von Mercurial (vgl. 10. , [5]), die nicht mit exportiert werden.

4.3. Merge

4.3.1. Standardsituation

```
$ hg pull ...
added 1 changeset with 1 changes to 1 files (+1 heads)
$ hg up
abort: crosses branches (use 'hg merge' ...)
$ hg merge [-r revision]
```

Das Arbeitsverzeichnis wird mit der angegebenen Version bzw. dem von zwei Köpfen, der nicht *parent* des Arbeitsverzeichnisses ist, zusammengeführt. Nach dem nächsten *commit* hat der tip zwei Eltern:



Das genügt in den meisten Fällen. -

4.3.2. Probleme

- Wenn zwei Entwickler am gleichen File arbeiten, aber an verschiedenen Zeilenbereichen, wird das File automatisch gemerged. Das kann heftige Kopfschmerzen verursachen! Automatisches Mergen kann mit `hg merge -P` unterdrückt werden. Wobei alle Änderungen vor dem folgenden `hg commit` noch rückgängig gemacht werden können.
- Bei Konflikten (Änderungen in gleichen Zeilenbereichen) wird das im Konfigurationsfile angegebene Programm gestartet, üblicherweise *kdiff3*. Ist kein Programm angegeben, werden Marker in das betroffene File eingearbeitet.

- Nicht aufgelöste Konflikte lassen sich mit `hg resolve` beseitigen (vgl. help).
- Bei größeren Änderungen würde ich Mercurial nicht vertrauen zu entscheiden, was "verschiedene Zeilenbereiche" sind.
- Prinzipielles Problem beim Mergen: Im Zweig A wird ein File entfernt, im Zweig B ein anderes. Beim Mergen fehlen beide Files. Oder: Im Zweig B ist das in A entfernte File noch vorhanden und wird gebraucht. Nach dem Merge funktioniert nichts mehr.

  Das Problem ist allgemein nicht lösbar - kann nur durch erzwungene Policy vermieden werden!

Bemerkung:

Das lässt ahnen, wie problematisch *rebase* oder gar *transplant* sein können (vgl. 7.).

5. Revisionsnamen, Zweignamen

Als Revisionsnummern können verschiedene Größen dienen, hier nochmals der Ausschnitt aus 4.1. :

o Änderung: 16:7a5a76e884bf
| Nutzer: Reinhard Wobst <r.wobst@gmx.de>
| Datum: Thu Apr 01 14:49:34 2010 +0200
| Zusammenfassung: Added tag Atmel1_final for changeset fe43113f54cc

- **Hashvalue:** *7a5a76* (so viele Stellen, bis er eindeutig wird, vgl. "man hg")
- **Integer:** *16* (ändert sich, wenn neue Zweige hinzukommen, wird nicht exportiert - vorher also mit "hg log", "hg glog", "hg serve" nachsehen!) - ist in der Regel am bequemsten

- **Tag:** *Atmel1_final* - muss also eindeutig sein
- **Zweigname:** *default* (falls der *tip* des Zweigs gemeint ist)
- **Bookmark-Name** (für Bookmark-Extension)

Es dürfen also Tagnamen und Zweignamen nicht kollidieren (und Bookmark-Namen auch nicht), und Hashwerte sind nur dann nicht mit Integern zu verwechseln, wenn sie wenigstens eine "echte" Hexziffer enthalten! (Alles sehr naheliegend, in der Praxis keine Probleme).

6. Konfiguration

Näheres dazu unter "man hgrc".

Konfigurationsfiles **`hgrc`** können auf verschiedenen Ebenen angegeben werden (UNIX, bei Windows vgl. Manpage):

```
/etc/mercurial/hgrc
/etc/mercurial/hgrc.d/*.rc
$HOME/.hgrc
$PWD/.hg/hgrc
```

Jeder der Files überschreibt die Einstellungen der vorigen (diese lassen sich aber nur für Extensions disablen!). Einige wichtige Bestandteile: persönliche Daten, Aliase, Extensions, Hooks.

