# Contributor Guidelines

## Advice for new contributors

Start small. The PRs most likely to be merged are the ones that make small,

easily reviewed changes with clear and specific intentions. See below for more

[guidelines on pull requests](#pull-requests).

It's a good idea to gauge interest in your intended work by finding the current issue

for it or creating a new one yourself. You can use also that issue as a place to signal

your intentions and get feedback from the users most likely to appreciate your changes.

Once you've spent a little bit of time planning your solution, it's a good idea to go

back to the issue and talk about your approach. We'd be happy to provide feedback. [An

ounce of prevention, as they say!](https://www.goodreads.com/quotes/247269-an-ounce-of-prevention-is-worth-a-pound-of-cure)

## Developer Setup

First, you'll need [Node.js](https://nodejs.org/) which matches our current version.

You can check [`.nvmrc` in the `development` branch](https://github.com/signalapp/Signal-Desktop/blob/development/.nvmrc) to see what the current version is. If you have [nvm](https://github.com/creationix/nvm)

you can just run `nvm use` in the project directory and it will switch to the project's

desired Node.js version. [nvm for windows](https://github.com/coreybutler/nvm-windows) is

still useful, but it doesn't support `.nvmrc` files.

Then you need `git`, if you don't have that yet: https://git-scm.com/

### macOS

1. Install the [Xcode Command-Line Tools](http://osxdaily.com/2014/02/12/install-command-line-tools-mac-os-x/).

2. Ensure [git-lfs](https://github.com/git-lfs/git-lfs/wiki/Installation) is installed. You'll need it to to checkout and install the node requirements. Install with `brew install git-lfs`

### Windows

1. \*\*Windows 7 only:\*\*

- Install Microsoft .NET Framework 4.5.1:

https://www.microsoft.com/en-us/download/details.aspx?id=40773

- Install Windows SDK version 8.1: https://developer.microsoft.com/en-us/windows/downloads/sdk-archive

1. Install \_Windows Build Tools\_: Open the [Command Prompt (`cmd.exe`) as Administrator](<https://technet.microsoft.com/en-us/library/cc947813(v=ws.10).aspx>)

and run: `npm install --vs2015 --global --production --add-python-to-path windows-build-tools`

### Linux

1. Pick your favorite package manager.

1. Install `python`

1. Install `gcc`

1. Install `g++`

1. Install `make`

### All platforms

Now, run these commands in your preferred terminal in a good directory for development:

```

git clone https://github.com/signalapp/Signal-Desktop.git

cd Signal-Desktop

npm install --global yarn # (only if you don’t already have `yarn`)

yarn install --frozen-lockfile # Install and build dependencies (this will take a while)

yarn grunt # Generate final JS and CSS assets

yarn build:webpack # Build parts of the app that use webpack (Sticker Creator)

yarn test # A good idea to make sure tests run first

yarn start # Start Signal!

```

You'll need to restart the application regularly to see your changes, as there

is no automatic restart mechanism. Alternatively, keep the developer tools open

(`View > Toggle Developer Tools`), hover over them, and press

<kbd>Cmd</kbd> + <kbd>R</kbd> (macOS) or <kbd>Ctrl</kbd> + <kbd>R</kbd>

(Windows & Linux).

Also, note that the assets loaded by the application are not necessarily the same files

you’re touching. You may not see your changes until you run `yarn grunt` on the

command-line like you did during setup. You can make it easier on yourself by generating

the latest built assets when you change a file. Run this in its own terminal instance

while you make changes:

```

yarn grunt dev # runs until you stop it, re-generating built assets on file changes

```

### webpack

Some parts of the app (such as the Sticker Creator) have moved to webpack.

You can run a development server for these parts of the app with the

following command:

```

yarn dev

```

In order for the app to make requests to the development server you must set

the `SIGNAL\_ENABLE\_HTTP` environment variable to a truthy value. On Linux and

macOS, that simply looks like this:

```

SIGNAL\_ENABLE\_HTTP=1 yarn start

```

## Setting up standalone

By default the application will connect to the \*\*staging\*\* servers, which means that you

\*\*will not\*\* be able to link it with your primary mobile device.

