GeckoView Documentation

Release 66

Mozilla

Jan 09, 2019

Contents:

CHAPTER 1

GeckoView Contributor Quick Start Guide

This is a guide for developers who want to contribute to the GeckoView project. If you want to get started using GeckoView in your app then you should refer to the [wiki](https://wiki.mozilla.org/Mobile/GeckoView#Get_Started).

You may also be interested in how to get up and running with [Firefox For Android(https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Build_Instructions/Simple_Firefox_for_Android_build).

The GeckoView codebase is part of the main Firefox tree and can be found in *mozilla-central*. You will need to get set up as a contributor to Firefox in order to contribute to GeckoView. To get set up with *mozilla-central*, follow the [Quick Start Guide for Git Users](MozCentralQuickStart.md), or the [Contributing to the Mozilla code base](https://developer.mozilla.org/docs/Mozilla/Developer_guide/Introduction) guide on [MDN](https://developer.mozilla.org/) for Mercurial users.

Once you have a copy of *mozilla-central*, you will need to build GeckoView.

## Bootstrap Gecko Bootstrap configures everything for GeckoView and Fennec development.

- Ensure you have *mozilla-central* checked out. If this is the first time you are doing this, it may take some time.

```
git checkout central/default
```

If you are on a mac, you will need to have the Xcode build tools installed. You can do this by either [installing Xcode](https://developer.apple.com/xcode/) or installing only the tools from the command line by running *xcode-select –install* and following the on screen instructions. Use the ‘ –no-interactive‘ argument to automatically accept any license agreements.

```
./mach bootstrap [ --no-interactive]
```

- Choose option *4. Firefox for Android* for GeckoView development. This will give you a version of Gecko configured for Android that has not bundled the native code into embedded libraries so you can amend the code.
- Say Y to all configuration options
- Once *mach bootstrap* is complete it will tell you to copy and paste some configuration into your *mozconfig* file. The *mozconfig* file can be found in the root of your *gecko* repo - or create a file called *mozconfig* if it does not exist. Check that the correct value is associated with the *–target* argument as this may not correctly match your setup. Your *mozconfig* should read something like this:

```
mk_add_options MOZ_OBJDIR=../objdir-android-opt

# Build Firefox for Android:
ac_add_options --enable-application=mobile/android
ac_add_options --target=<your target architecture>

# With the following java and javac:
ac_add_options --with-java-bin-path="/Library/Java/Home/bin"

# With the following Android SDK and NDK:
ac_add_options --with-android-sdk="$HOME/.mozbuild/<your-sdk>"
ac_add_options --with-android-ndk="$HOME/.mozbuild/android-ndk-r17b"
```

- Configure your build.

```
./mach configure
```

## Build from the command line

In order to pick up the configuration changes we just made we need to build from the command line and package the app.

```
./mach build
./mach package
```

- Install [Android Studio](https://developer.android.com/studio/install).
- Disable Instant Run. This is because Fennec and the Geckoview Example app cannot deploy with Instant Run on.
  - Select Android Studio > Preferences from the menu bar
  - Navigate to Build, Execution, Deployment > Instant Run.
  - Uncheck the box that reads *Enable Instant Run to hot swap code/resource changes on deploy*.

  ![alt text](assets/DisableInstantRun.png "Disable Instant Run")
- Choose File->Open from the toolbar
- Navigate to <path to gecko>/mobile/android/geckoview and click "Open"
- Click yes if it asks if you want to use the gradle wrapper.
- Wait for the project to index and gradle to sync. Once synced, the workspace will reconfigure to display the different projects.
  - annotations contains custom annotations used inside GeckoView and Fennec.
  - app is Fennec - Firefox for Android. Here is where you will find code specific to that app.
  - geckoview is the GeckoView project. Here is all the Java files related to GeckoView
  - geckoview_example is an example browser built using GeckoView.
  - omnijar contains the parts of Gecko and GeckoView that are not written in Java or Kotlin
  - thirdparty contains third party code that Fennec and GeckoView use.

  ![alt text](assets/GeckoViewStructure.png "GeckoView Structure")

Now you're set up and ready to go.

One you have got GeckoView building and running, you will want to start contributing. There is a general guide to [Performing a Bug Fix for Git Developers](ContributingToMC.md) for you to follow. To contribute to GeckoView specifically, you will need the following additional information.

It is advisable to run your tests before submitting your patch. You can do this using Mozilla's *try* server. To submit a GeckoView patch to *try* before submitting it for review, type:

```
./mach try fuzzy -q "android"
```

This will run all of the Android test suite. If your patch passes on *try* you can be (fairly) confident that it will land successfully after review.

