



Verteilte Versionskontrolle mit Mercurial



Moderator



Dirk Bielemeier
Binaris Informatik GmbH



DVCS ?

Geschichte, Vorteile,
Abgrenzung, Vergleich

Praxis 1

Setup, Versionierung eines
kleinen Projektes, Basisbefehle

Technik

Hinter den Kulissen
von Mercurial

Praxis 2

Workflows, Branch- und
Releasemanagement und mehr

Fazit

Was uns der Moderator
sagen wollte 😊

Noch Zeit ?...

Mercurial Hooks und Queues



Einführung in Distributed Version Control (DVCS)

Kurze Geschichte des DVCS

Vorteile eines DVCS

**Abgrenzung zu zentralisierten Repositories
wie CVS oder Subversion**

Vergleich der bekanntesten DVCS



Versionskontrolle ist der Prozess der Verwaltung multipler Versionen von Informationseinheiten

Historie:

~ 1980

SCCS (Source Code Control System), später als freie Version
RCS (Revision Control System)

Zentralisiertes Repository, File-Locking-Prinzip

~ 1990

CVS (Concurrent Version System)

Zentralisiert, Prinzip des Mergings wird eingeführt (kein File-Locking mehr)

2001

Subversion

Zentralisiert, besseres ggü. CVS besseres Namespace-Management,
Verwaltung binärer Daten etc.

Ab 2003

DVCS

Bitkeeper (2003), Bazaar (2004), Mercurial (2005), Git (2006)



Vorteile eines DVCS

Branching ist einfach,
Merging ist (relativ) einfach

Serverausfall
- Na und?

Nimm doch das Repo
von jemand anderem

Einfacher
Austausch
(Sharing)

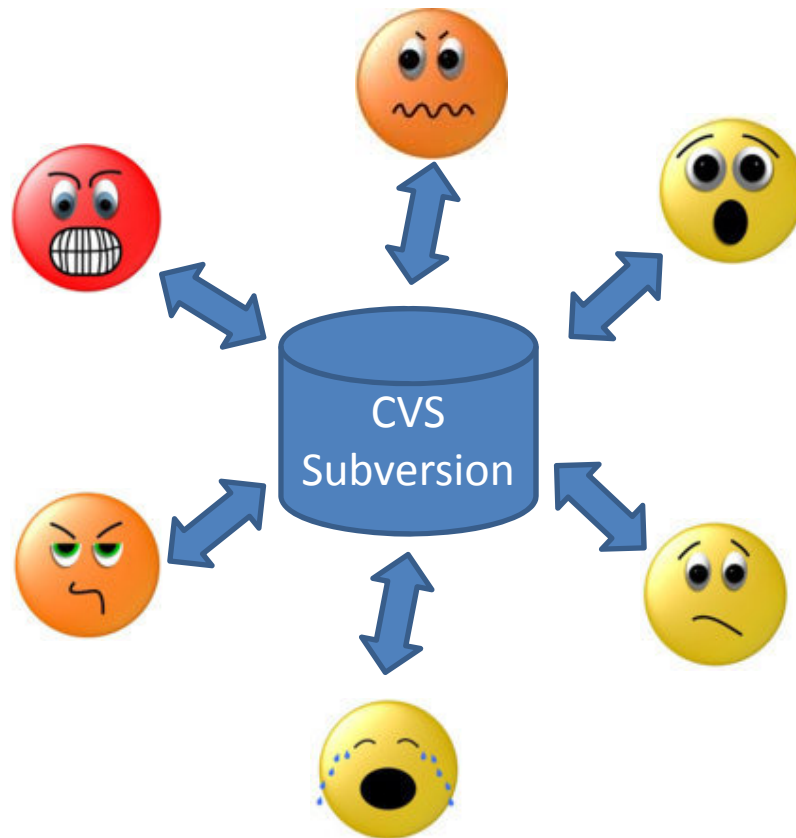
Liberté
Egalité
Fraternité

Lokale
Repositories

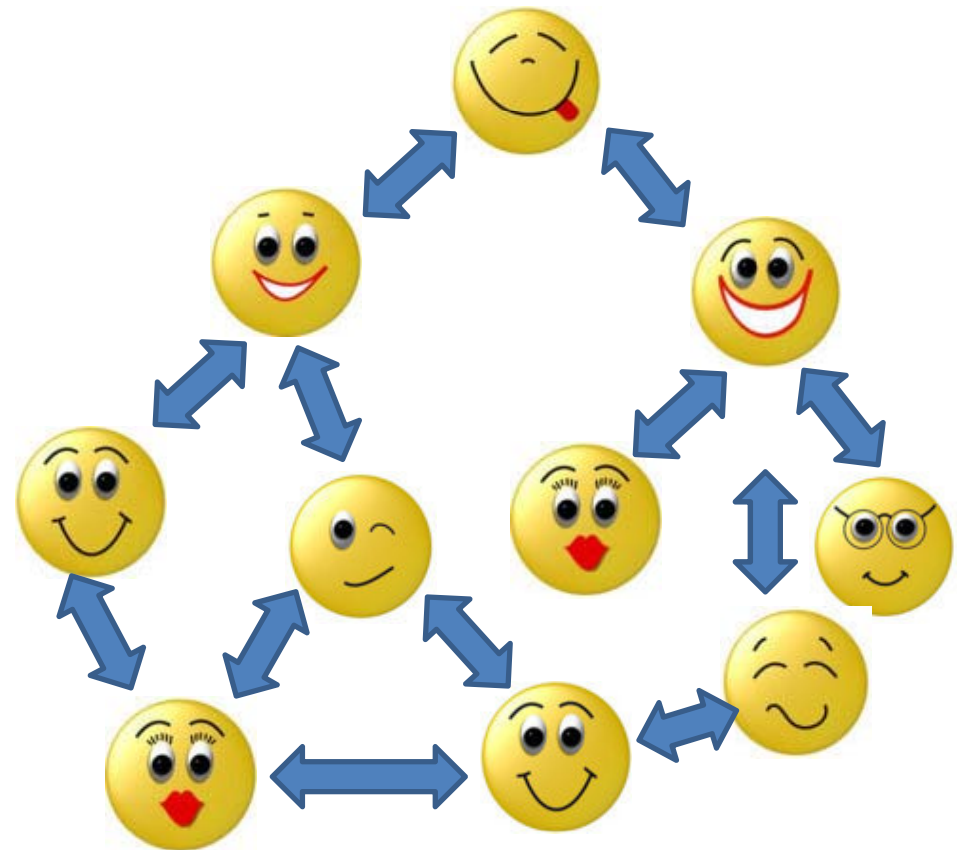
- Historie immer dabei
- Einfacher Versionswechsel
- Lokales Branching



Zentral?



Verteilt!



(Und das ist nur eine von vielen Möglichkeiten!)



DVCS im Vergleich

Seit 2004

Ubuntu,
MySQL ...

