Skara

Repository

- http://git.openjdk.java.net/skara/

Mailing List

- skara-dev@openjdk.java.net (Subscribe, Archives)

Issues

- https://bugs.openjdk.java.net/projects/SKARA/summary

IRC

- `#skara` on OFTC

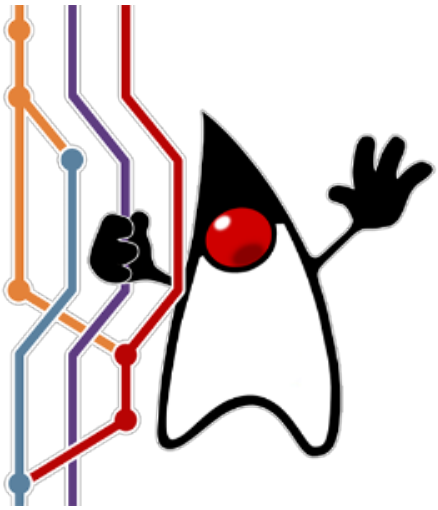

Table of Contents

Introduction

The goal of Project Skara is to investigate alternative SCM and code review options for the OpenJDK source code, including options based upon Git rather than Mercurial, and including options hosted by third parties.

The technical parts of project Skara includes several server-side tools (also called "bots") aiding contributors during code reviews. The Skara technical tooling also includes several command-line utilities for interacting with Git source code hosting providers from the command-line.

Using an *external* Git source code hosting provider comes with several benefits, including:

- Performance
- Community
- API

Many, if not all, external Git source code hosting services available have excellent performance, not only with regards to network performance but also when it comes to availability (uptime). The largest Git source code hosting providers also offer the OpenJDK community to tap into large, existing, communities of developers and potential contributors. An additional benefit of using an external Git source code hosting provider is getting access to an API. These APIs enables programs to interact with developers on the external Git source code hosting provider. Although not impossible to achieve today by interacting with developers over email, it is considerably harder to implement programs that interpret free form text in emails compared to using a structured API.

A significant risk when using an external source code hosting provider is to become dependent on the external source code hosting provider. The version control data itself will always be independent of source code hosting provider due to distributed architecture of Git itself. However there is a large risk in metadata such as code review comments becoming "locked in" on a particular external source code hosting provider. Mitigating this risk is a large part of Project Skara and the following work have been done so far:

- replicating all discussions in all pull requests to the OpenJDK mailing lists
- archiving all discussions in pull requests in two formats:
    - mbox (for human consumption)
    - json (for software consumption) (*not implemented yet*)
- notifications of all pushes to the corresponding *-changes@openjdk.java.net mailing lists to avoid dependence on any provider's RSS feeds
- using the OpenJDK census for user organization and privilege level to avoid any dependencies on external Git source code hosting provider's user organization tool
- setting up to the domain http://git.openjdk.java.net/to redirect to OpenJDK's current external source code hosting provider to avoid polluting JBS and mailing lists with direct links to an external Git source code hosting provider

Support for *multiple* external source code hosting provider has been a strict requirement for *all* server-side and client-side tooling to avoid the issue of having any tooling take on a dependency on a particular external source code hosting provider's API. All tooling is also required to work with the GitLab Community Edition (GitLab CE) which is an open source project.

Another large part of Project Skara has been to ensure that there are *multiple* workflows available when interacting with a Git external source code provider, including one that preservers as much as possible of the OpenJDK community's current workflow. The Skara tooling currently supports the following workflows:

- Mailing list + CLI based (similar to current workflow)
- Web browser + CLI based
- CLI only
- Text editor/IDE

All workflows can be used interchangeably and different contributors can use different workflows at the same time (even while working on the same change). The support of multiple workflows comes from the APIs provided by the external Git source control hosting provider. Examples of the work done to support multiple concurrent workflows include:

- Two-way mailing list synchronization (you can comment on pull requests via OpenJDK mailing lists)
- Automatic "RFR" emails sent to mailiing lists for newly created pull requests
- Automatically determining mailing lists to "CC" for newly created pull requests
- Automatic generation and hosting of "webrevs" (including incremental webrevs)
- Automatically run "jcheck" on every commit in every pull request
- Implement CLI tools to list, fetch, view, approve and integrate pull requests
- Implement backwards compatible ports of jcheck, webrev and defpath

Getting Started

The following sections will get you started with Git, the external Git source code hosting provider GitHub and, if you want, the Skara CLI tooling.

Git

Install Git CLI client

GNU/Linux

CentOS/Oracle Linux/RHEL

```
$ sudo yum install git
```

Debian/Ubuntu

```
$ sudo apt install git
```

Fedora

```
$ sudo dnf install git
```

macOS

There are two ways to install Git on macOS: using Homebrew or using a direct download.