When submitting a patch to Phabricator, if you know who you want to review your patch, put their Phabricator handle against the *reviewers* field.

If you don't know who to tag for a review in the Phabricator submission message, leave the field blank and, after submission, follow the link to the patch in Phabricator and scroll to the bottom of the screen until you see the comment box.

- Select the *Add Action* drop down and pick the *Change Reviewers* option.
- In the presented box, add *geckoview-reviewers*. Selecting this group as the reviewer will notify all the members of the GeckoView team there is a patch to review.
- Click *Submit* to submit the reviewer change request.

If you want to include a development version of GeckoView as a dependency inside another app, you must link to a local copy. There are two ways of doing this, publishing GeckoView to a local Maven repository (recommended), or linking to a local archive/

Publish GeckoView to your local maven by running

```
./gradlew geckoview:publishWithGeckoBinariesDebugPublicationToMavenLocal
```

- The binary will have been published to a repo found in *~/.m2*. Run the following command to figure out the name of the artifcat:

```
tree ~/.m2/repository/org/mozilla/geckoview
```

- Make a note of the name of your artifact. Update your gradle file to point to the dependency and link to your local repository.

```
dependencies {
    // ...
    armImplementation "geckoview-nightly-armeabi-v7a-65.0.20181128102620"
    // ...
}

// ...

repositories {
  //...
  mavenLocal()
  //..
}
```

```
./mach android archive-geckoview
```

This should create a file named geckoview-*.aar in your build output folder (MOZ_OBJDIR):

```
ls <your-output-directory>/gradle/build/mobile/android/geckoview/outputs/aar
geckoview-official-withGeckoBinaries-noMinApi-release.aar
```

Then all you need to do is point to the AAR in your gradle file.

```
repositories {
    // ...

    flatDir(
        name: 'localBuild',
        dirs: '<absolute path to AAR>'
    )
}
// ...
dependencies {
    // ...

    // armImplementation "org.mozilla:geckoview-nightly-armeabi-v7a:60.0a1"
    armImplementation (
            name: 'geckoview-official-withGeckoBinaries-noMinApi-release',
            ext: 'aar'
    )
    x86Implementation "org.mozilla:geckoview-nightly-x86:60.0a1"

    // ...
}
```

CHAPTER 2

Submitting a patch to Firefox using Git.

This guide will take you through submitting and updating a patch to *mozilla-central* as a git user. You need to already be [set up to use git to contribute to *mozilla-central*](MozCentralQuickStart.md).

All of the open bugs for issues in Firefox can be found in [Bugzilla](https://bugzilla.mozilla.org). If you know the component that you wish to contribute to you can use Bugzilla to search for issues in that project. If you are unsure which component you are interested in, you can search the [Good First Bugs](https://bugzilla.mozilla.org/buglist.cgi?quicksearch=good-first-bug) list to find something you want to work on.

- Once you have your bug, assign it to yourself in Bugzilla.
- Update your local copy of the firefox codebase to match the current version on the servers to ensure you are working with the most up to date code.

`` `bash git remote update ` `` * Create a new feature branch tracking either Central or Inbound.

`` ` git checkout -b bugxxxxxxx [inbound|central]/default ` `` * Work on your bug, checking into git according to your preferred workflow. _Try to ensure that each individual commit compiles and passes all of the tests for your component. This will make it easier to land if you use *moz-phab* to submit (details later in this post)._

You must have Mozilla commit access in order to submit your fix directly to the Firefox repository. There are three levels of commit access that give increasing levels of access to the repositories.

Level 1: Try/User access. You will need this level of access commit to the try server. Level 2: General access. This will give you full commit access to any mercurial or SVN repository not requiring level 3 access. Level 3: Core access. You will need this level to commit to any of the core repositories (Firefox/Thunderbird/Fennec).

If you wish to apply for commit access, please follow the guide found in the [Mozilla Commit Access Policy](https://www.mozilla.org/en-US/about/governance/policies/commit/access-policy/).

If you do not have access and still want to submit your fix, you can create a patch and attach it directly to the Bugzilla ticket. We do not recommend this method, though, and strongly encourage contributors to apply for commit access.

### Attaching a patch in Bugzilla

- Create a patch from your commits.

`` `bash git format-patch -[number of commits to include] ` `` This will create a patch for each commit. If you want to roll your commits into a single patch, you can use the following command.