- Portabilität (Windows, Unix-Derivate, Mac OS)
- Plugins (in Python)
- Unterstützt viele Workflows
- Migration von CVS / Subversion sehr langsam



Bazaar

Seit 2006

Linux,
Google Android,
OLPC ...

- Linux-Abhängigkeit, benötigt Cygwin unter Windows
- Starkes Client Storage Konzept
- Unterstützt N-Way-Merge
- Rebase und Änderung der Committhistorie möglich
- Komplexer, etwa 150 Befehle im Sprachumfang
- Migration von CVS / Subversion aufwändig

+++ git

Seit 2005

Mozilla,
Open Solaris,
OpenJDK ...

- Portabilität (Windows, Unix-Derivate, Mac OS)
- Einfach zu lernen, exzellente Dokumentation
- Historie ist heilig
- Plugins (in Python)
- Toolgestützte Migration von CVS / Subversion





Praxis 1

Mercurial Setup

Wir versionieren ein kleines Projekt

Pulling, Pushing + Basic Merging

Add, Remove, Copy, Rename, Diff's



Installation

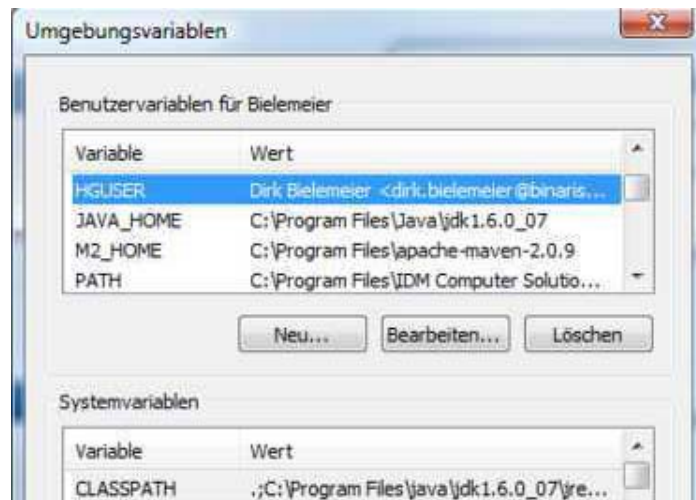
Windows: Beste Version gibt es im Bundle mit TortoiseHG <http://bitbucket.org/tortoisehg/stable/wiki/Home>

Linux: Über Package-Manager, OpenSUSE etwa "zypper install mercurial"

Solaris: Packages für Mercurial über <http://sunfreeware.com>

Mac OS X: Installer unter <http://mercurial.berkwood.com>

Post-Installation-Task (etwa Windows)



Fertig!

```
c:\hg>hg version
Mercurial Distributed SCM (version 1.3)

Copyright (C) 2005-2009 Matt Mackall <mpm@selenic.com> and others
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

c:\hg>hg help init
hg init [-e CMD] [--remotecmd CMD] [DEST]

create a new repository in the given directory

    Initialize a new repository in the given directory. If the given
    directory does not exist, it will be created.

    If no directory is given, the current directory is used.

    It is possible to specify an ssh:// URL as the destination.
    See 'hg help urls' for more information.

options:
    -e --ssh          specify ssh command to use
    --remotecmd       specify hg command to run on the remote side

use "hg -v help init" to show global options

c:\hg>
```

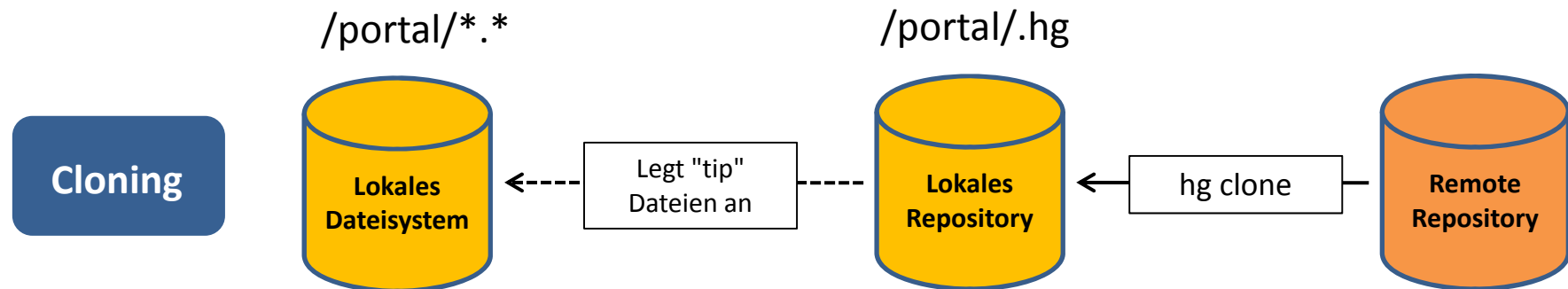
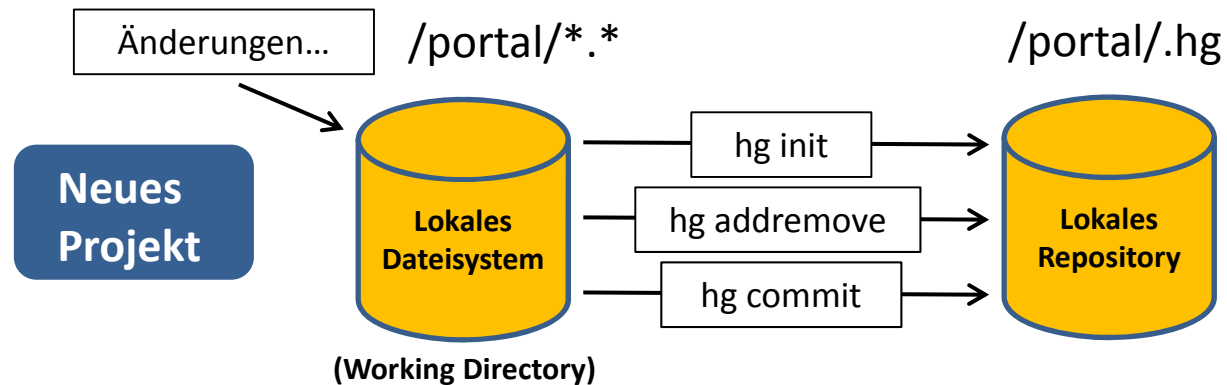


Leichter Einstieg für Subversion-Kenner

Subversion	Mercurial	Note
svn add	hg add	
svn checkout	hg clone	
svn commit	hg commit; hg push	hg push publishes after commit
svn delete (svn remove)	hg remove	
svn diff	hg diff	
svn help	hg help	
svn import	hg addremove; hg commit	
svn info	hg parents	Shows what revision is checked out
svn merge	hg merge	
svn status	hg status	
svn update	hg pull -u	



