TracNav

- JPFWiki - Welcome Page

  Introduction

  - What is JPF
  - Testing vs model checking
  - Random Example
  - Race Example
  - JPF classification

  Installing JPF

  - System requirements
  - Download snapshots
  - Download repositories
  - Create site.properties
  - Install NetBeans IDE plugin
  - Install Eclipse IDE plugin
  - Building and testing

  User Guide

  - Application Types
  - JPF Components
  - Configuring JPF
  - Running JPF
  - JPF Output
  - The JPF API

  Developer Guide

  - Design
  - Choice Generator
  - Partial Order Reduction
  - Attributes
  - Listener

    MJI

    - Mangling for MJI
  - Bytecode Factory
  - Logging
  - Report
  - Embedded
  - JPF tests
  - JPF project layout
  - Create a JPF project
  - Coding Conventions
  - Hosting update site

  Projects

  - jpf-core
  - jpf-actor
  - jpf-awt
  - jpf-awt-shell

    - jpf-concurrent
    - jpf-cv
    - jpf-delayed
    - jpf-guided-test
    - jpf-mango
    - jpf-racefinder
    - jpf-rtembed
    - jpf-statechart
    - net-iocache
    - jpf-aprop
    - jpf-numeric
    - jpf-symbc
    - jpf-concolic
    - jpf-symbc-load?
    - jpf-extended-test-gen
    - jpf-parallel-spf?
    - eclipse-jpf
    - netbeans-jpf
    - jpf-inspector
    - jpf-shell
    - jpf-template
    - jpf-trace-server
    - standard NB example
- Summer Projects
- External Projects
- Change(B)log

  About

    - About this Wiki
    - About the Mailing Lists
    - About the Development Process?
    - About the Repository?
    - How to Contribute
    - JPF contributor account
- Events
- Presentations
- Papers
- FAQ
- History?
- Support
- People?
- Playground
- Table of Context

Mercurial in 5 Minutes

Mercurial is a *distributed version control system* (DVCS) of the same category such as the likes of Git or Bazaar. If you know nothing else about DVCS, this means mostly one thing - **all repositories are created equal**. There is no such thing as a different repository format for public and private repositories. All repositories (that are synchronized) have the same, stored history information.

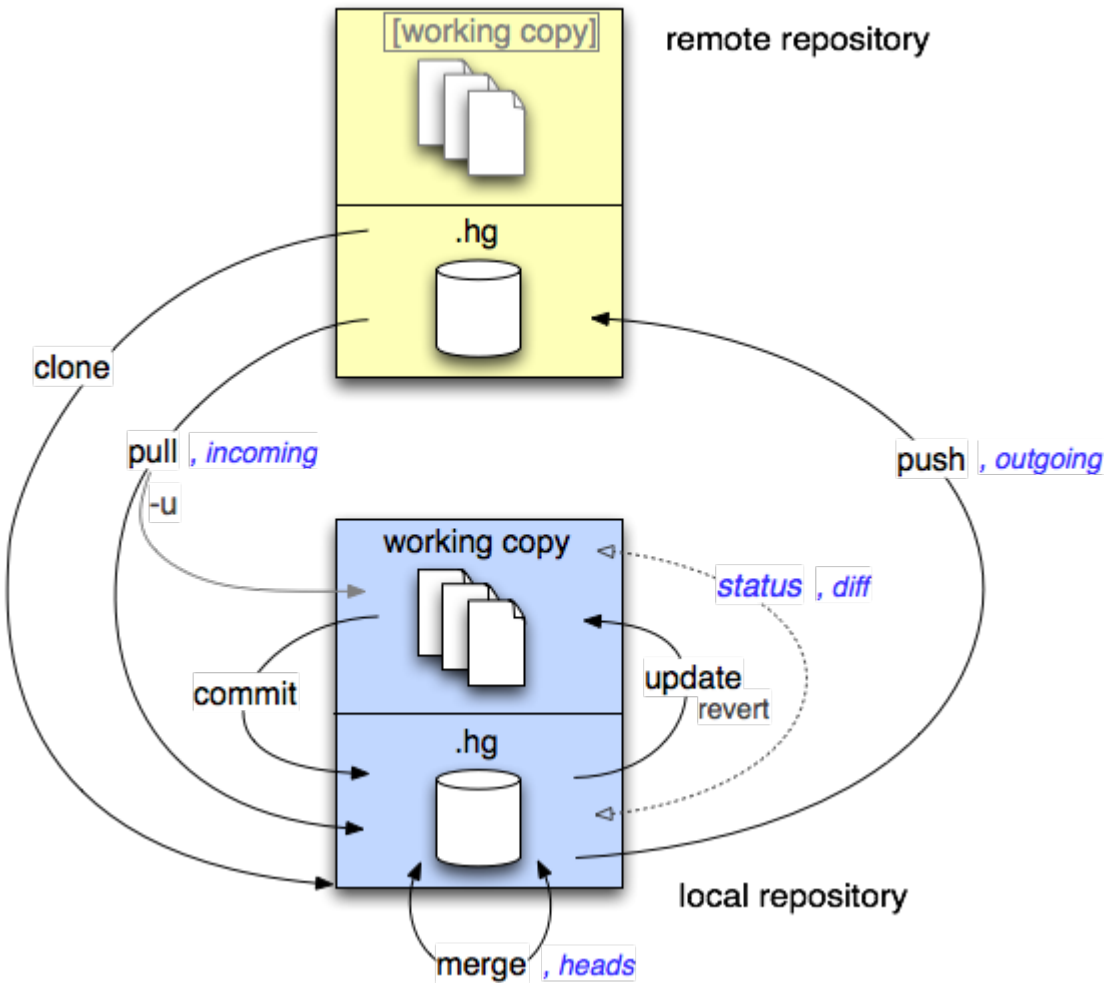
Yes, this means you can finally synchronize your working repos with more than one "master" repository, e.g. with several local ones, closed workgroup repos, and a public master repo (that just happens to be designated as such).