`` `bash git format-patch -[number of commits] --stdout > bugxxxxxxx-ddmmyyyy.patch ` `` * Visit your bug in Bugzilla

- Click on the *Attach File* link
- Drag or your patch file into the *File* box.
- Add a brief description of the patch (this is especially useful if you are submitting multiple patches)
- Check the *patch* checkbox to indicate that this is a patch.
- Select the *?* from the *review* checkbox.
- Add the bugzilla handle for the person you want to review in the associated text box that will appear. If you do not know who should review the patch, select an option from the *Suggested Reviewers* drop down.
- Add any comments that you want to make about the patch in the *Comments* box. This is where I would add the kind of message I would add to the description of a PR in Github.
- Submit the form.

If you only have Level 1 access, you will still need to attach your patch to the Bugzilla bug, but you can test it on the try server first.

- Create a commit using the [try syntax](https://wiki.mozilla.org/ReleaseEngineering/TryChooser)
- Push to the try server

`` `bash git push try ` ``

To commit anything to the repository, you will need to set up Arcanist and Phabricator. If you are using *git-cinnabar* then you will need to use git enabled versions of these tools.

- Ensure PHP is installed
- [Install Arcanist](https://secure.phabricator.com/book/phabricator/article/arcanist_quick_start/)
- In a browser, visit Mozilla's Phabricator instance at https://phabricator.services.mozilla.com/.
- Click "Log In" at the top of the page

  ![alt text](assets/LogInPhab.png "Log in to Phabricator")
- Click the "Log In or Register" button on the next page. This will take you to Bugzilla to log in or register a new account.

  ![alt text](assets/LogInOrRegister.png "Log in or register a Phabiricator account")
- Sign in with your Bugzilla credentials, or create a new account.

  ![alt text](assets/LogInBugzilla.png "Log in with Bugzilla")
- You will be redirected back to Phabricator, where you will have to create a new Phabricator account. <Screenshot Needed>
- Fill in/amend any fields on the form and click "Register Account". <Screenshot Needed>
- You now have a Phabricator account and can submit and review patches.
- Ensure you are on the branch where you have commits that you want to submit.

`` `bash git checkout "your-branch-with-commits" ` `` * Create a differential patch containing your commits

`` `bash arc diff ` ``

- If you have any uncommited files, Arcanist will ask if you want to commit them.
- If you have any files in the path not added to git Arcanist will ask if you want to ignore them.
- After formatting your patch, Arcanist will open a nano/emacs file for you to enter the commit details. If you have many individual git commits in your arcanist diff then the first line of the first commit message will become the patch title, and the rest of the commit, plus the messages for the other commits in the patch will form the summary.
- Ensure you have entered the bug number against the *Bug #* field.
- If you know who you want to review your patch, put their Phabricator handle against the *reviewers* field. If in doubt, look to see who filed, or is listed as a mentor on, the bug you are addressing and choose them.
- Close the editor (Ctrl X) to save the patch.
- Arcanist now formats your patch and submits it to Phabricator. It will display the Phabricator link in the output.
- Copy that link and paste it into a browser window to view your patch.

You may have noticed when using Arcanist that it wraps all of your carefully curated Github commits into a single patch. If you have made many commits that are self contained and pass all the tests then you may wish to submit a patch for each commit. This will make it easier to review. The way to do this is via *moz-phab*. *moz-phab* required Arcanist so you do have to have that installed first.

N.B. If each individual patch does not compile and pass tests you will not be able to land each patch individually. In this case, please use Arcanist.

- Download the latest version of [*moz-phab*](https://github.com/mozilla-conduit/review/releases/tags) from the repository.
- Add it to your path

`` `bash export PATH="$PATH:/somewhere/moz-phab/bin/" echo PATH="$PATH:/somewhere/moz-phab/bin/" >> ~/.bash_profile ` ``

- Ensure you are on the branch where you have commits that you want to submit.

`` `bash git checkout your-branch ` `` * Check the revision numbers for the commits you want to submit

`` `bash git log ` `` * Run *moz-phab*. Specifying a start commit will submit all commits from that commit. Specifying an end commit will submit all commits up to that commit. If no positional arguments are provided, the range is determined to be starting with the first non-public, non-obsolete changeset (for Mercurial) and ending with the currently checked-out changeset.

`` `bash moz-phab submit [start_rev] [end_rev] ` `` * You will recieve a Phabricator link for each commit in the set.

- Often you will need to make amendments to a patch after it has been submitted to address review comments. To do this, add your commits to the base branch of your fix as normal.