Projekt anlegen

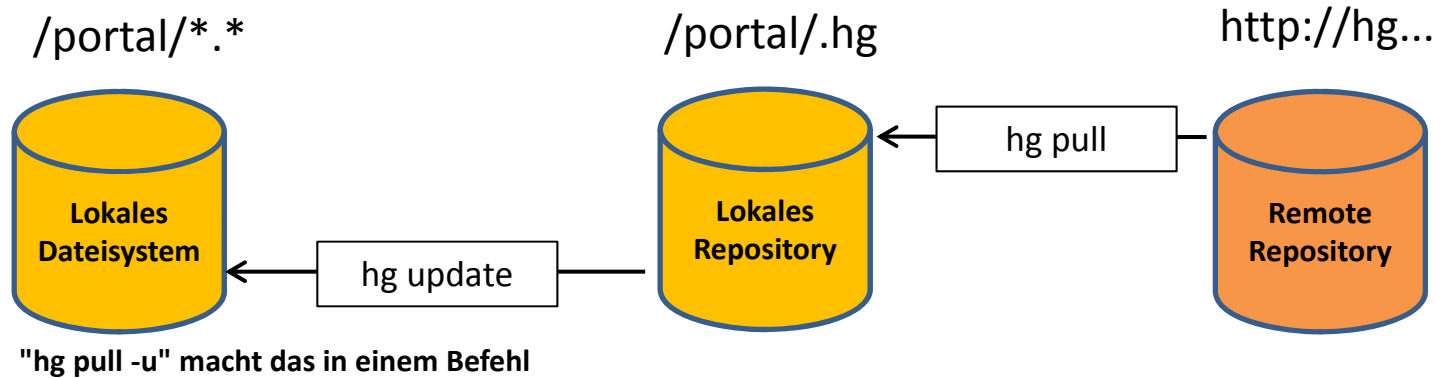


```
hg clone http://hg.serpentine.com/tutorial/hello
hg clone http://bitbucket.org/tortoisehg/stable/
hg clone http://bitbucket.org/lindenlab/certified-http-trunk/
```

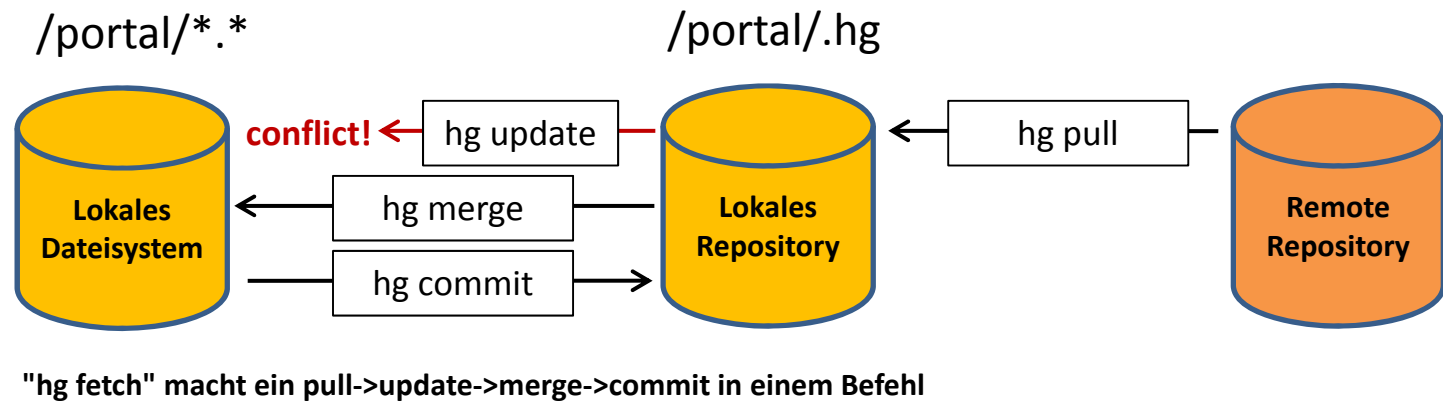


Pulling

Ohne
Konflikte

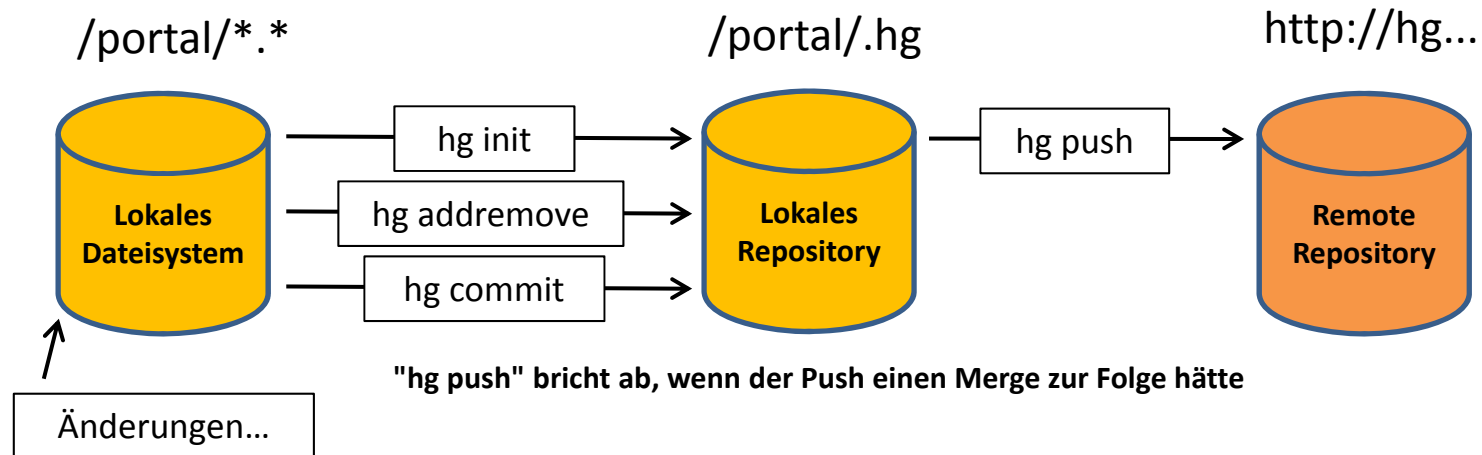


Mit
Konflikten



Dry Run: "hg incoming"

