The OpenJDK Developer Experience

Erik Duveblad
OpenJDK Developer @ Oracle
February, 2018
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Agenda

1. Developer Experience
2. Version Control
3. Building
4. Testing
5. Putting it all together
Who am I?

• Erik Duveblad
  – Works at Oracle as JVM GC developer
  – Worked with OpenJDK for 5 years
  – Contributes to:
    • HotSpot
    • Makefiles
    • JTReg, jcov, jtharness
  – ehelin @ #openjdk on irc.oftc.net
Agenda

1. Developer Experience
2. Version Control
3. Building
4. Testing
5. Putting it all together
Developer Experience

Definition

• *User experience* (UX)
  - “… a person's emotions and attitudes about using a particular product, system or service”

• Developers are users too!
  - version control, build systems, testing systems, frameworks, etc.

• *Developer experience* (DX)
  - the experience of using all the developer tools for a project
Developer Experience

Definition

• Applied to OpenJDK:
  - The VCS tools (hg, webrev, defpath, jcheck)
  - The build system (configure, make)
  - The test systems (JTReg, googletest, TestNG, Junit)
  - IDE support (IntelliJ, NetBeans, Eclipse, Visual Studio, Emacs, VIM)
  - Services (JBS, cr, wiki, mailing lists, IRC)

• Benefit of being a developer: both producer and consumer!
Developer Experience

Definition

• Applied to OpenJDK:
  – **The VCS tools** (hg, webrev, defpath, jcheck)
  – **The build system** (configure, make)
  – **The test systems** (JTReg, googletest, TestNG, Junit)
  – IDE support (IntelliJ, NetBeans, Eclipse, Visual Studio, Emacs, VIM)
  – Services (JBS, cr, wiki, mailing lists, IRC)

• Benefit of being a developer: both producer and consumer!
Agenda

1. Developer Experience
2. Version control
3. Building
4. Testing
5. Putting it all together
Version control

Mercurial (hg)

• OpenJDK uses Mercurial (hg) for version control
• JEP 296: Consolidate the JDK Forest into a Single Repository
  – Consolidated eight repositories to one!
  – $ hg clone http://hg.openjdk.java.net/jdk/jdk
• Benefits
  – Easier to get started
  – Atomic commits
  – One hash describes entire state
Version control

Mercurial (hg)

• Mercurial (hg) has also been greatly improved in 2017
  - Editing of local commits and history
    • amend, unamend, uncommit, split, strip, histedit, rebase, shelve
  - Improved user interface
    • ui.pagenate, ui.color, show, graphlog
  - Improved performance
    • diff (2x faster), zstd, revlog, conflict checking, copy detection, fsmonitor
Version control

Mercurial (hg)

• Mercurial (hg) has also been greatly improved in 2017
  - Editing of local commits and history
    • amend, unamend, uncommit, split, strip, histedit, rebase, shelve
  - Improved user interface
    • ui.pagenate, ui.color, show, graphlog
  - Improved performance
    • diff (2x faster), zstd, revlog, conflict checking, copy detection, fsmonitor
Version control
Mercurial (hg)

• Sample configuration (hg version 4.5)

```bash
[ui]
username = ehelin
pagenate = True
color = True

[extensions]
amend =
uncommit =
show =
graphlog =
sparse =
strip =
shelve =
record =
fsmonitor =
split =
```
Agenda

1. Developer Experience
2. Version control
3. Building
4. Testing
5. Putting it all together
Building

Makefiles

• Since around September 2016, the “new” build system is used for all code
  
  $ bash configure
  $ make

• configure will help you if dependencies are missing
  
  $ bash configure

  configure: error: Could not find a C++ compiler.
  You might be able to fix this by running
  'sudo apt-get install build-essential'.

• Extensive documentation in doc/building.md
Building

Makefiles

• OpenJDK requires a “boot JDK” for building
  - e.g. OpenJDK 9 requires JDK 8
• Often hard to find suitable “boot JDK” for development
  - jdk/jdk currently requires JDK 9, soon JDK 10
  - few distributions package that recent JDKs
  - the Oracle JDK is still proprietary
• Upstream binary builds of OpenJDK now available!
  - http://jdk.java.net/9/
Building

Makefiles

• Building from scratch on Debian 9

```
$ hg clone http://hg.openjdk.java.net/jdk/jdk
$ cd jdk
$ wget https://download.java.net/java/GA/jdk9/9.0.4/binaries/openjdk-9.0.4_linux-x64_bin.tar.gz
$ bash configure --with-boot-jdk=openjdk-9.0.4_linux-x64_bin.tar.gz
$ make images
```

• If any package is missing, configure will help you :)

Building

Makefiles

• Many DX improvements in the last year
  - debug symbols are no longer zipped by default
  - fontconfig no longer bundled
  - compiler errors are repeated at end of build output
  - all compiler commands for each file in *.cmdline files
  - run-test target introduced
Building