Using Homebrew

- Install Homebrew: https://brew.sh/
- `brew install git`

Using direct download

- Go to https://git-scm.com/download/mac

Windows

Install Git For Windows (maintained by Microsoft): https://gitforwindows.org

Initial Configuration

Git requires that you configure a username and an email. Use your full name as your username and your regular email address for the email:

```
$ git config --global user.name 'Your Full Name'
$ git config --global user.email 'your.name@host.com'
```

For example my name is "Erik Duveblad" and my email is "erik.helin@oracle.com". Therefore I would run:

```
$ git config --global user.name 'Erik Duveblad'
$ git config --global user.email 'erik.helin@oracle.com'
```

Additional Clients

There are several additional clients available for Git that can be used instead of (or in combination with) the Git CLI tool. Please see the following subsections for how to interact with Git from the desktop, IDE or a text editor.

Desktop

- SourceTree (macOS, Windows)
- gitg (GNU/Linux)
- Tower (macOS, Windows)
- GitKraken (GNU/Linux, macOS, Windows)
- Sublime Merge (GNU/Linux, macOS, Windows)
- TortoiseGit (Windows)

IDE

The following integrated develop environments (IDEs) all have Git support built-in:

- Eclipse
- NetBeans
- IntelliJ IDEA
- Visual Studio

Text Editors

The following text editors either have Git support built-in or as part of a plugin:

- Vim (fugitive.vim plugin)
- Emacs (magit plugin)
- VS Code (builtin)
- Atom (builtin)

GitHub

GitHub is an external Git source code hosting provider at https://github.com/. To create an account and get started, follow the instructions at https://github.com/join. We strongly recommend that all OpenJDK contributors enable two-factor authentication (2FA) on GitHub. To enable 2FA for your account on GitHub, follow the instructions at https://help.github.com/en/articles/securing-your-account-with-two-factor-authentication-2fa.

Associating your GitHub account and your OpenJDK username

If you are an OpenJDK Author, Committer or Reviewer then you can associate your GitHub account with your OpenJDK username by opening an issue at https://bugs.openjdk.java.net/secure/CreateIssue.jspa?pid=11300&issuetype=1. As a title for the issue, please use "Associate GitHub user <YOUR-GITHUB-USERNAME> with OpenJDK user <YOUR-OPENJDK-USERNAME>".

This way the server-side tooling (the "bots") will recognize you on GitHub as an OpenJDK Author, Committer or Reviewer.

Skara

The Skara projects contains tooling for contributors working with OpenJDK projects and their repositories. The following CLI tools are available:

- git-jcheck - a backwards compatible Git port of jcheck
- git-webrev - a backwards compatible Git port of webrev
- git-defpath - a backwards compatible Git port of defpath
- git-fork - fork a project on an external Git source code hosting provider to your personal space and optionally clone it
- git-pr - interact with pull requests for a project on an external Git source code hosting provider
- git-info - show OpenJDK information about commits, e.g. issue links, authors, contributors, etc.
- git-token - interact with a Git credential manager for handling personal access tokens
- git-translate - translate between Mercurial and Git hashes
- git-skara - learn about and update the Skara CLI tools

As mentioned in the "Introduction", all tools support multiple external Git source code hosting providers.

It is **not** necessary to use the Skara tools to contribute changes. Contributors that prefer to use e.g. desktop applications and/or IDEs that integrate with applicable external Git source code hosting providers are free to do so.

Installing

To install the Skara tooling, simply clone the Skara repository and include the Skara Git configuration file:

```
$ git clone http://git.openjdk.java.net/skara
$ git config --global include.path "$PWD/skara/skara.gitconfig"
```

The Skara tooling will build itself the first time you use any of the Skara commands. To check that everything works run `git skara help`.

Personal Access Token

Some of the Skara tools requires a "Personal Access Token" (PAT) to authenticate against an external Git source code hosting provider's API. These tools include:

- git-fork
- git-pr

If you do not intend to use the above tools, then there is no need to set up a PAT and you can skip this section. If you want to make use of the above tools, then please read on.

Git Credential Manager

The first step is to ensure you have a Git credential manager to store your PAT once it has been generated. See the subsections below for how to set up a Git credential manager for you operating system.

Windows

If you installed Git via https://gitforwindows.org/ then you already have a credential manager from Microsoft installed (it is bundled with "Git for Windows"). If you installed Git via some other mechanism, then install https://github.com/Microsoft/Git-Credential-Manager-for-Windows.

macOS

You already have a Git credential manager in Keychain, there is nothing to install or configure.

GNU/Linux

On GNU/Linux the recommended setup is to use libsecret and the "libsecret credential helper" in order to use GNOME Keyring as the Git credential manager. If you are using a desktop environment or distribution without support for GNOME Keyring, please see the "Maual" section.

