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

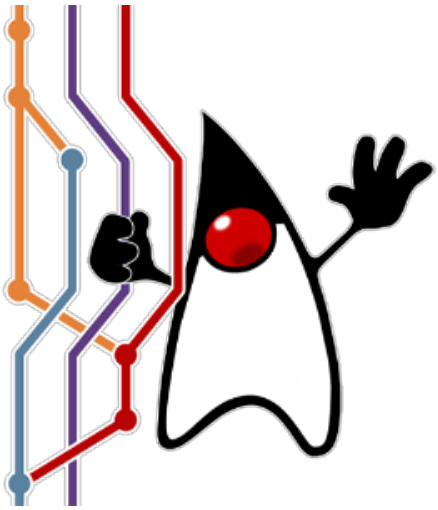

Table of Contents

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 the 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 levels 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 preserves 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
- Desktop application
- Mobile/tablet application
- 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 mailing 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 adding links to pull requests in issues
- Automatically adding links to issues in pull requests
- 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 and the external Git source code hosting provider GitHub.

Git

Installing the 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. Make sure you select the option to always use unix line endings. The OpenJDK build system does not work if the source code has Windows line endings. If you missed this during installation, you can correct it like this:

```
$ git config --global core.autocrlf false
```

The OpenJDK build system runs best in Cygwin and while you may use Git from Cygwin for most Git operations, it will not work well with the Skara tools. The native Windows Git works well enough in Cygwin as long as you do not feed it Cygwin style absolute paths and keep the autocrlf setting as instructed above.

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'
```

You will also want to setup the editor that Git will use whenever you need to enter multiple lines of input, for example when writing a commit message or doing an interactive rebase. I use `vim` as my editor and therefore I would run:

```
$ git config --global core.editor 'vim'
```

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, mobile, tablet, IDE or a text editor.

CLI

The following CLI applications can all interact with Git repositories:

- lazygit (GNU/Linux, macOS, Windows)
- tig (GNU/Linux, macOS, Windows)

Desktop

The following desktop applications can all interact with Git repositories:

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

Mobile/Tablet

The following mobile/tablet applications can all interact with Git repositories:

- Working Copy (iOS, ipadOS)
- Pocket Git (Android)

IDE

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

- Eclipse
- NetBeans
- IntelliJ IDEA
- Visual Studio
- Xcode

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)
- Sublime Text (builtin)

Resources

There are a lot blog posts, tutorials, tweets and material about Git on the internet. Unfortunately much of this material is outdated since Git was created back in 2005. We strongly recommend the *online* or *e-book* version of the "Pro Git" book by Scott Chacon and Ben Straub available at https://git-scm.com/book/en/v2 for learning more about Git (the print version is not up-to-date). The book is available online in HTML or can be downloaded in pdf, epub and/or mobi formats. The book is available under the CC BY-NC-SA 3.0 license and a community keep it continuously up-to-date by contributing to the book's Git repository.

For more information about a particular `git` command, see the online reference or the man page for the command (`git help <commmand>`)

If you want to learn more about the inner workings of Git and how it can be implemented then the online book "Write yourself a Git!" by Thibault Polge is a good resource: https://wyag.thb.lt/

GitHub

GitHub is an external Git source code hosting provider at https://github.com/.

Creating an account

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 "Add GitHub user <YOUR-GITHUB-USERNAME>". This way the server-side tooling (the "bots") will recognize you on GitHub as an OpenJDK Author, Committer or Reviewer.

After you have filed the issue and it has been handled by the Skara administrators you will get an e-mail from GitHub with an invitation to join the OpenJDK organization on GitHub. The OpenJDK organization on GitHub is just a mirror of the OpenJDK census, it is *not* used to control access rights to repositories etc. If you want your GitHub profile to show that you are a member of the OpenJDK organization, please see the GitHub documentation for how to enable this.

OpenJDK Email

When you integrate changes to an OpenJDK repository, the author will be "Your Name <openjdk user at openjdk.org>". After completing the steps above, this email should start forwarding to the address you have registered with OpenJDK. If you add this email to your Github account, Github will automatically link your OpenJDK commits with your Github user.

Resources

The GitHub Guides provides a good introduction to many of the concepts on GitHub. For more information about a particular topic, see the GitHub reference documentation. If you prefer watching a video over reading an article then there are a number of good introduction videos on GitHub Guide's YouTube channel. If you want to learn using a more hands-on approach, then we recommend trying out the GitHub Learning Lab.

Note: Skara makes some slight changes to parts of the GitHub workflow, see the Workflows section for more details

Workflow

As mentioned in the "Introduction" there are several tools a contributor can use based on their preference, but all tools realize an abstract workflow that has two distinct features:

- every contributor will have a *personal fork* of an OpenJDK repository they want to contribute to
- every change will start out as a *pull request*

The following two sections will describe the concepts of a personal fork and a pull request in more details. If you are new to Git then we recommend that you read at least chapter 2 (Git Basics) and chapter 3 (Git Branching) in the Pro Git book before proceeding.

The last two sections give an overview of the abstract workflow and the services and commands that Skara's server-side tooling (bots) provide.

Personal forks

A *personal fork* is a copy of another repository with one major difference:

- you can use a pull request to suggest that some changes from your personal fork should be incorporated into the original repository the fork was created from

There are several advantages to each contributor having a personal fork of the original OpenJDK repository they want contribute to:

- Contributors can freely experiment in their personal fork without affecting the original repository
- Contributors can back-up work that is in progress by pushing it to their personal fork
- Contributors can do ad-hoc collaboration with other contributors in their personal forks

There is also one drawback of each contributor having a personal fork:

- The personal fork must from time to time be synced with the original repository. See the FAQ for how to sync a personal fork.

This single drawback is fortunately remedied with the help of tools that reduce the overhead of syncing a personal fork with the original repository to almost nothing.

An example of a personal fork is edvbld/jdk which is a personal fork of the openjdk/jdk repository.

The concept of a personal fork is present in almost all external Git source code hosting providers.

Pull requests

A *pull request* is a way to suggest that some changes from a personal fork should be incorporated into the original repository the personal fork was created from. Reviewers can comment upon and need to approve a pull request before it can be integrated. The concept of a pull request is very similar to OpenJDK's concept of "RFR" emails - both are used to suggest changes and offer a way for reviewers to provide feedback on the suggested changes.

Pull requests are most commonly created from a branch in a personal fork and are said to be *targeting* a branch in the original repository the fork was created from. For example Jorn has created a pull request targeting the foreign-jextract branch of the OpenJDK panama-foreign repository from the ErrorCode branch in his personal fork of the panama-foreign repository.

Reviewers can leave comments on a pull request for the author. The author can use the feedback from the reviewers to *update* the pull request. The pull request is updated by pushing commits to the branch in the author's personal fork that the pull request was created from.

The outcome of a pull request that has been approved by reviewers is a commit on the branch that the pull request is targeting. The pull request has then been *integrated*.

Pull request commands

Project Skara provides contributors and reviewers with additional pull request commands that enable additional functionality. A *pull request command* is a comment made to a pull request that starts with a slash ("/"), for example "/integrate". This is a example where the Skara workflow differs slightly from the workflow offered by most external Git source code hosting providers - almost all external Git source code hosting providers require that a reviewer/maintainer integrates a pull request into a repository. Skara instead enables the *contributor* to integrate the pull request with the "/integrate" command, but the contributor can only issue the "/integrate" command once the pull request passes all pre-integration checks (e.g. jcheck).

For more information about available commands and what they do, please see the reference.

Overview

With the concepts of personal forks and pull requests the abstract and high-level workflow for a contributor looks like:

- Contributor creates a personal fork of the OpenJDK repository they want to contribute to
- For each change the contributor wants to suggest:
    - Contributor creates a branch in their personal fork
    - Contributor pushes commit(s) describing suggested change to the above created branch
    - Contributor creates a pull request from the above branch in the contributor's personal fork towards a branch in the original OpenJDK repository
    - Contributor updates the pull request based on feedback from reviewers (if needed)
    - Reviewers approve contributor's pull request
    - Contributor integrates the pull request

Note that the contributor only has to create a personal fork *once*, the same personal fork can be re-used for multiple pull requests. Note also that all steps in the above abstract workflow might not be needed depending on which tools the contributor choose to use to realize this workflow.

Tools

A contributor can choose between multiple different tools to achieve a workflow suited to their personal preferences. Some contributors might prefer a close integration with their IDE of choice while others might work on servers not even having a graphical environment. A contributor can also choose to combine multiple different tools to great effect. We can not in detail cover all possible setups, but the most common ones will be covered in the section Recommended Setups.

The following sections provides external links for those wishing to setup e.g. desktop applications, mobile apps 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 external Git source code hosting providers:

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

Mobile/Tablet

The following mobile and tablet applications integrates with one or more external Git source code hosting providers:

- GitHub for Mobile (iOS, ipadOS, Android)
- Working Copy (iOS, ipadOS)
- GitPoint (iOS, Android)
- Git Hawk (iOS)
- FastHub (Android)

IDEs

The following integrated development environments (IDEs) integrates with one or more external Git 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 external Git 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)
- GitHub CLI (GNU/Linux, macOS, Windows)
- Skara (GNU/Linux, macOS, Windows)

Web Browser

The following web applications (web sites) can be used from a web browser:

- GitHub (Firefox, Safari, Chrome, Chromium, Edge)

Mailing Lists

Skara enables two-way synchronization between the OpenJDK mailing lists and external Git source soure code hosting providers. This enables contributors and reviewers to use the OpenJDK mailing lists for discussing pull requests. The following email clients are recommended to interact with OpenJDK mailing lists:

- Thunderbird (GNU/Linux, macOS, Windows)
- mutt (*BSD, GNU/Linux)

Setups

The following sections describe recommended setups and provide a good starting to point for further exploration of other tools. For contributors who are completely new to Git and external Git source code hosting providers we recommend starting out with using the Git CLI client and a web application via a web browser. Many other tools assume a familiarity with this setup and it is also the most well documented. For contributors who are new to Git and not as used to the command-line we recommended starting out with a Git desktop client and a web application via web browser. For a bit more experienced contributors primarily using CLI applications we recommend using the Skara CLI tools and the OpenJDK mailing lists.

Git CLI Client + Web Browser (recommended)

This setup uses the Git CLI client in combination with a web browser to use the external Git source code hosting provider's web application. To use this setup contributors must have the following applications installed:

- a terminal emulator (e.g. GNOME Terminal, Konsole, Terminal, iTerm2, Windows Terminal)
- a web browser (e.g. Chrome, Firefox, Safari, Edge)
- the Git CLI client

Contributors will start out by creating a personal fork of the upstream OpenJDK repository they want to contribute to. The contributor will then use a terminal emulator and the Git CLI client to clone the personal fork to local repository on their computer (see chapter 2 "Git Basics" for how to do this). The contributor will create a local branch and then continue to make a number of local commits on that local branch. Finally, when the contributor is ready to propose the change, they will push their local branch (with the local commits) to their personal fork. The following is an example of Erik cloning his personal fork of the OpenJDK Skara repository, making a branch, making a commit and finally pushing the branch and commit to his personal fork:

```
$ git clone https://github.com/edvbld/skara
cloning into 'skara'...
remote: Enumerating objects: 48, done.
remote: Counting objects: 100% (48/48), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 13764 (delta 5), reused 40 (delta 3), pack-reused 13716
Receiving objects: 100% (13764/13764), 2.18 MiB | 941.00 KiB/s, done.
Resolving deltas: 100% (4565/4565), done.

$ cd skara

$ git checkout -b SKARA-296
Switched to a new branch 'SKARA-296'

$ # hack, hack

$ git add <paths/to/files/that/have/changed>

$ git commit -m skara-296
[SKARA-296 88d7b01] skara-296
 1 file changed, 17 insertions(+), 4 deletions(-)

$ git push --set-upstream origin SKARA-296
Enumerating objects: 25, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (13/13), 1.11 KiB | 1.11 MiB/s, done.
Total 13 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
remote:
remote: Create a pull request for 'SKARA-296' on GitHub by visiting:
remote:      https://github.com/edvbld/skara/pull/new/SKARA-296
remote:
To https://github.com/edvbld/skara
 * [new branch]      SKARA-296 -> SKARA-296
```

The contributor will create the pull request by clicking on the link shown in the output of the command `git push --set-upstream origin SKARA-296`. The contributor will then continue to interact with reviewers and making pull request commands via the web browser and the external Git source code hosting provider's web application.

Note that with this setup the terminal session does not have to be on the same computer as the web browser - it is perfectly fine to have the Git repository on a server that the contributor SSH into and then have the web browser on another machine.

Once the contributor has created a couple of pull requests they will probably want to do some additional configuration to speed up parts of the workflow. We would recommend exploring the following items:

- Generating an SSH key
- Using ssh-agent for SSH keys
- Adding an SSH key to your GitHub account
- Use a shorthand for GitHub URLs when cloning over SSH
- Adding aliases for common commands
- Display the currently checked out branch in the Bash prompt

Git Desktop Client + Web Browser

This setup is very similar to Git CLI Client + Web Browser but instead of using the Git CLI client for interacting with a local and/or remote Git repository, you use a desktop application. If you are using macOS or Windows as your operating system, then we recommend the SourceTree Git desktop application. If you want a Git desktop application that is more integrated with GitHub, then we recommend GitHub Desktop.

Skara CLI Tools + Mailing Lists

This setup is the one with the most in common with the traditional OpenJDK workflow. Contributors will use the Skara CLI tools to create, list, update and integrate pull requests while using the OpenJDK mailing lists to interact with reviewers and add comments to pull requests (thanks to Skara prodiving bi-directional synchronization between mailing lists and external Git source code hosting providers). See the Skara CLI tools page for more information about this setup.

Reporting issues

To report any issues, please file a bug on the Skara project in the JDK Bug System (JBS). If you have any questions and/or comments, then you can also send an e-mail to project Skara's mailing list.