Fear not! You don't have to link the app with your phone. On the QR code screen, you can

select 'Set Up as Standalone Device' from the File menu, which goes through the

registration process like you would on a phone.

Note: you won't be linked to a primary phone, which will make testing certain things very

difficult (contacts, profiles, and groups are all solely managed on your phone).

## The staging environment

Sadly, this default setup results in no contacts and no message history, an entirely

empty application. But you can use the information from your production install of Signal

Desktop to populate your testing application!

First, find your application data:

- macOS: `~/Library/Application Support/Signal`

- Linux: `~/.config/Signal`

- Windows 10: `C:\Users\<YourName>\AppData\Roaming\Signal`

Now make a copy of this production data directory in the same place, and call it

`Signal-development`. Now start up the development version of the app as normal,

and you'll see all of your contacts and messages!

You'll notice a prompt to re-link, because your production credentials won't work on

staging. Click 'Relink', then 'Standalone', then verify the phone number and click

'Send SMS.'

Once you've entered the confirmation code sent to your phone, you are registered as a

standalone staging device with your normal phone number, and a copy of your production

message history and contact list.

Here's the catch: you can't message any of these contacts, since they haven't done the

same thing. Who can you message for testing?

## Additional storage profiles

What you need for proper testing is additional phone numbers, to register additional

standalone devices. You can get them via

[Twilio ($1/mo. per number + $0.0075 per SMS)](https://www.twilio.com/), or via

[Google Voice (one number per Google Account, free SMS)](https://voice.google.com/).

Once you have the additional numbers, you can setup additional storage profiles and switch

between them using the `NODE\_APP\_INSTANCE` environment variable.

For example, to create an 'alice' profile, put a file called `local-alice.json` in the

`config` directory:

```

{

"storageProfile": "aliceProfile"

}

```

Then you can start up the application a little differently to load the profile:

```

NODE\_APP\_INSTANCE=alice yarn run start

```

This changes the [userData](https://electron.atom.io/docs/all/#appgetpathname)

directory from `%appData%/Signal` to `%appData%/Signal-aliceProfile`.

# Making changes

So you're in the process of preparing that pull request. Here's how to make that go

smoothly.

## Tests

Please write tests! Our testing framework is

[mocha](http://mochajs.org/) and our assertion library is

[chai](http://chaijs.com/api/assert/).

The easiest way to run all tests at once is `yarn test`.

You can browse tests from the command line with `grunt unit-tests` or in an

interactive session with `NODE\_ENV=test yarn run start`. The `libtextsecure` tests are run

similarly: `grunt lib-unit-tests` and `NODE\_ENV=test-lib yarn run start`. You can tweak

the appropriate `test.html` for both of these runs to get code coverage numbers via

`blanket.js` (it's shown at the bottom of the web page when the run is complete).

To run Node.js tests, you can run `yarn test-server` from the command line. You can get

code coverage numbers for this kind of run via `yarn test-server-coverage`, then display

the report with `yarn open-coverage`.

## Pull requests

So you wanna make a pull request? Please observe the following guidelines.

- First, make sure that your `yarn ready` run passes - it's very similar to what our

Continuous Integration servers do to test the app.

- Please do not submit pull requests for translation fixes. Anyone can update

the translations in

[Transifex](https://www.transifex.com/projects/p/signal-desktop).

- Never use plain strings right in the source code - pull them from `messages.json`!

You \*\*only\*\* need to modify the default locale

[`\_locales/en/messages.json`](\_locales/en/messages.json). Other locales are generated

automatically based on that file and then periodically uploaded to Transifex for

translation.

- [Rebase](https://nathanleclaire.com/blog/2014/09/14/dont-be-scared-of-git-rebase/) your

changes on the latest `development` branch, resolving any conflicts.

This ensures that your changes will merge cleanly when you open your PR.

- Be sure to add and run tests!