Pushing



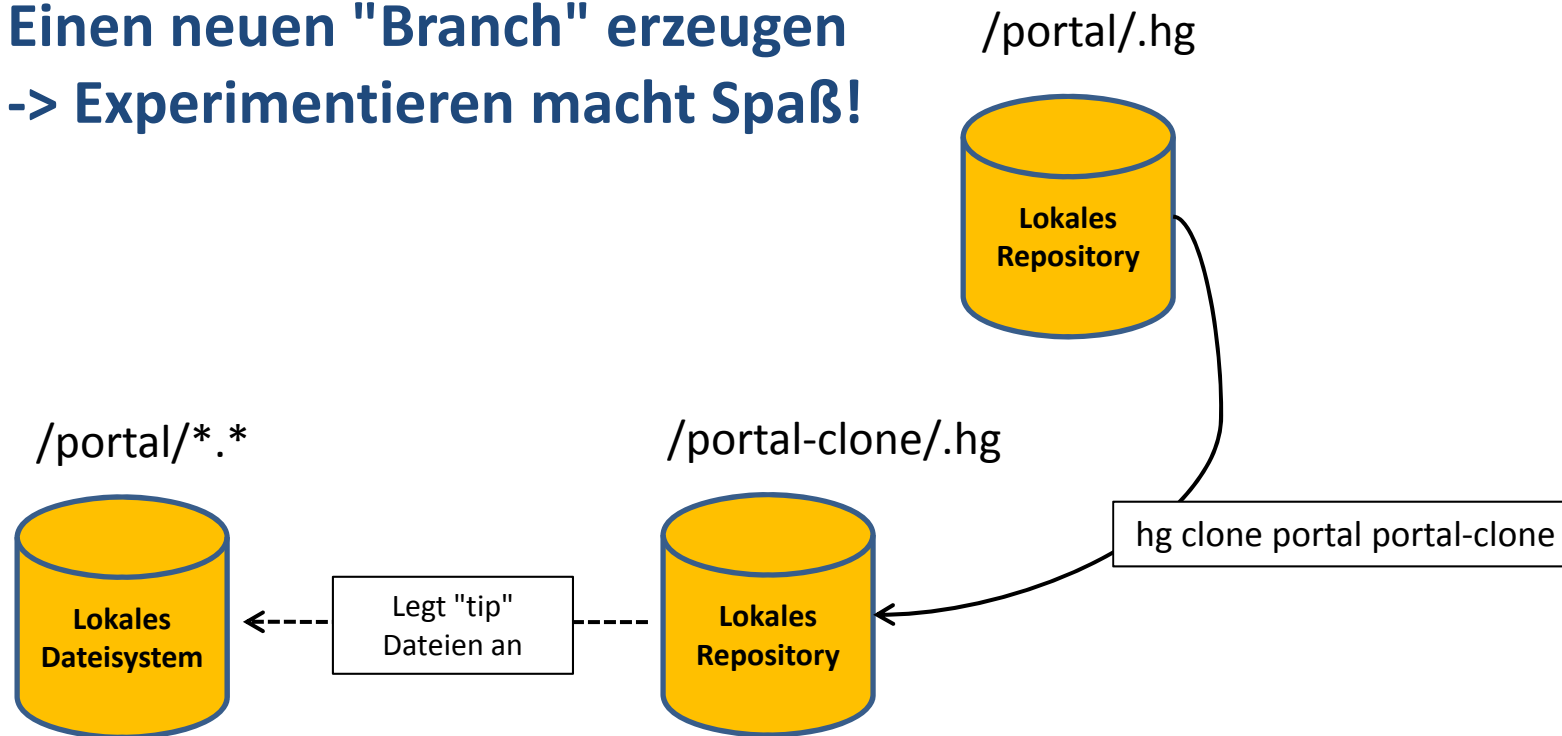
Dry Run: "hg outgoing"



Beim push steht die Option -u nicht zur Verfügung. Man darf also kein fremdes Repository auf eine andere Version setzen.

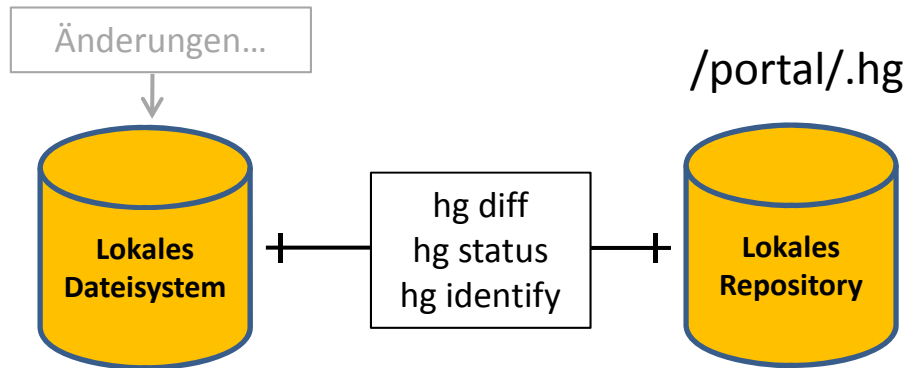


Einen neuen "Branch" erzeugen -> Experimentieren macht Spaß!

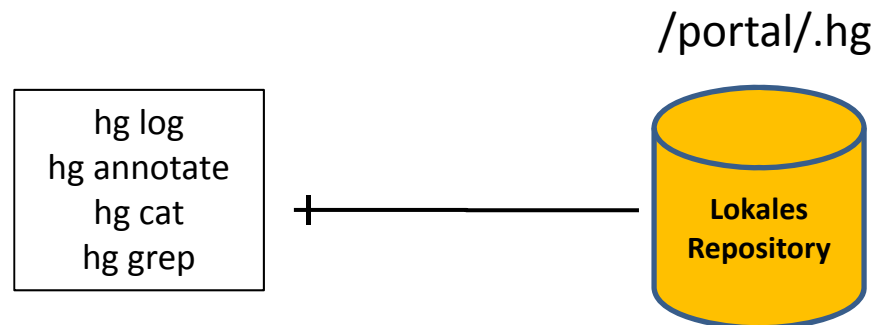




Dateisystem mit Repo vergleichen



Lokales Repository abfragen





Probleme ?

Änderungen zurücknehmen

`hg revert [OPTION] [FILE]`

Konflikte lösen

`hg resolve [OPTION] [FILE]`

Parameter HGMERGE ggf. setzen

```
c:\hg\conflict>hg merge  
abort: outstanding uncommitted merges  
  
c:\hg\conflict>hg commit -m "Try to commit"  
abort: unresolved merge conflicts (see hg resolve)
```

1

3

```
c:\hg\conflict>hg resolve myfile.txt -m
```

```
c:\hg\conflict>hg resolve -l  
U myfile.txt
```

2

```
c:\hg\conflict>hg resolve -l  
R myfile.txt  
  
c:\hg\conflict>hg commit -m "Try to commit"
```

