Flutter Conductor

Command-line tool for managing a release of the Flutter SDK. Also see https://github.com/flutter/flutter/wiki/Release-process for more information on the release process.

Requirements

Some basic requirements to conduct a release are:

- a Linux or macOS computer set up for Flutter development. The conductor does not support Windows.
- git
- Mirrors on GitHub of the Flutter framework and engine repositories.

For the best experience, it is recommended to use ssh protocol for connecting to GitHub remote repositories (i.e. for `--framework-mirror` and `--engine-mirror` specify the remote with the format `git@github.com:username/reponame`). If your local ssh key is password-protected, it is recommended to use ssh-agent to unlock your ssh key for the session; if you do not, each time the conductor attempts to interact with a remote, the user will be prompted to enter their ssh key password.

Usage

The main entrypoint for the conductor is bin/conductor. For brevity, the rest of this document will assume that this entrypoint is on the shell path.

All available commands can be seen via:

```
conductor help
```

Releases are initialized with the `start` sub-command, like:

```
conductor start \
  --candidate-branch=flutter-2.2-candidate.10 \
  --release-channel=beta \
  --framework-mirror=git@github.com:username/flutter.git \
  --engine-mirror=git@github.com:username/engine.git \
  --engine-cherrypicks=72114dafe28c8700f1d5d629c6ae9d34172ba395 \
  --framework-
cherrypicks=a3e66b396746f6581b2b7efd1b0d0f0074215128,d8d853436206e86f416236b930e97779b143a100
 \
  --dart-revision=4511eb2a779a612d9d6b2012123575013e0aef12 \
```

The conductor will, based on the release channel and the presence/lack of previous tags, determine which part of the release version should be incremented. In the cases where this is not correct, the version can be overridden with `--version-override=3.0.0`.

For more details on these command line arguments, see `conductor help start`. This command will write to disk a state file that will persist until the release is completed. If you already have a persistent state file, this command will fail with an error message. To see the current status of the release (at any time), issue the command:

```
conductor status
```

To delete a persistent state file (either because the release was successfully completed or abandoned), use the command:

```
conductor clean
```

Once initializing the release, the conductor tool will issue instructions for manual steps that must be executed by the user. At any time these instructions can be seen via `conductor status`. Once these manual steps have been completed, you can

proceed to the next step by using the command:

`conductor next`

Steps

Once the user has finished manual steps for each step, they proceed to the next step with the command:

`conductor next`

Apply Engine Cherrypicks

The tool will attempt to auto-apply all engine cherrypicks. However, any cherrypicks that result in a merge conflict will be reverted and it is left to the user to manually cherry-pick them (with the command `git cherry-pick $REVISION`) and resolve the merge conflict in their checkout.

Once a PR is opened, the user must validate CI builds. If there are regressions (or if the `licenses_check` fails, then `//engine/ci/licenses_golden/licenses_third_party` must be updated to match the output of the failing test), then the user must fix these tests in their local checkout and push their changes again.

Codesign Engine Binaries

The user must validate post-submit CI builds for their merged engine PR have passed. A link to the web dashboard is available via `conductor status`. Once the post-submit CI builds have all passed, the user must codesign engine binaries for the **merged** engine commit.

Apply Framework Cherrypicks

The tool will attempt to auto-apply all framework cherrypicks. However, any cherrypicks that result in a merge conflict will be reverted and it is left to the user to manually cherry-pick them (with the command `git cherry-pick $REVISION`) and resolve the merge conflict in their checkout.

Publish Version

This step will add a version git tag to the final Framework commit and push it to the upstream repository. The presence of a tag affects what the flutter CLI tool reports the current version is.

Publish Channel

This step will push the Framework candidate branch to the upstream release branch (e.g. the `stable` branch). Once this push happens upstream, the release has officially been published, and the code will be available to existing Flutter users via `flutter upgrade`.

Verify Release

For the final step, the user must manually verify that packaging builds have finished successfully. The SDK compressed archives will not be available from the website until the packaging build has finished. The conductor will produce links to the dashboards for monitoring CI builds.