Makefiles

- Many DX improvements in the last year
  - debug symbols are no longer zipped by default
  - fontconfig no longer bundled
  - compiler errors are repeated at end of build output
  - all compiler commands for each file in *.cmdline files
  - run-test target introduced
Building

run-test

• New top-level make target: run-test
  
  $ make run-test TEST=test/jdk/java/lang/String/Split.java

• Supports multiple test frameworks
  - JTReg
  - googletest

• Supports running test suites, directories and files
  
  $ make run-test TEST=tier1
  $ make run-test-tier1
  
  $ make run-test TEST=test/hotspot/jtreg/gc
Building run-test

- Runs tests correctly
  - Tests from ProblemList.txt are ignored
  - Uses failure handlers
  - Always uses fresh, clean JTWork directory
  - Uses multiple cores if possible
  - Adds correct JVM flags
Building
run-test

• Supports passing options
  
  $ make run-test TEST="gtest:LogTagSet" GTEST="REPEAT=-1"

• Extensive documentation

  $ less doc/testing.md

• Requires jtreg

  $ bash configure --with-jtreg=path/to/jtreg/image
Agenda

1. Developer Experience
2. Version control
3. Building
4. Testing
5. Putting it all together
Testing
jtreg

- run-test requires passing --with-jtreg to configure
- Building jtreg *used* to be problematic...
  - javahelp only available in ancient SVN repository
  - jtharness required proprietary library (!)
  - Third-party dependencies not properly specified
  - Use of outdated third-party dependencies
  - Use of incompatible third-party dependencies
Testing
jtreg

• All this has been fixed!
  
  $ hg clone http://hg.openjdk.java.net/code-tools/jtreg
  $ cd jtreg
  $ sudo apt install openjdk-8-jdk
  $ bash make/build-all.sh /usr/lib/jvm/java-8-openjdk-amd64/

• Automatically downloads and verifies dependencies
  - http://hg.openjdk.java.net
  - https://repo1.maven.org/maven2

• Resulting image can be passed to configure
Testing

jtreg

• Can now build *and* test from scratch on Debian 9

$ hg clone http://hg.openjdk.java.net/jdk/jdk
$ cd jdk
$ bash configure
  --with-boot-jdk=../openjdk-9.0.4_linux-x64_bin.tar.gz
  --with-jtreg=../jtreg/build/images/jtreg

$ make run-test-tier1
Testing

jtreg

• JTReg testlibraries have converged
  − Available at test/lib/jdk/test/lib
  − Provides asserts, process tools, platform helpers etc.

• How to get started
  − http://openjdk.java.net/projects/code-tools/jtreg/intro.html
Testing
googletest

• googletest is available for writing C++ unit tests for HotSpot

$ cat test/hotspot/gtest/gc/g1/test_g1Analytics.cpp

```
#include "precompiled.hpp"
#include "gc/g1/g1Analytics.hpp"
#include "gc/g1/g1Predictions.hpp"
#include "unittest.hpp"

TEST_VM(G1Analytics, is_initialized) {
    G1Predictions p(0.888888);
    G1Analytics a(&p);
    ASSERT_EQ(a.recent_avg_pause_time_ratio(), 0.0);
    ASSERT_EQ(a.last_pause_time_ratio(), 0.0);
}
```
Testing

googletest

• How to get started
  - https://github.com/google/googletest/blob/master/googletest/docs/Primer.md
  - TEST runs test without VM
  - TEST_VM runs test in a shared VM
  - TEST_OTHER_VM runs test in separate VM
  - TEST_VM_ASSERT runs test in separate and expects failure

• Always make unittest.hpp the last include
Agenda

1. Developer Experience
2. Version control
3. Building
4. Testing
5. Putting it all together
Putting it all together

Workflow

• Clone the repository
  $ hg clone http://hg.openjdk.java.net/jdk/jdk && cd jdk

• Create a branch for your issue
  $ hg branch JDK-8012345

• Name your build profile the same as your branch
  $ bash configure --with-conf-name=JDK-8012345
  "--with-jtreg=../jtreg/build/images/jtreg"
  "--with-boot-jdk=../linux_9.0.4_x64_bin.tar.gz"

• Hack, write tests, commit, run tests locally
  $ make CONF_NAME=JDK-8012345 run-test-tier1 run-test-tier2
Putting it all together

Workflow

• Add sandbox and submit repositories as paths

```bash
$ echo >> .hg/hgrc <<EOF
[paths]
submit = http://hg.openjdk.java.net/jdk/submit
sandbox = http://hg.openjdk.java.net/jdk/sandbox
EOF
```

• Push to the submit repository for *final* testing

```bash
$ hg push -b JDK-8012345 --new-branch submit
```

• Push to the sandbox to share

```bash
$ hg push -b JDK-8012345 --new-branch sandbox
```
Putting it all together

Future

• make compile-commands
  - Creates compile_commands.json
  - Better integration with
    • clang-tools
    • cquery
Putting it all together

Join us!

• Feedback
  - what can be improved?

• Contribute
  - start hacking on code-tools, Makefiles, test libraries

• Keep in touch
  - #openjdk @ irc.oftc.net
  - code-tools-dev@openjdk.java.net