To run locally, do the usual:

1. Create a Python 3.5 virtualenv
2. Install dependencies:
3. pip install -r requirements/dev.txt
4. npm install

Alternatively use the make task:

make install

1. Make a directory to store the project's data (MEDIA\_ROOT, DOC\_BUILDS\_ROOT, etc.). We'll use ~/.djangoproject for example purposes.

Create a 'secrets.json' file in a directory named 'conf' in that directory, containing something like:

{ "secret\_key": "xyz",

"superfeedr\_creds": ["any@email.com", "some\_string"],

"db\_host": "localhost",

"db\_password": "secret",

"trac\_db\_host": "localhost",

"trac\_db\_password": "secret" }

Add export DJANGOPROJECT\_DATA\_DIR=~/.djangoproject (without the backticks) to your ~/.bashrc (or ~/.zshrc if you're using zsh) file and then run source ~/.bashrc (or source ~/.zshrc) to load the changes.

1. Create databases:
2. createuser -d djangoproject --superuser
3. createdb -O djangoproject djangoproject
4. createuser -d code.djangoproject --superuser
5. createdb -O code.djangoproject code.djangoproject
6. Setting up database access

If you are using the default postgres configuration, chances are you will have to give a password for the newly created users in order to be able to use them for Django:

psql

ALTER USER djangoproject WITH PASSWORD 'secret';

ALTER USER "code.djangoproject" WITH PASSWORD 'secret';

\d

(Use the same passwords as the ones you've used in your secrets.json file)

1. Create tables:
2. psql -d code.djangoproject < tracdb/trac.sql
3. ./manage.py migrate
4. Create a superuser:
5. ./manage.py createsuperuser
6. Populate the www and docs hostnames in the django.contrib.sites app:
7. ./manage.py loaddata dev\_sites
8. For docs:
9. ./manage.py loaddata doc\_releases
10. ./manage.py update\_docs
11. For dashboard:

To load the latest dashboard categories and metrics:

./manage.py loaddata dashboard\_production\_metrics

Alternatively, to load a full set of sample data (takes a few minutes):

./manage.py loaddata dashboard\_example\_data

Finally, make sure the loaded metrics have at least one data point (this makes API calls to the URLs from the metrics objects loaded above and may take some time depending on the metrics chosen):

./manage.py update\_metrics

1. Point the www.djangoproject.localhost, docs.djangoproject.localhost, and dashboard.djangoproject.localhost hostnames with your /etc/hosts file to localhost/127.0.0.1:
2. 127.0.0.1 docs.djangoproject.localhost www.djangoproject.localhost dashboard.djangoproject.localhost

This is unnecessary with some browsers (e.g. Opera and Chromium/Chrome) as they handle localhost subdomains automatically.

If you're on Mac OS and don't feel like editing the /etc/hosts file manually, there is a great preference pane called [Hosts.prefpane](https://github.com/specialunderwear/Hosts.prefpane). On Ubuntu there is a [built-in network admin](https://help.ubuntu.com/community/NetworkAdmin) GUI to do the same. Remember both require admin privileges, just like you'd need when editing /etc/hosts with your favorite editor.

If you don't have admin rights but have an internet connection, you can use a service like [xip.io](http://xip.io). In that case you'll also have to update ALLOWED\_HOSTS in djangoproject/settings/dev.py as well as the content of the django\_site table in your database.

1. Compile the CSS (only the source SCSS files are stored in the repository):
2. make compile-scss
3. Finally run the server:
4. make run

This runs both the main site ("www") as well as the docs and dashboard site in the same process. Open <http://www.djangoproject.localhost:8000/>, <http://docs.djangoproject.localhost:8000/>, or <http://dashboard.djangoproject.localhost:8000/>.

**Running the tests**

We use [Travis-CI](https://travis-ci.org/) for continuous testing and [GitHub](https://github.com/) pull request integration. If you're familiar with those systems you should not have any problems writing tests.

Our test results can be found here:

<https://travis-ci.org/django/djangoproject.com>

For local development don't hesitate to install [tox](https://tox.readthedocs.io/) to run the website's test suite.

Then in the root directory (next to the manage.py file) run:

tox

Behind the scenes this will run the usual ./manage.py test management command with a preset list of apps that we want to test as well as [flake8](https://flake8.readthedocs.io/) for code quality checks. We collect test coverage data as part of that tox run, to show the result simply run:

coverage report

or for a HTML-based report:

coverage html

**(Optional)** In case you're using an own virtualenv you can also run the tests manually using the test task of the Makefile. Don't forget to install the test requirements with the following command first though:

pip install -r requirements/tests.txt

Then run:

make test

or simply the usual test management command:

./manage.py test [list of app labels]

**Supported browsers**

The goal of the site is to target various levels of browsers, depending on their ability to use the technologies in use on the site, such as HTML5, CSS3, SVG, webfonts.

We're following [Mozilla's example](https://wiki.mozilla.org/Support/Browser_Support) when it comes to categorize browser support.