Beispiel:

```
[ui]
username = Reinhard Wobst <r.wobst@gmx.de>

[alias]
chg = stat -m -a -r -d -C

[extensions]
hgext.graphlog =
pager =

[pager]
pager = less
attend = help, status, log, glog, diff, manifest,
    grep
```

```
[hooks]
pre-commit = $HOME/util/PD/Hg/hooks/Trailwhite.py
```

7. Gimmicks

- **GUI:**
  - Zum Betrachten der History und diff's reicht schon der lokale Webserver (`hg serve`, s.o.), Stil (Mercurial/Git) umschaltbar
  - **TortoiseHg** ist vor allem für Windows gut (gibt es ebenso für Linux)
  - weitere 15 Vorschläge nebst IDE/Editor-Anbindung, Trac-Unterstützung u.a. auf 10. , [7].
- **Hooks** werden über `hgrc` eingebunden (s.o.) und sind üblicherweise normale Skripte, können aber auch Python-Plugins mit Zugriff auf Internas sein. Lösen Probleme wie Prüfung auf

vorgegebenen Code Style vor dem Commit, Zeilenendungen, File-Locks u.a.m.

- **hg bisect:** Unterstützung der Bisektionsmethode bei der Suche in History, wo ein Fehler erstmals auftrat
- **Löschen von Changesets:** Kommt von Git, entspricht eigentlich nicht der Mercurial-Philosophie, wird aber durch **MQ-Extension** mit `mq strip` möglich
- **Behandlung von Binärfiles:**
  - Können Repository aufblähen (praxisches Beispiel: Satelliten- und Röntgenbilder), obwohl read-only.

Trick: zentrales Verzeichnis, nur symbolische Links committen! (Geht auch für Subversion.) Unter Windows dann immer noch Linkadresse sichtbar.

- Kleine Änderungen wie z.B. bei *doc* sind nicht so kritisch, da Mercurial mit bdiff arbeitet (vgl. 8.). Schwierig bei komprimierten Files (OpenOffice, *docx).* Grob getestete Idee hierzu: Per pre-commit Hook Archiv in *tar* verwandeln und dieses committen. Post-update-Hook verwandelt dann das *tar*-File zurück in die ursprüngliche Form. Nebeneffekte noch nicht praktisch untersucht.

- **rebase extension:** (vgl. 10. , [6]) Von Git übernommen; manchmal nützlich, aber gefährlich. Bester Link zum Thema: http://blog.experimentalworks.net/2009/03/merge-vs-rebase-a-deep-dive-into-the-mysteries-of-revision-control/

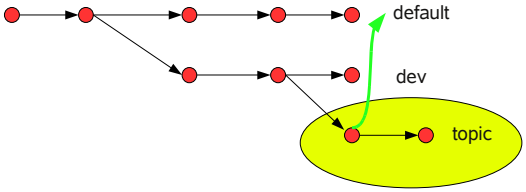
  Quintessenz:

  - Prinzipielle Schwierigkeiten mit "merge" werden nicht durch rebase gelöst! (merge ist schon problematisch genug)
  - rebase nur für lokalen Branch anwenden, niemals für pushed oder pulled branches! Wird oft missverstanden.

- vgl.a.: http://changelog.complete.org/archives/586-rebase-considered-harmful

  (negative Erfahrungen werden nicht weitergegeben)

- Es gibt sogar eine **transplant** extension, mit der man Zweige verschieben kann (noch gefährlicher!):



```
$ hg clone http://example.com/upstream/repo
$ hg update default
$ hg transplant --branch topic
```

- Die **ConvertExtension** (vgl. 9.) erlaubt, Repositorys von Unterverzeichnissen zu bilden.

8. Background

Mercurial speichert normalerweise Differenzen von Dateien, auch von binären, falls das einen Speichervorteil bringt *(bdiff)* (Git komprimiert nur). Um robuster und schneller zu sein, werden in Abständen die Vollversionen hinterlegt. Das interne Format des Repositorys hat sich seit Anfang nur wenig verändert!
Dieses "**revlog**"-System ist auf Speicher und Suchzeit optimiert und wird ausführlich in 10. , [8] begründet.

9. Zusammenarbeit mit anderen VCS

Überblick: http://mercurial.selenic.com/wiki/RepositoryConversion?action=show&redirect=ConvertingRepositories

Konvertieren von Repositorys von/nach: Arch, TLA, Baz 1.5, Bazaar, ClearCase, RCS, CVS, Darcs, Git, Subversion, Perforce, Bitkeeper, TeamWare, Visual SourceSafe, Monotone.

Wichtigster Link = **ConvertExtension:**

http://mercurial.selenic.com/wiki/ConvertExtension

Eingebaut, also mitgeliefert, muss nur noch aktiviert werden.