4



Technik

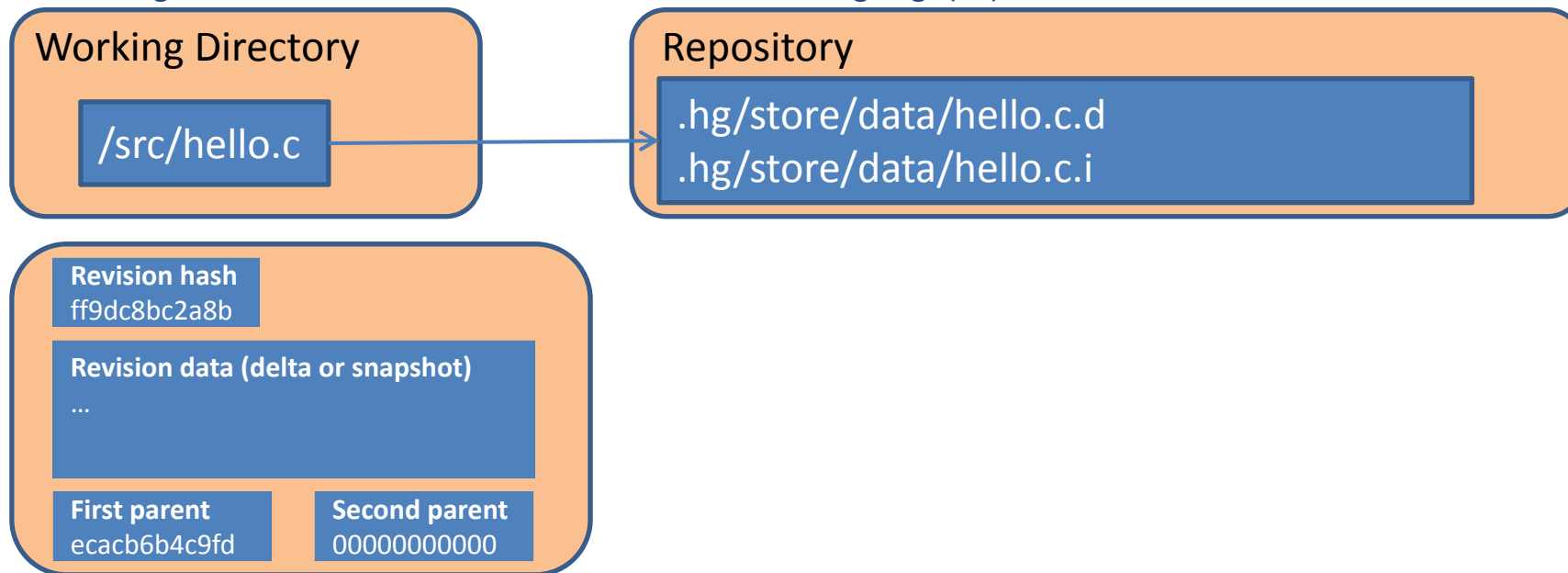
Mercurial's Historical Record
Effizientes Speichern
Schnelle Wiederherstellung
u.a.



Mercurial's Historical Recordings

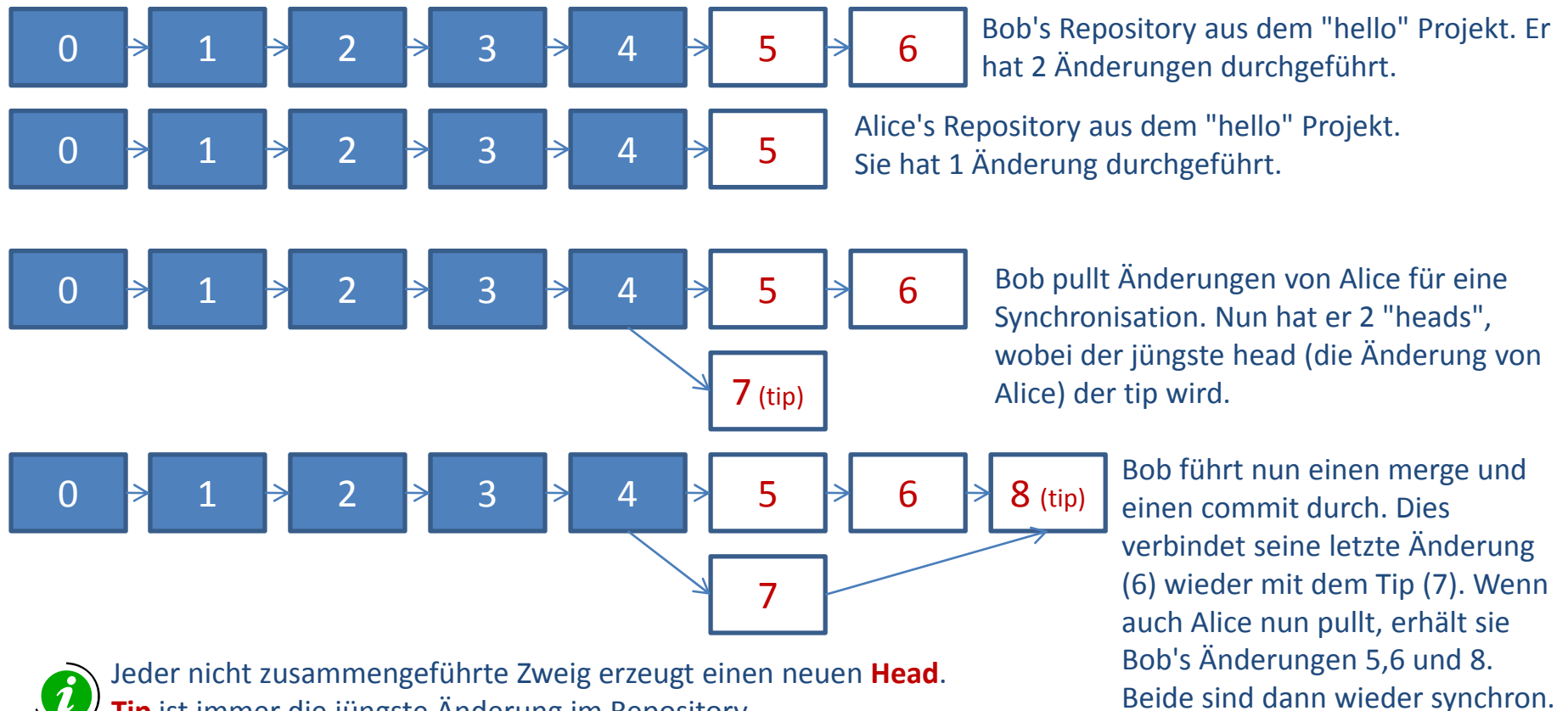
Mercurial kennt 3 Artefakte, die zusammen für die Organisation des Repositories verantwortlich sind:

- Revlog: Der Revision-Log speichert die Daten in Datenfiles mit der Endung *.d im Verzeichnis .hg/store/data. Bei großen Dateien wird auch noch eine Indexdatei angelegt (*.i).



- Manifest: Speichert Informationen über die Dateien, die im Repository liegen. Enthält Informationen darüber, welche Dateien ein Changeset ausmachen.
- Changelog: Hier liegen die Informationen über jedes Changeset, etwa User, Committext und die Revision des Manifests, das zum Changelog gehört.

Tip, Head und Parent



Jeder nicht zusammengeführte Zweig erzeugt einen neuen **Head**.

Tip ist immer die jüngste Änderung im Repository.

Parents sind die Eltern der aktuellen Revision des Working Directories.

Eine Revision kann maximal 2 Parents haben, da Mercurial nur den Merge von max. 2 Zweigen auf einmal zulässt.



Technisches Design

Effizientes Speichern

Inkmente und/oder
Snapshots. Binär
oder Text

Atomares Speichern

Alle oder Keins.
Read Consistency.