Fedora

Fedora 29 and 30 (the only two currently supported versions of Fedora) comes with libsecret and GNOME Keyring installed by default. When you install the git package you also get the libsecret credential helper installed. To configure git to use the libsecret credential helper run:

```
$ git config --global credential.helper /usr/libexec/git-core/git-credential-libsecret
```

If you want to have a graphical utility to inspect the GNOME Keyring we recommend that you install GNOME Seahorse:

```
sudo dnf install seahorse
```

Ubuntu

Ubuntu 19.04 and 18.04.2 (LTS) (the only two currently supported versions of desktop Ubuntu) comes with libsecret and GNOME Keyring installed by default. Unfortunately even if you install the Git package you will not get a binary version of the libsecret credential helper installed (you only get the source). This means you have to compile the libsecret credential helper yourself. This is easy to do, it just requires two extra commands:

```
$ sudo apt install libsecret-dev
$ sudo make --directory=/usr/share/doc/git/contrib/credential/libsecret
```

Once you have compiled the libsecret credential helper you must configure Git to use it:

```
$ git config --global credential.helper /usr/share/doc/git/contrib/credential/libsecret/git-credential-
libsecret
```

If you want to have a graphical utility to inspect the GNOME Keyring we recommend that you install GNOME Seahorse:

```
$ sudo apt install seahorse
```

Manual

If you are using a desktop environment or distribution without support for GNOME Keyring, or if you want to use your own scheme for storing the PAT, then that is also supported. You can store non-sensitive data such as your username and the URL of the Git source code hosting provider in your `~/.gitconfig` file in the "credentials" section:

```
[credentials "https://github.com"]
username = foobar
```

For the PAT itself, all Skara tools interacting with an external Git source code hosting provider's API supports the `GIT_TOKEN` environment variable. This means that instead of storing your PAT in a Git credential manager you can store it e.g. encrypted on disk using `gpg` and then set the `GIT_TOKEN` environment variable to the decrypted value when using applicable Skara tools. For example:

```
GIT_TOKEN=$(gpg --decrypt ~/pat.gpg) git pr list
```

Creating a Personal Access Token

For creating a PAT on GitHub, please see https://help.github.com/en/articles/creating-a-personal-access-token-for-the-command-line. You need to generate a token with "repo" scope (as is described in the linked guide). After you have generated your token, store it in your credential manager using `git token store`:

```
$ git token store https://github.com
Username: <insert your Github username>
Password: <insert your "Personal Access Token", not your GitHub password>
```

Updating

To update the Skara tooling run `git skara update`. this will pull eventual updates and rebuild the tooling if necessary.

Workflows

As mentioned in the "Introduction" there are several workflows available depending on a contributor's preference. In this section we will describe how the Skara tooling can be used, which is one particular workflow. We will also include links to external resources for those whishing to set up e.g. desktop applications, text editors and/or IDEs to integrate with a Git external source code hosting provider.

Desktop Applications

The following desktop applications has integrations with one or more Git external source code hosting providers:

- SourceTree (macOS, Windows)
- Tower (macOS, Windows)
- GitKraken (GNU/Linux, macOS, Windows)
- GitHub Desktop (Arch Linux, macOS, Windows)

IDEs

The following integrated development environments (IDEs) integrates with one or more Git external source code hosting providers:

- Eclipse (requires MyLyn connector)
- IntelliJ IDEA (builtin)
- Visual Studio (requires GitHub Extension for Visual Studio plugin)

Text Editors

The following text editors integrates with one or more Git external source code hosting providers:

- Emacs (requires magit plugin and forge plugin)
- VS Code (requires GitHub Pull Requests plugin)
- Atom (requires GitHub for Atom plugin)

CLI

The following command-line interface applications integrates with one or more Git external source code hosting providers:

- hub (FreeBSD, GNU/Linux, macOS, Windows)

Skara

The Skara tooling enables a CLI driven workflow where reviews are made either via the mailing lists or in an web browser using an external Git source code hosting provider's web application. The first step in the Skara workflow is to fork and clone an existing upstream repository and create a local branch. A contributor will then make his or her changes, run jcheck on the local commits, push the branch to his or her fork and then finally create a pull request:

```
$ git fork https://github.com/openjdk/skara skara
$ git checkout -b bugfix
$ # make changes
$ git jcheck --local
$ git push --set-upstream origin bugfix
$ git pr create
```

Note that the `git jcheck --local` step above is not required, the pull request bot will run jcheck on all commits when the pull request is created.

The contributor will now use the mailing lists or the external Git source code hosting provider's web application to respond to feedback from reviewers. Reviewers can review the change with any tools they prefer (including automatically generated webrevs linked from the automatically generated "RFR" email).

A reviewer who wish to use the Skara tooling to mark a pull request as approved can do so by running `git pr approve`.

The contributor can integrate changes once reviewers are satisfied with them and has has marked them as reviewed. This is done by executing `git pr integrate`.