Development

This document is intended to provide instructions and helpful information for developers who are contributing pull-requests (or otherwise making changes) to **Meteor Core itself (not Meteor apps)**.

As the first suggestion to the reader of this document: If, during the course of development, a Meteor-specific process is revealed which is helpful and not documented here, please consider editing this document and submitting a pull-request. Another developer will be thankful!

Running from a Git checkout

If you want to run on the bleeding edge, or help contribute to Meteor, you can run Meteor directly from a Git checkout using these steps:

1. **Clone from GitHub**

   ```
   $ git clone --recursive https://github.com/meteor/meteor.git
   $ cd meteor
   ```

   > **Important note about Git submodules!** This repository uses Git submodules. If you clone without the `--recursive` flag, re-fetch with `git pull` or experience "`Depending on unknown package`" errors, run the following in the repository root to sync things up again:
   >
   > ```
   > $ git submodule update --init --recursive
   > ```

2. **Run a Meteor command to install dependencies**

   If you did not compile dependencies above, this will also download the binaries.

   ```
   $ ./meteor --help
   ```

3. **Ready to Go!**

   Your local Meteor checkout is now ready to use! You can use this `./meteor` anywhere you would normally call the system `meteor`. For example,:

   ```
   $ cd my-app/
   $ /path/to/meteor-checkout/meteor run
   ```

   > *Tip:* Consider making an easy-to-run alias for frequent use:
   >
   > ```
   > alias mymeteor=/path/to-meteor-checkout/meteor
   > ```
   >
   > This allows the use of `mymeteor` in place of `meteor`. To persist this across shell logouts, simply add it to `~/.bashrc` or `.zshrc`.

Notes when running from a checkout

The following are some distinct differences you must pay attention to when running Meteor from a checkout:

- You cannot pin apps to specific Meteor releases or change the release using `--release`.

The "Dev Bundle"

The "dev bundle" (identified as the `dev_bundle` in the folder structure) is a generated bundle of code, packages and tools which are essential to providing the functionality of the Meteor tool (`meteor`) and the app bundles which it builds.

When `meteor` is run from a checkout, a `dev_bundle` is automatically downloaded and should be sufficient for most development. However, some more substantial changes will require rebuilding the `dev_bundle`. This include changes to the:

- Node.js version
- npm version
- MongoDB version
- TypeScript version
- Packages used by `meteor-tool`
- Packages used by the server bundle

While it may be tempting to make changes to these variables, please consider the repercussions (including compatibility and stability) and make sure to test changes extensively. For example, major version changes (especially to Node.js and MongoDB) usually require substantial changes to other components.

"Dev Bundle" versions

The working version number of the `dev_bundle` to be downloaded (or generated) is stored as `BUNDLE_VERSION` at the top of the `meteor` script. When submitting a pull request which changes components of the `dev_bundle`, the minor version should be bumped (at the very least). In local development, it is advisable to use a different major version (e.g. `100.0.0`) so as not to clash with the official versions which are cached locally.

To enable caching of downloaded `dev_bundle` versions, set the `SAVE_DEV_BUNDLE_TARBALL` environment variable before running Meteor, for example:

```
SAVE_DEV_BUNDLE_TARBALL=1 ./meteor
```

Cached versions of the `dev_bundle` are stored in the root directory of the checkout. Keeping them around will prevent the need to re-download them when switching between branches, but they do become quite large as they collect, so delete them as necessary!

Rebuilding the "Dev Bundle"

Rebuilding requires a C and C++ compiler, `autotools`, and `scons`.

To build everything from scratch and re-package dependencies, simply run the following script:

```
$ ./scripts/generate-dev-bundle.sh
```

This will generate a new tarball (`dev_bundle_<Platform>_<arch>_<version>.tar.gz`) in the root of the checkout. Assuming you bumped the `BUNDLE_VERSION`, the new version will be extracted automatically when you run `./meteor`. If you are rebuilding the same version (or didn't bump the version number), you should delete the existing `dev_bundle` directory to ensure the new tarball is extracted when you run `./meteor`.

Submitting "Dev Bundle" Pull Requests

It's important to note that while `dev_bundle` pull requests are accepted/reviewed, a new `dev_bundle` can only be published to Meteor Software's Meteor infrastructure by a Meteor Software staff member. This means that the build tool and package tests of submitted `dev_bundle` pull requests will always initially fail (since the new `dev_bundle` hasn't yet been built/published by Meteor Software, which means it can't be downloaded by Meteor's continuous integration environment).

Pull requests that contain `dev_bundle` changes will be noted by repo collaborators, and a request to have a new `dev_bundle` built/published will be forwarded to Meteor Software.

Additional documentation

The Meteor core is best documented within the code itself, however, many components also have a `README.md` in their respective directories.

Some compartmentalized portions of Meteor are broken into packages (see a list of packages) and almost all of them have a `README.md` within their directory. For example, `ddp`, `ecmascript` and `tinytest`.

For the rest, try looking nearby for a `README.md`. For example, `isobuild` or `cordova`.

Tests

Test against the local meteor copy

When running any tests, be sure to run them against the checked-out copy of Meteor instead of the globally-installed version. This means ensuring that the command is `path-to-meteor-checkout/meteor` and not just `meteor`.