Schnelle Wiederherstellung

Intelligente Kompression

Extra Netzwerk-
komprimierung

Hohe Integrität

Durch Hashes und
die schnelle Wieder-
herstellung

Konkurrierender Zugriff problemlos



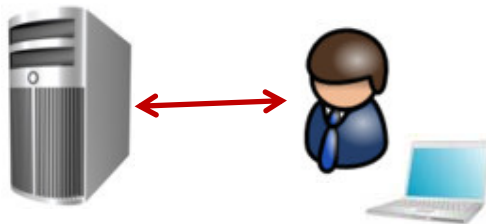
Praxis 2

**Workflow Modelle
Release und Branchmanagement
Technischer Austausch
Fehlersuche und -behebung
Mercurial GUI's**

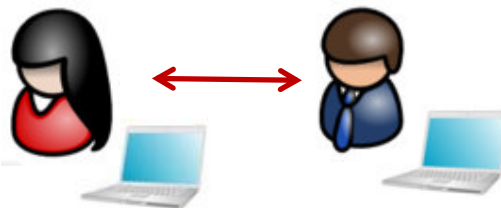


Workflow Modelle

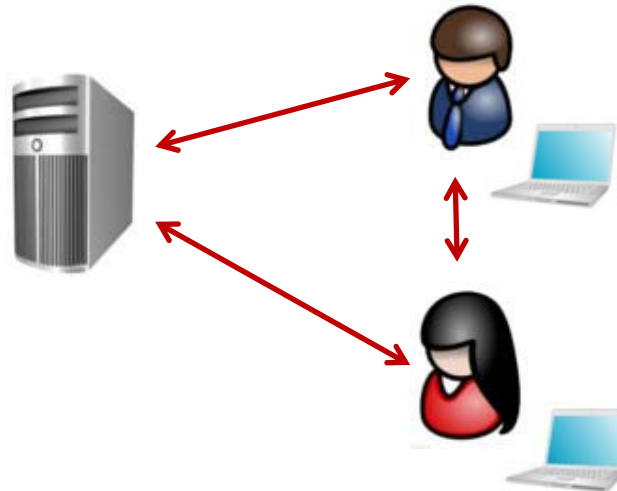
Eine Person



Partner



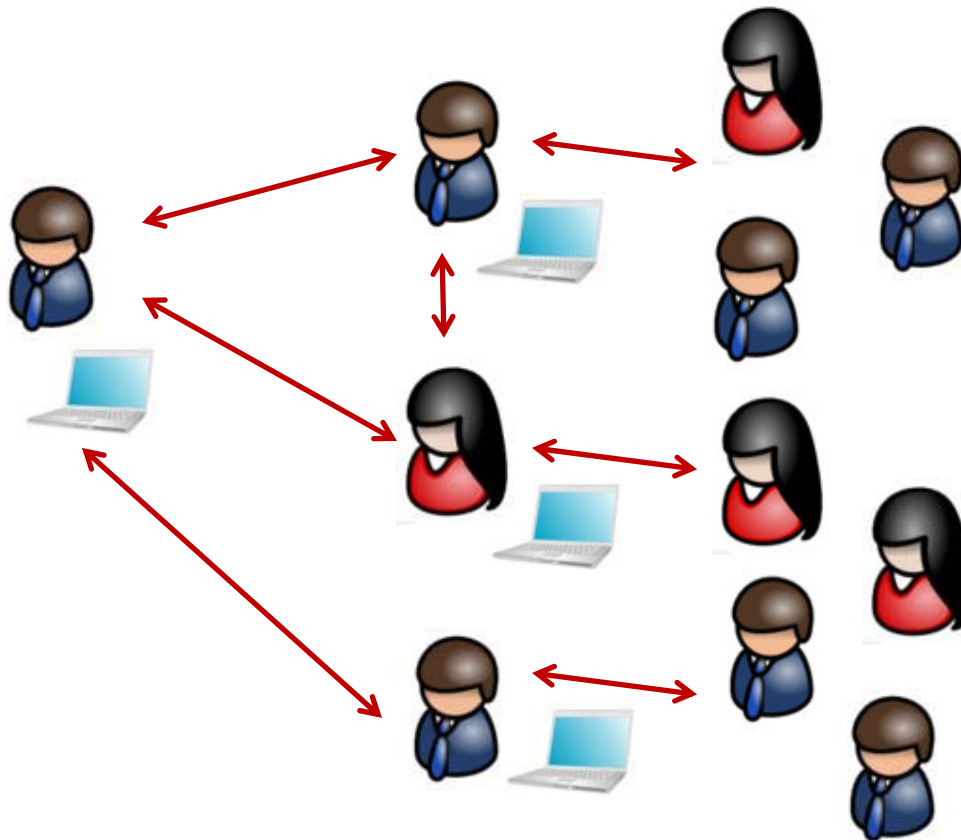
Shared Push: Zentralisiert, Push und Pull für alle erlaubt, lokale Branches, Austausch untereinander





Workflow Modelle (2)

Linux Kernel Model: Cloning immer möglich, hierarchisch
höhere Ebene pullen Änderungen von "Vertrauten"



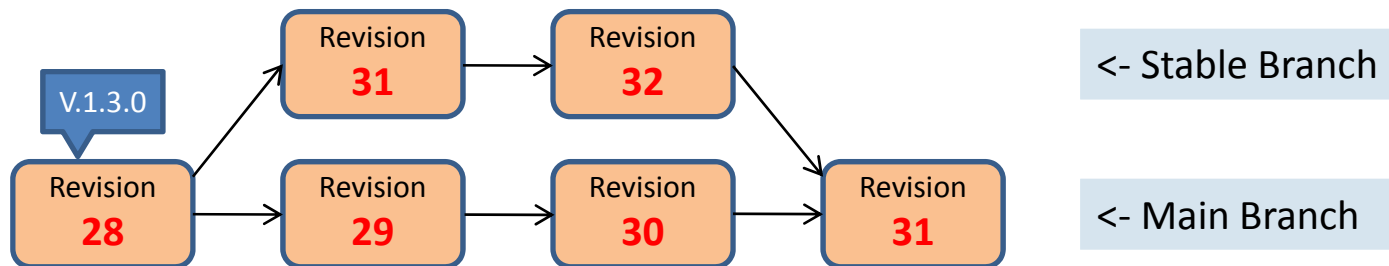
Kernel Report August 2009:

- 2,7 Mio. Codezeilen in den letzten 16 Monaten
- 5.000 Developer von 500 Firmen
- Top: Red Hat 12,9 % aller Changes, IBM 6,3 % der Changes
- u.a. Volkswagen (Automotive Net), Quantum Control (Yachtnavigation)

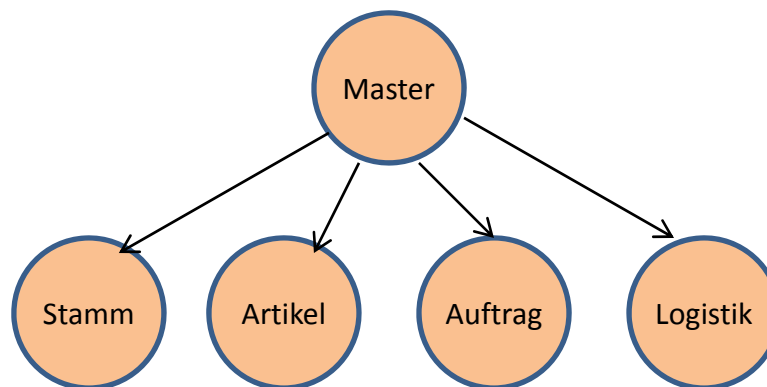


Release und Branchmanagement

Stable and Main Branch



Feature Branch





Sharing (technical)

hg serve

Ad-Hoc-Austausch, interner HTTP-Server

ssh

Etwa OpenSSH unter Unix,
unter Windows wird TortoiseHG
mit plink ausgeliefert.