* Desktop browsers, except as noted below, are **A grade**, meaning that everything needs to work.
* IE < 11 is **not supported** (based on Microsoft's support).
* Mobile browsers should be considered **B grade** as well. Mobile Safari, Firefox on Android and the Android Browser should support the responsive styles as much as possible but some degredation can't be prevented due to the limited screen size and other platform restrictions.

**File locations**

Static files such as CSS, JavaScript or image files can be found in the djangoproject/static subdirectory.

Templates can be found in the djangoproject/templates subdirectory.

**Styles**

CSS is written in [Scss](http://sass-lang.com/) and compiled via [Libsass](http://libsass.org/).

Run the following to compile the Scss files to CSS:

make compile-scss-debug

Alternatively you can also run the following command in a separate shell to continuously watch for changes to the Scss files and automatically compile to CSS:

make watch-scss

**Running all at once**

Optionally you can use a tool like [Foreman](https://github.com/ddollar/foreman) to run all process at once:

* the site (similar to [www.djangoproject.com](http://www.djangoproject.com)) on <http://0.0.0.0:8000/> to be used with the modified /etc/hosts file (see above)
* the make task to automatically compile the SASS files to CSS files

This is great during development. Assuming you're using Foreman simply run:

foreman start

If you just want to run one of the processes defined above use the run subcommand like so:

foreman run web

That'll just run the www server.

Check out the Procfile file for all the process names.

**JavaScript libraries**

This project uses [Bower](https://bower.io/) to manage JavaScript libraries.

At any time, you can run it to install a new library (e.g., jquery-ui):

npm run bower install jquery-ui --save

or check if there are newer versions of the libraries that we use:

npm run bower ls

If you need to update an existing library, the easiest way is to change the version requirement in bower.json and then to run npm run bower install again.

We commit the libraries to the repository, so if you add, update, or remove a library from bower.json, you will need to commit the changes in djangoproject/static too.

**Documentation search**

When running ./manage.py update\_docs to build all documents it will also automatically index every document it builds in the search engine as well. In case you've already built the documents and would like to reindex the search index run the command:

./manage.py update\_index

This is also the right command to run when you work on the search feature itself. You can pass the -d option to try to drop the search index first before indexing all the documents.

**Updating metrics from production**

The business logic for dashboard metrics is edited via the admin interface and contained in the models in the dashboard app (other than Dataum, which contains the data itself). From time to time, those metrics should be extracted from a copy of the production database and saved to the dashboard/fixtures/dashboard\_production\_metrics.json file.

To update this file, run:

./manage.py dumpdata dashboard --exclude dashboard.Datum --indent=4 > dashboard\_production\_metrics.json

**Translation**

We're using Transifex to help manage the translation process. The requirements/dev.txt file will install the Transifex client.

Before using the command-line Transifex client, create ~/.transifexrc according to the instructions at <https://docs.transifex.com/client/client-configuration>. You'll need to be a member of the Django team in the [Django](https://www.transifex.com/django/) organization at Transifex. For information on how to join, please see the [Translations](https://docs.djangoproject.com/en/dev/internals/contributing/localizing/#translations) section of the documentation on contributing to and localizing Django.

Since this repo hosts three separate sites, our .po files are organized by website domain. At the moment, we have:

* dashboard/locale/ contains the translation files for <https://dashboard.djangoproject.com>
* docs/locale/ contains the translation files for <https://docs.djangoproject.com> (only for the strings in this repository; translation of the documentation itself is handled elsewhere)
* locale/ contains the translation files for <https://www.djangoproject.com> (including strings from all apps other than dashboard and docs)

**Important:** To keep this working properly, note that any templates for the dashboard and docs apps **must** be placed in the <app name>/templates/docs/ directory for their respective app, **not** in the djangoproject/templates/ directory.

**Updating messages on Transifex**

When there are changes to the messages in the code or templates, a member of the translations team will need to update Transifex as follows:

1. Regenerate the English (only) .po file:
2. python manage.py makemessages -l en

(Never update alternate language .po files using makemessages. We'll update the English file, upload it to Transifex, then later pull the .po files with translations down from Transifex.)

1. Push the updated source file to Transifex:
2. tx push -s
3. Commit and push the changes to github:
4. git commit -m "Updated messages" locale/en/LC\_MESSAGES/\*
5. git push

**Updating translations from Transifex**

Anytime translations on Transifex have been updated, someone should update our translation files as follows:

1. Review the translations in Transifex and add to the space-delimited LANGUAGES list in update-translations.sh any new languages that have reached 100% translation.
2. Pull the updated translation files:
3. ./update-translations.sh
4. Use git diff to see if any translations have actually changed. If not, you can just revert the .po file changes and stop here.
5. Compile the messages:
6. python manage.py compilemessages
7. Run the test suite one more time:
8. python manage.py test
9. Commit and push the changes to GitHub:
10. git commit -m "Updated translations" locale/\*/LC\_MESSAGES/\*
11. git push

**Running Locally with Docker**

1. Build the images:
2. docker-compose build
3. Spin up the containers:
4. docker-compose up
5. View the site at <http://localhost:8000/>
6. Run the tests:
7. docker-compose exec web tox
8. docker-compose exec web python manage.py test