To submit the update using Arcanist, run *arc diff –update <PhabricatorDifferentialNumber>*.

For *moz-phab* run in the same way as the initial submission with the same arguments, that is, specifying the full original range of commits. Note that, while inserting and amending commits should work fine, reordering commits is not yet supported, and deleting commits will leave the associated revisions open, which should be abandoned manually

CHAPTER 3

Firefox Developer Git Quick Start Guide

Getting setup to as a first time Mozilla contributor is hard. There are plenty of guides out there to help you get started as a contributor, but many of the new contributor guides out of date often more current ones are aimed at more experienced contributors. If you want to review these guides, you can find several linked to from [Contributing to the Mozilla code base](https://developer.mozilla.org/docs/Mozilla/Developer_guide/Introduction) on [MDN](https://developer.mozilla.org/).

This guide will take you through setting up as a contributor to *mozilla-central*, the Firefox main repository, as a git user.

The first thing you will need is to install Mercurial as this is the VCS that *mozilla-central* uses.

```
brew install mercurial
```

```
sudo port install mercurial
```

```
sudo apt-get install mercurial
```

Alternatively you can install [Mercurial directly](https://www.mercurial-scm.org/wiki/Download).

Check that you have successfully installed Mercurial by running:

`` `bash hg --version ` ``

If you are an experienced git user and are unfamiliar with Mercurial, you may want to install *git-cinnabar*. Cinnabar is a git remote helper that allows you to interact with Mercurial repos using git semantics.

There is a Homebrew install option for *git-cinnabar*, but this did not work for me, nor did the installer option. Using these tools, when I tried to clone the Mercurial repo it hung and did not complete. I had to do a manual install before I could use *git-cinnabar* successfully to download a Mercurial repo. If you would like to try either of these option, however, here they are:

`` `bash brew install git-cinnabar ` ``

`` `bash git cinnabar download ` ``

```
`bash git clone https://github.com/glandium/git-cinnabar.git && cd
git-cinnabar make export PATH="$PATH:/somewhere/git-cinnabar/" echo
```

`PATH="$PATH:/somewhere/git-cinnabar/" >> ~/.bash_profile export PATH="$PATH:/somewhere/git-cinnabar/git-core/bin-wrappers" echo PATH="$PATH:/somewhere/git-cinnabar/git-core/bin-wrappers" >> ~/.bash_profile ``

git-cinnabar's creator, [glandium](https://glandium.org/), has written a number of posts about setting up for Firefox Development with git. This [post](https://glandium.org/blog/?page_id=3438) is the one that has formed the basis for this walkthrough.

In synopsis:

- initialize an empty git repository

`` `bash git init gecko && cd gecko ` ``

- Configure git:

`` `bash git config fetch.prune true git config push.default upstream ` ``

- Add remotes for your repositories. There are several to choose from, *central*, *inbound*, *beta*, *release* etc. but in reality, if you plan on using Phabricator, which is Firefox's preferred patch submission system, you only need to set up *central*. It might be advisable to have access to *inbound* however, if you want to work on a version of Firefox that is queued for release. This guide will be focussed on Phabricator.

`` `bash git remote add central hg::https://hg.mozilla.org/mozilla-central -t branches/default/tip git remote add inbound hg::https://hg.mozilla.org/integration/mozilla-inbound -t branches/default/tip git remote set-url --push central hg::ssh://hg.mozilla.org/mozilla-central git remote set-url --push inbound hg::ssh://hg.mozilla.org/integration/mozilla-inbound ` ``

- Expose the branch tip to get quick access with some easy names.

`` `bash git config remote.central.fetch +refs/heads/branches/default/tip:refs/remotes/central/default git config remote.inbound.fetch +refs/heads/branches/default/tip:refs/remotes/inbound/default ` ``

- Setup a remote for the try server. The try server is an easy way to test a patch without actually checking the patch into the core repository. Your code will go through the same tests as a *mozilla-central* push, and you'll be able to download builds if you wish.

`` `bash git remote add try hg::https://hg.mozilla.org/try git config remote.try.skipDefaultUpdate true git remote set-url --push try hg::ssh://hg.mozilla.org/try git config remote.try.push +HEAD:refs/heads/branches/default/tip ` ``

- Now update all the remotes. This performs a *git fetch* on all the remotes. Mozilla Central is a _large_ repository. Be prepared for this to take a very long time.

`` `bash git remote update ` ``

All that's left to do now is pick a bug to fix and [submit a patch](ContributingToMC.md).

CHAPTER 4

Indices and tables

- genindex
- modindex
- search