HTTP mit CGI

Ein oder mehrere Repos sind über einem CGI-konfigurierten Webserver erreichbar.

Webinterface

Features: Browsing, Log,
Verschiedene
Downloadformate, RSS Feeds



Mercurial

directory / @ 9352:37042e8b3b34

log
graph
tags
branches
changeset
browse
bz2
zip
gz

name	size	permissions
[up]	0	drwxr-xr-x
constric/	0	drwxr-xr-x
doc/	0	drwxr-xr-x
hpxet/	0	drwxr-xr-x
liba/	0	drwxr-xr-x
mercurial/	0	drwxr-xr-x
templates/	0	drwxr-xr-x
tests/	0	drwxr-xr-x
.hgignore	545	-rw-r--r--
.hgslgc	2168	-rw-r--r--
.hgtags	1281	-rw-r--r--
CONTRIBUTORS	1443	-rw-r--r--
COPYING	17982	-rw-r--r--
Makefile	1295	-rw-r--r--
README	306	-rw-r--r--
hg	787	-rw-r-xr-x
hgeditor	1201	-rw-r-xr-x
hgws.cgi	306	-rw-r--r--
hgwebdir.cgi	2220	-rw-r-xr-x
setup.py	9113	-rw-r--r--



Fehlersuche und -behebung

hg update

- Überführt das Working Directory auf die angegebene Revision
- Ohne Angabe der Revision wird auf den tip des aktuellen Branch aktualisiert

hg rollback

- Rollet genau eine Transaktion zurück
- Vorsicht in "Public Repositories"
- Der Push kann nur lokal zurückgerollt werden

hg revert

- Rollet Dateiänderungen zurück
- Funktioniert etwa bei add, remove, copy und rename
- Kann auch ganze Verzeichnisse zurückrollen
- Auch Angabe eines Datums oder einer Revision möglich



Fehlersuche und -behebung (2)

hg backout

- "Undo" - Funktion für ganze Changesets
- Löscht keine Historie, sondern macht genau die Änderungen, die notwendig sind, um die Änderungen eines speziellen Changesets rückgängig zu machen
- Die Änderungen werden in einem Merge zusammengefasst

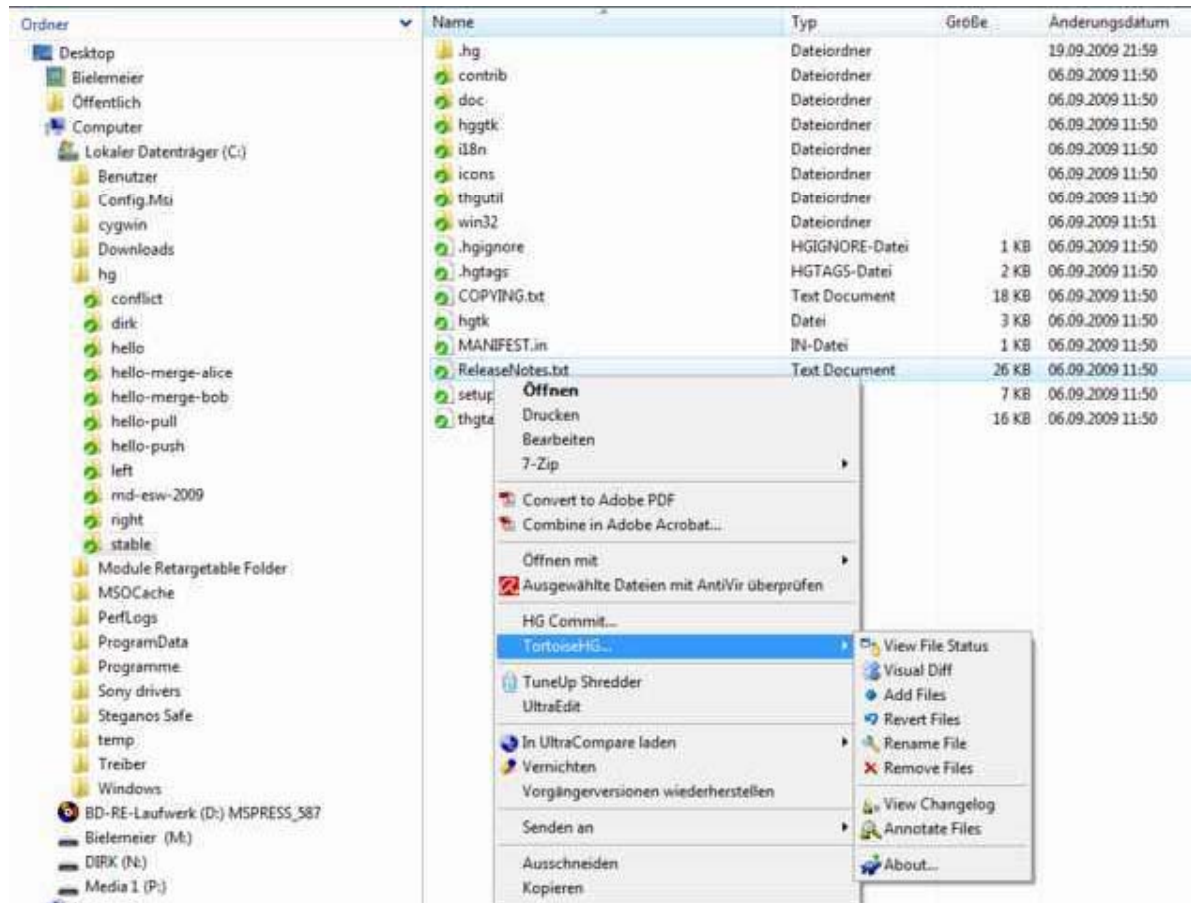
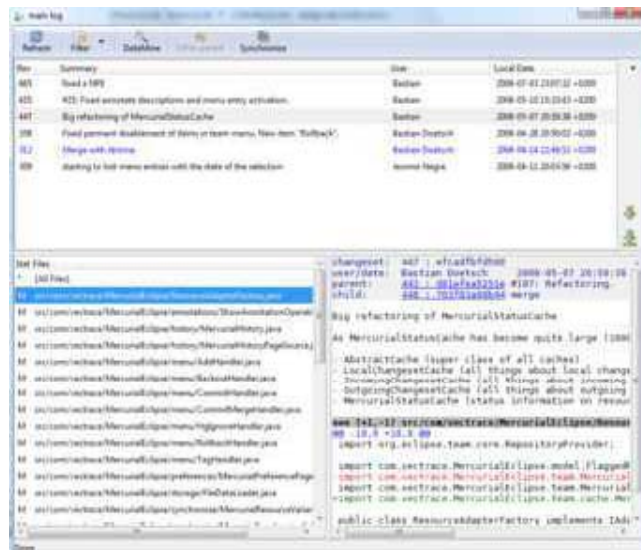
hg bisect

- Testet Revisionen auf einen Bug oder andere Probleme
- Lässt je Revision einen Test (Skript) laufen, dass die Revision prüft. Kann daher beliebige Tests durchführen.



