Building and Testing Angular

This document describes how to set up your development environment to build and test Angular. It also explains the basic mechanics of using `git`, `node`, and `yarn`.

- Prerequisite Software
- Getting the Sources
- Installing NPM Modules
- Building
- Running Tests Locally
- Formatting your Source Code
- Linting/verifying your Source Code
- Publishing Snapshot Builds
- Bazel Support

See the contribution guidelines if you'd like to contribute to Angular.

Prerequisite Software

Before you can build and test Angular, you must install and configure the following products on your development machine:

- Git and/or the **GitHub app** (for Mac and Windows); GitHub's Guide to Installing Git is a good source of information.
- Node.js, (version specified in the engines field of `package.json`) which is used to run a development web server, run tests, and generate distributable files.
- Yarn (version specified in the engines field of `package.json`) which is used to install dependencies.
- Optional: Java version 7 or higher as required by Closure Compiler. Most developers will not need this. Java is required for running some of the integration tests.

Getting the Sources

Fork and clone the Angular repository:

1. Login to your GitHub account or create one by following the instructions given here.
2. Fork the main Angular repository.
3. Clone your fork of the Angular repository and define an `upstream` remote pointing back to the Angular repository that you forked in the first place.

```
# Clone your GitHub repository:
git clone git@github.com:<github username>/angular.git
```

```
# Go to the Angular directory:
cd angular

# Add the main Angular repository as an upstream remote to your repository:
git remote add upstream https://github.com/angular/angular.git
```

Installing NPM Modules

Next, install the JavaScript modules needed to build and test Angular:

```
# Install Angular project dependencies (package.json)
yarn install
```

Building

To build Angular run:

```
node ./scripts/build/build-packages-dist.js
```

- Results are put in the `dist/packages-dist` folder.

Running Tests Locally

Bazel is used as the primary tool for building and testing Angular. Building and testing are incremental with Bazel, and it's possible to only run tests for an individual package instead of for all packages. Read more about this in the BAZEL.md document.

You should execute all test suites before submitting a PR to GitHub. - `yarn test //packages/...`

Note: The ellipsis in the commands above is not meant to be substituted by a package name, but is used by Bazel as a wildcard to execute all tests in the specified path. To execute tests for a single package, the commands are (exemplary): - `yarn test //packages/core/...` for all tests, or - `yarn test //packages/core/test:test_web_firefox` for a particular test suite.

Note: The first test run will be much slower than future runs. This is because future runs will benefit from Bazel's capability to do incremental builds.

All the tests are executed on our Continuous Integration infrastructure. PRs can only be merged if the code is formatted properly and all tests are passing.

Testing changes against a local library/project

Often for developers the best way to ensure the changes they have made work as expected is to run use changes in another library or project. To do this developers can build Angular locally, and using `yarn link` build a local project with the created artifacts.

This can be done by running:

```
yarn ng-dev misc build-and-link <path-to-local-project-root>
```

Formatting your source code

Angular uses clang-format to format the source code. If the source code is not properly formatted, the CI will fail and the PR cannot be merged.

You can automatically format your code by running: - `yarn ng-dev format changed [shaOrRef]`: format only files changed since the provided sha/ref. `shaOrRef` defaults to `master`. - `yarn ng-dev format all`: format *all* source code - `yarn ng-dev format files <files..>`: format only provided files

A better way is to set up your IDE to format the changed file on each file save.

VS Code

1. Install Clang-Format extension for VS Code.
2. It will automatically pick up the settings from `.vscode/settings.json`. If you haven't already, create a `settings.json` file by following the instructions here.

WebStorm / IntelliJ

1. Install the ClangFormatIJ plugin
2. Open `Preferences->Tools->clang-format`
3. Find the field named "PATH"
4. Add `<PATH_TO_YOUR_WORKSPACE>/angular/node_modules/clang-format/bin/<OS>/` where the OS options are: `darwin_x64`, `linux_x64`, and `win32`.