This is important so that tests are run against your local development version and not the stable (installed) Meteor release.

Running tests on Meteor core

When you are working with code in the core Meteor packages, you will want to make sure you run the full test-suite (including the tests you added) to ensure you haven't broken anything in Meteor. The `test-packages` command will do just that for you:

```
./meteor test-packages
```

Exactly in the same way that `test-packages` works in standalone Meteor apps, the `test-packages` command will start up a Meteor app with TinyTest. To view the results, just connect to `http://localhost:3000`.

If you want to see results in the console you can use:

```
PUPPETEER_DOWNLOAD_PATH=~/.npm/chromium ./packages/test-in-console/run.sh
```

> PUPPETEER_DOWNLOAD_PATH is optional but this is useful to skip Downloading Chromium on every run
>
> We run our tests on Travis like above.

Running specific tests Specific package tests can be run by passing a `<package name>` or `<package path>` to the `test-packages` command. For example, to run `mongo` tests, it's possible to run:

```
./meteor test-packages mongo
```

For more fine-grained control, if you're interested in running only the specific tests that relate to the functionality you're working on, you can filter individual tests by using the `TINYTEST_FILTER` environment variable (which supports regex's). For example, to run only the package tests that verify `new Mongo.Collection` behavior, try:

```
TINYTEST_FILTER="collection - call new Mongo.Collection" ./meteor test-packages
```

You can also provide the same filters for `./packages/test-in-console/run.sh` explained above.

Running Meteor Tool self-tests

While TinyTest and the `test-packages` command can be used to test internal Meteor packages, they cannot be used to test the Meteor Tool itself. The Meteor Tool is a node app that uses a home-grown "self test" system.

Listing available tests To see a list of tests included in the self-test system, use the `--list` option:

```
./meteor self-test --list
```

Running specific tests The self-test commands support a regular-expression syntax in order to specific/search for specific tests. For example, to search for tests starting with `a` or `b`, it's possible to run:

```
./meteor self-test "^[a-b]" --list
```

Simply remove the `--list` flag to actually run the matching tests.

Excluding specific tests In a similar way to the method of specifying which tests TO run, there is a way to specify which tests should NOT run. Again, using regular-expressions, this command will NOT list any tests which start with `a` or `b`:

```
./meteor self-test --exclude "^[a-b]" --list
```

Simply remove the `--list` flag to actually run the matching tests.

Avoiding retries On CI we want to retry the tests to avoid false failures but in development can take some time if you retry every time a test is failing. So to avoid retries use:

```
./meteor self-test --retries 0
```

More reading For even more details on how to run Meteor Tool "self tests", please refer to the Testing section of the Meteor Tool README.

Continuous integration

Any time a pull-request is submitted or a commit is pushed directly to the `devel` branch, continuous integration tests will be started automatically by the CI server. These are run by Circle CI and defined in the `circle.yml` file. Even more specifically, the tests to run and the containers to run them under are defined in the `/scripts/ci.sh` script, which is a script which can run locally to replicate the exact tests.

Not every test which is defined in a test spec is actually ran by the CI server. Some tests are simply too long-running and some tests are just no longer relevant. As one particular example, there is a suite of very slow tests grouped into a `slow` designator within the test framework. These can be executed by adding the `--slow` option to the `self-test` command.

> Please Note: Windows
>
> There is not currently a continuous integration system setup for Windows. Additionally, not all tests are known to work on Windows. If you're able to take time to improve those tests, it would be greatly appreciated. Currently, there isn't an official list of known tests

which do not run on Windows, but a PR to note those here and get them fixed would be ideal!

Running your own CircleCI Since Meteor is a free, open-source project, you can run tests in the context of your own CircleCI account at no cost (up to the maximum number of containers allowed by them) during development and prior to submitting a pull-request. For some, this may be quicker or more convenient than running tests on their own workstation. As an added advantage, when your tests are "green", that status will be immediately shown (as passing) when a pull-request is opened with the official Meteor repository.

To enable CircleCI for your development:

0. Make sure you have an account with CircleCI
1. Make sure you have forked Meteor into your own GitHub account.
2. Go to the Add Projects page on CircleCI.
3. On the left, click on your GitHub username.
4. On the right, find `meteor`
5. Click on the "Build project" button next to `meteor`.
6. Your build will start automatically!

Code style

- New contributions should follow the Meteor Style Guide as closely as possible.
  - The Meteor Style Guide is very close to the Airbnb Style Guide with a few notable changes.
- New code should match existing code (in the same vicinity) when the context of a change is minimal, but larger amounts of new code should follow the guide.
- Do not change code that doesn't directly relate to the feature/bug that you're working on.
- Basic linting is accomplished (via ESLint) by running `./scripts/admin/eslint/eslint.sh`.
  - Many files have not been converted yet and are thus excluded.

Commit messages

Good commit messages are very important and you should make sure to explain what is changing and why. The commit message should include:

- A short and helpful commit title (maximum 80 characters).
- A commit description which clearly explains the change if it's not super-obvious by the title. Some description always helps!
- Reference related issues and pull-requests by number in the description body (e.g. "#9999").
- Add "Fixes" before the issue number if the addition of that commit fully resolves the issue.