Tortoise HG

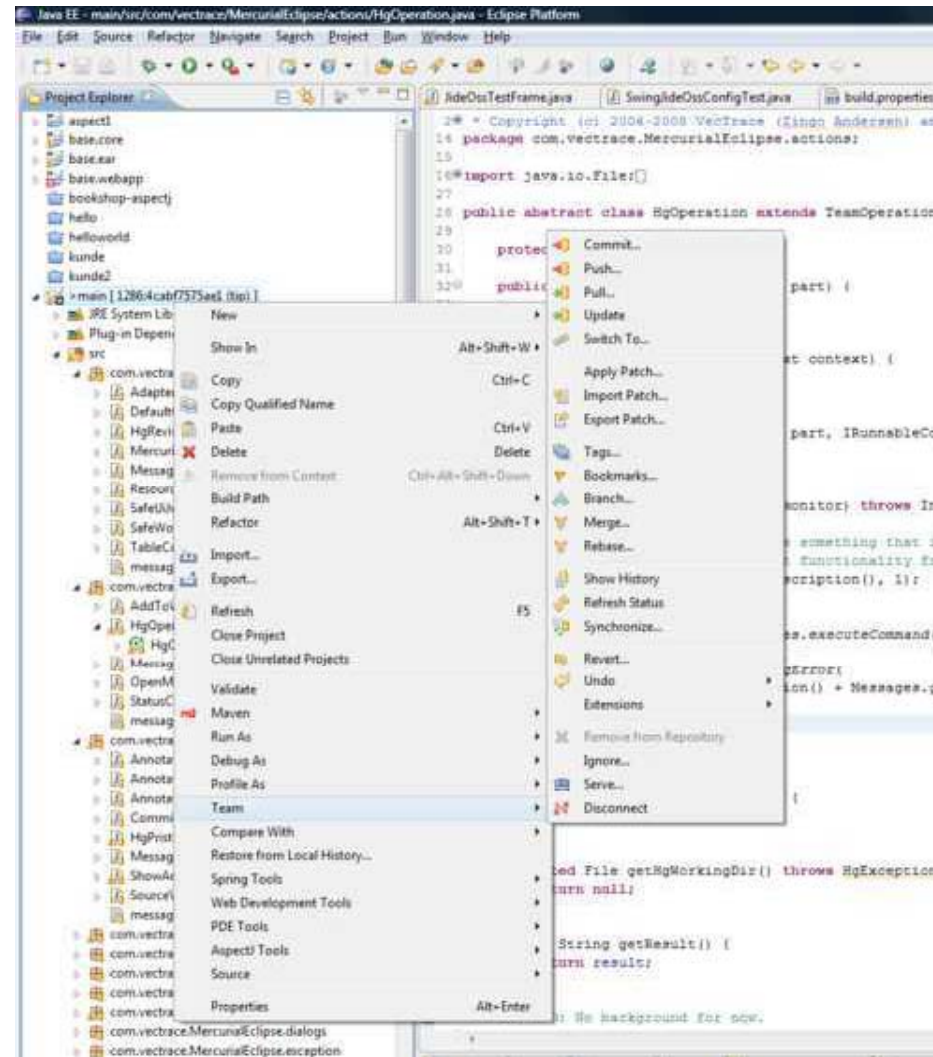
- Grafisches Tool, eingebettet in den Windows Explorer
- Nutzt Overlay-Icons und das Kontext-Menü des Explorers
- Verfügbar für Windows, Linux und Mac OS X





Eclipse Mercurial

- Plugin für Eclipse von Vectrace
- Projekt anlegen durch Cloning eines Repo
- Unterstützung von Rebase und Bookmarks
- Shelve / Unshelve Support zum Wegpacken kurzfristig nicht benötigter Änderungen





Fazit - Go!

Umsteigen - Ihr bereut es nicht - Versprochen!
**Mercurial hat sehr gute Migrationstools für
Subversion und CVS**
**Sogar das Lesen und Schreiben in ein
Subversion-Repository ist mit Mercurial
möglich**



Fortgeschrittene Themen

Hooks (Trigger)
Mercurial Queues



Hooks

Hooks (Trigger) werden ausgelöst, wenn bestimmte Repository Events erfolgen

Es gibt verschiedene Events, an die Hooks angehängt werden können.
Beispiele sind:

- **precommit**: Bevor ein Commit ausgelöst wird.
- **commit**: Nachdem ein neues Changeset erzeugt wurde.
- **incoming**: Wenn ein neues Changeset von aussen in das Repository kommt.
- **outgoing**: Wenn ein Changeset von aktuellen Repository aus übertragen wird.
- **preupdate**: Läuft vor einem Update oder Merge des Repositories.
- **pretag**: Bevor ein neuer Tag angelegt wird.

Ein Hook kann ein Shell-Skript oder eine Python Funktion sein.

Beispiel: Ein PreTxnCommit-Hook zum Prüfen der Länge des Committextes

```
cat .hg/hgrc
```

```
[hooks]
```

```
pretxncommit.msglen = test `hg tip --template {desc} | wc -c` -ge 10
```



Mercurial Queues

Ergänzt Revisionsmanagement um das Patch Management

Was ist ein Patch?

```
$ echo 'this is my original thought' > oldfile
$ echo 'i have changed my mind' > newfile
$ diff -u oldfile newfile > tiny.patch
$ cat tiny.patch

--- oldfile 2009-05-05 06:44:39.554480179 +0000
+++ newfile 2009-05-05 06:44:39.554480179 +0000
@@ -1 +1 @@
-this is my original thought
+i have changed my mind

$ patch < tiny.patch
patching file oldfile
$ cat oldfile
i have changed my mind
```



Mercurial Queues (2)

Patches bieten große Vorteile:

- Hohe Agilität durch flexible Hereinnahme eines Patches oder dessen Entfernung aus Source Code
- Erhöhung der Transparenz etwa bei der Bugsuche. Z.B. zeigt `hg log <Filename>` alle Changesets + die Patches an, die das File änderten

Danke schön!

Zeit für Fragen ☺



Home: <http://mercurial.selenic.com>
Buch: <http://hgbook.red-bean.com>