Vim

1. Install Vim Clang-Format.
2. Create a project-specific `.vimrc` in your Angular directory containing

```
let g:clang_format#command = '$ANGULAR_PATH/node_modules/.bin/clang-format'
```

where `$ANGULAR_PATH` is an environment variable of the absolute path of your Angular directory.

Linting/verifying your Source Code

You can check that your code is properly formatted and adheres to coding style by running:

```
$ yarn lint
```

Publishing Snapshot Builds

When a build of any branch on the upstream fork angular/angular is green on CircleCI, it automatically publishes build artifacts to repositories in the Angular org, eg. the `@angular/core` package is published to https://github.com/angular/core-builds.

You may find that your un-merged change needs some validation from external participants. Rather than requiring them to pull your Pull Request and build Angular locally, they can depend on snapshots of the Angular packages created based on the code in the Pull Request.

Getting Packages from Build Artifacts

Each CI run for a Pull Request stores the built Angular packages as build artifacts. The artifacts are not guaranteed to be available as a long-term distribution mechanism, but they are guaranteed to be available around the time of the build.

You can access the artifacts for a specific CI run by going to the workflow page, clicking on the `publish_packages_as_artifacts` job and then switching to the "ARTIFACTS" tab.

Archives for each Package On the "Artifacts" tab, there is a list of links to compressed archives for Angular packages. The archive names are of the format `<package-name>-pr<pr-number>-<sha>.tgz` (for example `core-pr12345-a1b2c3d.tgz`).

One can use the URL to the `.tgz` file for each package to install them as dependencies in a project they need to test the Pull Request changes against. Both npm and yarn support installing dependencies from URLs to `.tgz` files, for example by updating the dependencies in `package.json` to point to the artifact URLs and then running `npm/yarn install`:

```
"dependencies": {
  "@angular/common": "https://<...>.circle-artifacts.com/0/angular/common-pr12345-a1b2c3d.tg
  "@angular/core": "https://<...>.circle-artifacts.com/0/angular/core-pr12345-a1b2c3d.tgz",
  "...": "..."
}
```

Download all Packages In addition to the individual package archives, a `.tgz` file including all packages is also available (named `all-pr<pr-number>-<sha>.tgz`). This can be used if one prefers to download all packages locally and test them by either of the following ways:

1. Update the dependencies in `package.json` to point to the local uncompressed package directories.

2. Directly copy the local uncompressed package directories into the `node_modules/` directory of a project.

Note that (while faster) the second approach has limitations. For example: a. Any transitive dependencies of the copied packages will not be automatically updated. b. The packages need to be copied over every time `npm/yarn install` is run. c. Some package managers (such as `pnpm` or `yarn pnp`) might not work correctly.

Publishing to GitHub Repos

You can also manually publish `*-builds` snapshots just like our CircleCI build does for upstream builds. Before being able to publish the packages, you need to build them locally by running the `./scripts/build/build-packages-dist.js` script.

First time, you need to create the GitHub repositories:

```
$ export TOKEN=[get one from https://github.com/settings/tokens]
$ CREATE_REPOS=1 ./scripts/ci/publish-build-artifacts.sh [GitHub username]
```

For subsequent snapshots, just run:

```
$ ./scripts/ci/publish-build-artifacts.sh [GitHub username]
```

The script will publish the build snapshot to a branch with the same name as your current branch, and create it if it doesn't exist.

Bazel Support

IDEs

VS Code

1. Install Bazel extension for VS Code.

WebStorm / IntelliJ

1. Install the Bazel plugin
2. You can find the settings under `Preferences->Other Settings->Bazel Settings`

It will automatically recognize `*.bazel` and `*.bzl` files.

Remote Build Execution and Remote Caching

Bazel builds in the Angular repository use a shared cache. When a build occurs a hash of the inputs is computed and checked against available outputs in the shared cache. If an output is found, it is used as the output for the build action rather than performing the build locally.

Remote Build Execution requires authentication as a google.com or angular.io account.

–config=remote flag The `--config=remote` flag can be added to enable remote execution of builds. This flag can be added to the `.bazelrc.user` file using the script at `scripts/local-dev/setup-rbe.sh`.