- Make sure the diff between our master and your branch contains only the

minimal set of changes needed to implement your feature or bugfix. This will

make it easier for the person reviewing your code to approve the changes.

Please do not submit a PR with commented out code or unfinished features.

- Avoid meaningless or too-granular commits. If your branch contains commits like

the lines of "Oops, reverted this change" or "Just experimenting, will

delete this later", please [squash or rebase those changes away](https://robots.thoughtbot.com/git-interactive-rebase-squash-amend-rewriting-history).

- Don't have too few commits. If you have a complicated or long lived feature

branch, it may make sense to break the changes up into logical atomic chunks

to aid in the review process.

- Provide a well written and nicely formatted commit message. See [this

link](http://chris.beams.io/posts/git-commit/)

for some tips on formatting. As far as content, try to include in your

summary

1. What you changed

2. Why this change was made (including git issue # if appropriate)

3. Any relevant technical details or motivations for your implementation

choices that may be helpful to someone reviewing or auditing the commit

history in the future. When in doubt, err on the side of a longer

commit message.

Above all, spend some time with the repository. Follow the pull request template added to

your pull request description automatically. Take a look at recent approved pull requests,

see how they did things.

## Linking to a staging mobile device

Multiple standalone desktop devices are great for testing of a lot of scenarios. But a lot

of the Signal experience requires a primary mobile device: contact management,

synchronizing read and verification states among all linked devices, etc.

This presents a problem - even if you had another phone, the production versions of the

iOS and Android apps are locked to the production servers. To test all scenarios in

staging, your best bet is to pull down the development version of the iOS or Android app,

and register it with one of your extra phone numbers:

First, build Signal for Android or iOS from source, and point its service URL to `textsecure-service-staging.whispersystems.org`:

\*\*on Android:\*\* Replace the `SIGNAL\_URL` value in [build.gradle](https://github.com/signalapp/Signal-Android/blob/master/build.gradle)

\*\*on iOS:\*\* Replace the `textSecureServerURL` value in `TSConstants.h`(located in the SignalServiceKit pod)

This task is 1% search and replace, 99% setting up your build environment. Instructions are available for both

the [Android](https://github.com/signalapp/Signal-Android/blob/master/BUILDING.md)

and [iOS](https://github.com/signalapp/Signal-iOS/blob/master/BUILDING.md) projects.

Then you can set up your development build of Signal Desktop as normal. If you've already

set up as a standalone install, you can switch by opening the DevTools (View -> Toggle

Developer Tools) and entering this into the Console and pressing enter: `window.owsDesktopApp.appView.openInstaller();`

## Changing to production

If you're completely sure that your changes will have no impact to the production servers,

you can connect your development build to the production server by putting a file called

`local-development.json` in the `config` directory with the same contents as

`production.json`, except that you should also remove the `updatesEnabled` setting so that

the auto update infrastructure doesn't kick in while you are developing.

`local-development.json` should look something like this:

```json

{

"serverUrl": "https://textsecure-service.whispersystems.org",

"serverTrustRoot": "SOME\_ALPHANUMERIC\_STRING\_MATCHING\_PRODUCTION\_JSON",

"cdn": {

"0": "https://cdn.signal.org",

"2": "https://cdn2.signal.org"

}

}

```

\*\*Beware:\*\* Setting up standalone with your primary phone number when connected to the

production servers will \_unregister\_ your mobile device! All messages from your contacts

will go to your new development desktop app instead of your phone.

## Testing Production Builds

To test changes to the build system, build a release using

```

yarn generate

yarn build

```

Then, run the tests using `grunt test-release:osx --dir=release`, replacing `osx` with `linux` or `win` depending on your platform.

## Translations

To pull the latest translations, follow these steps:

1. Download Transifex client:

https://docs.transifex.com/client/installing-the-client

2. Create Transifex account: https://transifex.com

3. Generate API token: https://www.transifex.com/user/settings/api/

4. Create `~/.transifexrc` configuration:

https://docs.transifex.com/client/client-configuration#-transifexrc

5. Run `yarn grunt tx`.