9.1. Mit Subversion

http://mercurial.selenic.com/wiki/WorkingWithSubversion

Erlaubt Ein- und Auschecken, Konvertieren etc.; mehrere Methoden mit Pros und Kontras werden im Artikel diskutiert.

Ausführliche Anleitung: http://ww2.samhart.com/node/49
("Converting from Subversion to Mercurial")

9.2. Mit CVS

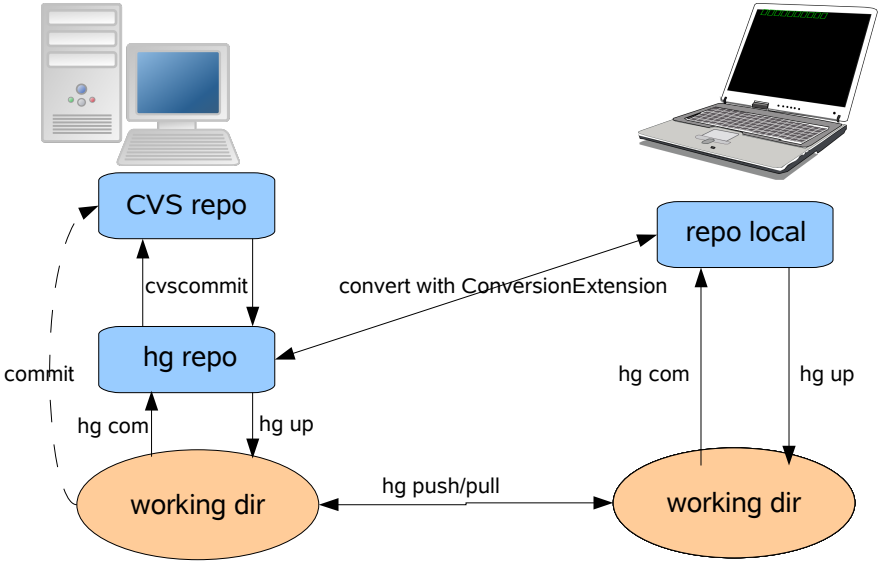
Unterschiedliche Konzepte von CVS und Hg: Changesets in Hg umfassen i.a. viele Files, in CVS oft nur einen; Tags und Branches haben nur gleiche Namen, aber verschiedene Bedeutung.

Lösungen und Probleme für Routinearbeit werden in https://wiki.mozilla.org/Using_Mercurial_locally_with_CVS diskutiert.

9.3. Mit Git

Konvertierung ist mit obiger Extension möglich, direktes Arbeiten mit Git-Repositories: http://mercurial.selenic.com/wiki/HgGit (gute Anleitung). Git-Branches werden dabei in Hg-Bookmarks übersetzt (vgl. 4.2.4.). Kann auch genutzt werden, ohne dass Git installiert ist.

9.4. Parallele Nutzung mehrerer Systeme



- Mercurial kann immer auch privat genutzt werden, ohne bisherigen Workflow zu stören:
- Manchmal Parallelbetrieb möglich (meist vermutlich Hg-SVN)
- Umstellung erfordert
  - Umschreiben von Hooks
  - Ändern von Konzepten
  - Lösen von auftretenden Problemen
  - Umschulung

Man wird also erst dann umstellen, wenn die Schmerzen groß genug sind.

10. Literatur

[1] **Homepage:** http://www.selenic.com/mercurial/wiki/

[2] **Hauptwerk:** Bryan O'Sullivan; Mercurial - The Definitive Guide (O'Reilly?), online unter http://hgbook.red-bean.com/read/

[3] Vermutlich beste Einführung in **Branching:** http://stevelosh.com/blog/2009/08/a-guide-to-branching-in-mercurial.html

[4] http://mercurial.selenic.com/wiki/BookmarksExtension

[5] http://mercurial.selenic.com/wiki/LocalbranchExtension

[6] http://mercurial.selenic.com/wiki/RebaseExtension

[7] http://mercurial.selenic.com/wiki/OtherTools

[8] Matt Mackall, Towards a better SCM: Revlog and Mercurial (Hintergründe gut erklärt):
http://mercurial.selenic.com/wiki/Presentations?action=AttachFile&do=view&target=ols-mercurial-paper.pdf