Each Mercurial repository consists of an (optional) working copy of your files, and a `.hg` directory in your repo root dir that holds all the version control info. Be careful not to delete the `.hg` dir, otherwise your repo is reduced to just a snapshot of your current working files.

A central concept of DVCS is the *change set* - think of it as the diff of all files that have been changed when you do a commit. This is what DVCS store (e.g. Mercurial in the `.hg` dir) and compare between different repositories. One implication of this is that you can't do partial commits like in CVS or SVN (e.g. by just committing changes within a certain subdirectory). You always have to execute the Mercurial commands in the top dir of the repository you are working in.

If you have previously worked with CVS or SVN, another difference you have to wrap your head around is that there are commands that work locally, and commands that interact with the remote repo. The `commit` and `update` commands are only local. The `push` and `pull` commands synchronize your local repo with an external one (like commit and update did in CVS/SVN).

Here are the main commands to interact with Mercurial:



hg clone <url> - this is the first command that clones an external repository (either from `file:///...` or `http://...` URLs). It creates both a working copy and the `.hg` directory (with contents)

hg init - is what you do if you create a local repository for which there is no external one yet. Just create your files, `cd` into the top directory, and execute the command (which will create the `.hg` for you)

hg pull [-u] <url> - updates the repo you are currently in from the provided URL. Note that your working copy is only updated if you use the `-u` option

hg incoming <url> - is the little brother of `pull`. It just tells you if there are changesets in the remote repository you would pull

hg status - tells you if there are uncommitted changes in your working copy that have to be committed to the local `.hg` directory before you can `push` or `pull`

hg diff - does a bit more, it also shows you a diff file with the uncommitted changes

hg commit - saves changes in your working copy to your local `.hg`. Do this early and often, this is now just a local operation that doesn't change anything outside the repo you are working with

hg update - would update your working copy from your local `.hg`. It is rarely used if you pull with the `-u` option

hg push <url> - pushes the relevant changesets of your local `.hg` back to the external repo with the provided URL. Make sure you have no uncommitted changes (Mercurial would refuse to push), and - in case the external repo is shared - that you pulled/merged all changesets of the external repo before you push

hg outgoing <url> - is the dry run version of a push, it tells you if there are changesets that would have to be pushed

hg revert - reverts your working copy to a previous version stored in your local `.hg`

hg heads - tells you if there are branches in your repo, which usually happens if you pull from a changed external repo while having un-pushed local changes that are committed

hg merge - gets rid of multiple heads. Make sure to get rid of these by merging as soon as you detect them, it only gets more difficult with more external and local changes. Once there are collisions, you are back to the CVS/SVN merge mess

You can always obtain more information by executing

```
> hg help [command]
```

If you don't list a command, all available ones will be displayed.

Commands that refer to external repos take URLs such as `http://babelfish.arc.nasa.gov/hg/jpf/jpf-core` as arguments.