JEP 357: Migrate from Mercurial to Git

Authors	Erik Duveblad, Joe Darcy
Owner	Joe Darcy
Type	Infrastructure
Scope	JDK
Status	Closed / Delivered
Release	16
Component	infrastructure
Discussion	discuss at openjdk dot java dot net
Effort	L
Duration	M
Relates to	JEP 369: Migrate to GitHub
Reviewed by	Mark Reinhold
Endorsed by	Mark Reinhold
Created	2019/07/12 02:20
Updated	2021/01/27 21:56
Issue	8227614

Summary

Migrate the OpenJDK Community's source code repositories from Mercurial (hg) to Git.

Goals

- Migrate all single-repository OpenJDK Projects from Mercurial to Git
- Preserve all version control history, including tags
- Reformat commit messages according to Git best practices
- Port the jcheck, webrev, and defpath tools to Git
- Create a tool to translate between Mercurial and Git hashes

Non-Goals

- We will not migrate multi-repository OpenJDK Projects, such as the JDK 8 Updates Project. Those Projects can migrate to Git if and when they consolidate into a single repository.
- We will not change the bug system from JBS.
- We will not aggressively decommission the existing Mercurial repos or take other actions to prematurely invalidate the many URLs pointing into the repos, such as in comments in bugs in JBS. The existing master Mercurial repos will at least be kept as read-only archives for a defined transition period. Longer term, a Mercurial URL to Git URL translator might be put into place.
- This JEP does not address the question of whether OpenJDK Git repositories will be self-hosted or hosted by an external provider. That issue is the topic of JEP 369: Migrate to GitHub.
- We will not propose changes to the current JDK development process, though this JEP does enable such changes.

Motivation

There are three primary reasons for migrating to Git:

1. Size of version control system metadata
2. Available tooling
3. Available hosting

Initial prototypes of converted repositories show a significant reduction in the size of the version control metadata. For example, the `.git` directory of the `jdk/jdk` repository is approximately 300 MB with Git and the `.hg` directory is around 1.2 GB with Mercurial, depending on the Mercurial version being used. The reduction in metadata preserves local disk space and reduces clone times, since fewer bits have to go over the wire. Git also features *shallow clones* that only clone parts of the history, resulting in even less metadata for those users who do not need the entire history.

There are many more tools for interacting with Git than Mercurial:

- All text editors have Git integration, either natively or in the form of plugins including Emacs (magit plugin), Vim (fugitive.git plugin), VS Code (builtin), and Atom (builtin).
- Almost all integrated development environments (IDEs) also ship with Git integration out-of-the-box, including Intellij (builtin), Eclipse (builtin), NetBeans (builtin), and Visual Studio (builtin).
- There are multiple desktop clients available for interacting with Git repositories locally.

Lastly, there are many options available for hosting Git repositories, whether self-hosted or hosted as a service.

Description

We have already prototyped a program to convert a Mercurial repository to a Git repository. It uses the `git-fast-import` protocol to import Mercurial changesets into Git, and it adjusts existing commit messages to align with Git best practices. A commit message for the Mercurial `jdk/jdk` repository has this structure:

> *JdkHgCommitMessage* : *BugIdLine*+ *SummaryLine*? *ReviewersLine* *ContributedByLine*?
>
> *BugIdLine* : /[0-9]{8}/ ": " *Text*
>
> *SummaryLine* : "Summary: " *Text*
>
> *ReviewersLine* : "Reviewed-by: " *Username* (", " *Username*)* "\n"
>
> *ContributedByLine* : "Contributed-by: " *Text*
>
> *Username* : /[a-z]+/
>
> *Text* : /[^\n]+/ "\n"

A commit message for the Git `jdk/jdk` repository will have a somewhat different structure:

> *JdkGitCommitMessage* : *BugIdLine*+ *Body*? *Trailers*
>
> *BugIdLine* : /[0-9]{8}/ ": " *Text*
>
> *Body* : *BlankLine* *Text**
>
> *Trailers* : *BlankLine* *Co-authors*? *Reviewers*
>
> *Co-authors* : (*BlankLine* *Co-author*)+
>
> *Co-author* : "Co-authored-by: " *Real-name* <*Email*>
>
> *Reviewers* : "Reviewed-by: " *Username* (", " *Username*)* "\n"
>
> *BlankLine* = "\n"
>
> *Username* : /[a-z]+/
>
> *Text* : /[^\n]+/ "\n"

The reasons to change the message structure are:

- The use of a title (a single line followed by a blank line) is strongly encouraged by the Git CLI tool.
- The use of a title then enables a free-form body.
- The Git ecosystem recognizes *trailers*, i.e., lines separated from the body with a newline. For example, both GitHub and GitLab recognizes the `Co-authored-by:` trailer and will recognize that the commit has multiple authors.

Git uses an additional field in the commit metadata to denote the committer of a commit, as distinct from the author. We'll use the committer field to signify sponsorship: in the event of a sponsored commit, the author will be named in the `author` field of the commit and the sponsor will be named in the `committer` field. If the sponsor also is a co-author, then an appropriate `Co-authored-by` trailer will be added, which is a situation that we cannot capture in the existing Mercurial message structure.

Another possible use of the distinct committer field is to identify backport commits from feature releases to update releases, which are typically done by someone other than the author of the original commit.

The content of the author and committer fields will be of the form *Real-name* <*Email*>, since not every commit author will be an OpenJDK Author. A special email address will be used to show that an author or committer is also an OpenJDK Author: `<openjdk-username>@openjdk.org`.

Here is an example of a Git commit message:

```
76543210: Crash when starting the JVM

Fixed a tricky race condition when the JVM is starting up by using a Mutex.

Co-authored-by: Robin Westberg <rwestberg@openjdk.org>
Reviewed-by: darcy
```

The author and committer field for such a commit would be:

```
Author: Erik Duveblad <ehelin@openjdk.org>
Commit: Erik Duveblad <ehelin@openjdk.org>
```

In the conversion process a commit will be considered as sponsored when the author is *not* listed on the `Contributed-by:` line. In that case the first person on the `Contributed-by:` line will be taken as the author, the sponsor will be taken as the committer, and any additional contributors will be taken as co-authors.

Examples of converted repositories are available at https://github.com/openjdk/.

Tools

We've already prototyped backward compatible ports of the Mercurial tools jcheck, webrev and defpath.

We've also prototyped new tool, `git-translate`. This tool uses a file called `.hgcommits` that is generated by the conversion tools and committed to the Git repositories. This file contains a sequence of lines, each of which contains two hexadecimal hashes: the first is the hash of a Mercurial changeset and the second is the hash of the Git commit resulting from converting that Mercurial changeset. The tool `git-translate` simply queries the file `.hgcommits`:

```
$ git translate --to-hg 0f8927e8b5bf88e7e2c7c453b4cd75e01eeccaf4
bd613b97c7c88658801b0f0c603a55345dfef022
$
```

Alternatives

Keep using Mercurial.

Testing

The `.hgcommits` mapping file, described above, can be used to verify that all the metadata for a given commit is correctly converted and also that the source code trees for the two commits are identical.

Risks and Assumptions

The outstanding risk is that errors are introduced as part of the conversion. That risk will be mitigated through rigorous verification as described above.

Dependencies

- JEP 296 (Consolidate the JDK Forest into a Single Repository)
- JEP 369: Migrate to GitHub