PatchRepository

Organization of the Patch Repository

The OpenJDK mlvm/mlvm repository forest is (at present) only a set of patches, not a full set of JDK sources. The patches apply to some version of the full jdk7/jdk7 forest. The structure of the mlvm forest parallels the structure of the full jdk7 sources, but each repository in mvlm is only the `.hg/patches` directory (of the Mercurial mq extension). Thus, repositories under mlvm are called "patch repositories" and those under jdk7 are by contrast called "source repositories".

Commits in the mlvm repositories do not update the full source trees, only the patches. To make this clear, when a commit occurs in a patch repository, we will refer to it specifically as a "patch commit".

TO DO: link the text in this page to suitable instructions for configuring and using Mercurial.

Patches

All patch files must end with the suffix `.patch`.

Patches must not have patch headers, since they are easy to lose if patches are regenerated.

All patches must be in "git" format, without file dates. To ensure this, add the following lines to your `.hgrc` file:

.hgrc

```
[diff]
nodates=1
git=1
```

A patch file may be accompanied by a similar file with the suffix `.txt`. This file will contain brief comments about the patch, including:

- references to project documentation, if needed
- draft commit comments
- dependencies on other patches

Patch repositories may also contain scripts and documentation. All these non-patch files are ignored by the Mercurial patch queue, since they do not appear in the series file.

Patches may be split. Files that contribute to a split patch must all have the same prefix up to the first dot.

Patches may depend on each other. (For example, invoke dynamic may depend on method handles, which in turn may depend on anonymous classes.) Such dependencies should be clearly stated.

The patch sequence is ordered by both stability and by dependency. If patch B is less stable than patch A, then B should come later in the series. If B depends on A, it also must come later in the series, and A must not be less stable than B.

Patches of the same name in multiple patch repositories (hotspot and jdk) are to be applied simultaneously to parallel source repositories. Their series file elements must be kept exactly in sync. The documenting text file for a patch does not need to be on both sides, and should not be duplicated.

Patch Guards

The series file contains guard annotations for each patch. Patches are guarded with the OpenJDK release tags that they apply to. In this way, as the OpenJDK release advances, patches can be rebased independently from each other. Patches must be rebased in a special-purpose "rebasing patch commit" which includes a change to the patch guard in the series file also. Actual development must be placed in patch commits that are **not** rebasing.

Each patch must have one or more positive guards, and each must be the tag of an OpenJDK build, such as `jdk7-b25`. If a patch is guarded by such a tag, it is guaranteed to apply, without rejects, to that particular OpenJDK build, and to build successfully.

Each patch must have a negative guard which names that patch with a "slash" prefix. This allows developers to control individual entries in the patch queue without editing the series file. Editing the series file is risky, since it is under version control and shared by all developers. By contrast, the guards file is not under version control.

The following guards are also significant as negative guards on patches which do not yet have the relevant quality level:

- `#-buildable`: the patch does not build, or interferes with the operation of the JVM
- `#-testable`: the patch fails to have a working test suite

For normal development, the guards 'buildable' and 'testable' should be present in the guard file, as well as the OpenJDK release in use.

Example series file entries:

mq series file

```
anonk.patch #-/anonk #+jdk7-b25
meth.patch  #-/meth  #+jdk7-b25
```

The 'qselect' command can be used to control these patches:

qselect commands

```
hg qselect jdk7-b25  # select both patches, plus any other jdk7-b25 ones
hg qselect jdk7-b25 /meth  # select anonk but not meth
```

References:

- more on guards: http://hgbook.red-bean.com/hgbookch13.html
- patch rebasing procedures: http://www.selenic.com/mercurial/wiki/index.cgi/MqMerge
- a good summary on rebasing: http://www.selenic.com/pipermail/mercurial/2008-February/017367.html

Multi-based Patches

If a patch must be split so as to apply to several OpenJDK builds, the latest patch in the series must be a complete patch for the most recent build, and for each previous build, a temporary patches must simply track the relevant changes up to the most recent build, so as to make the newest patch apply correctly in all cases.

For example, to support builds 25 and 28 behind build 30 two fixup patches are needed:

mq series file, split patch

```
anonk.jdk7-b25-b30.patch #-/anonk #+jdk7-b25
anonk.jdk7-b28-b30.patch #-/anonk #+jdk7-b28
anonk.patch #-/anonk #+jdk7-b30 #+jdk7-b28 #+jdk7-b25
```

(Note: This way of doing patches might prove difficult to manage in practice. We'll figure it out as we go along.)

Normally this will not be necessary, unless the patch provides functionality which several other patches depend on, and those patches are in different stages of rebasing.