# Contributing to Urbit

Thank you for your interest in contributing to Urbit.

See [urbit.org/docs/getting-started][start] for basic orientation and usage

instructions. You may also want to subscribe to [urbit-dev][list], the Urbit

development mailing list. For specific information on contributing to the Urbit

interface, see its [contribution guidelines][interface].

[start]: https://urbit.org/docs/getting-started/#arvo

[interface]: /pkg/interface/CONTRIBUTING.md

## Fake ships

You may have an identity on the live network, but doing all your development on

the live network would be cumbersome and unnecessary. Standard practice in

Urbit development is to work on a fake `~zod`. Fake ships use deterministic

keys (derived from the ship address) and don't talk to the live network. They

can talk to each other over the local loopback.

To start a fake ship, simply specify the name with `-F`:

```

$ urbit -F zod

```

You can also pass a name for the \*pier\* (or ship directory):

```

$ urbit -F zod -c my-fake-zod

```

To resume a fake ship, just pass the name of the pier:

```

$ urbit my-fake-zod

```

## Git practice

### Contributing

The canonical source tree is the `master` branch of

[https://github.com/urbit/urbit][repo]. You should typically branch off of

`master` when commencing new work; similarly, when we pull in your

contribution, we'll do so by merging it to `master`.

Since we use GitHub, it's helpful (though not required) to contribute via a

GitHub pull request. You can also post patches to the [mailing list][list],

email them to maintainers, or request a maintainer pull from your tree directly

-- but note that some maintainers will be more receptive to these methods than

others.

When contributing changes, via whatever means, make sure you describe them

appropriately. You should attach a reasonably high-level summary of what the

changes are and what they do; reference any useful background material that may

exist, e.g. a GitHub issue, a mailing list discussion, a UP, etc. [Here][jbpr]

is a good example of a pull request with a useful, concise description.

If your changes replace significant extant functionality, be sure to compare

them with the thing you're replacing. You may also want to cc maintainers,

reviewers, or other parties who might have a particular interest in what you're

contributing.

[jbpr]: https://github.com/urbit/urbit/pull/1782

### Hygiene

Commits should generally be relevant, atomic, and have descriptions formatted

in the following manner:

> component: short description

>

> long description

The 'component' is a short prefix of what area of the codebase the commit

applies to. If a commit patches `%gall`, for example, the description should

be prefixed by 'gall'. If it touches `:aqua`, it should be prefixed by 'aqua'.

If it touches multiple components, then separate these by commas, e.g. "gall,

aqua, ph" -- but note that this may be a warning that too many changes are

being packed into a single commit. The 'component' and 'short description'

combined should be no more than 50 characters.

A lengthier description is encouraged, where useful, but is not always strictly

required. You should use the longer description to give any useful background

on or motivation for the commit, provide a summary of what it does, link to

relevant issues, proposals, or other commits, and so on.

Here is an example of our commit format, taken from a commit in the history:

> zuse: remove superfluous 'scup' and 'culm' types.

>

> %zuse includes definitions for 'scup' and 'culm', both of which are

> superfluous. 'scup' is simply (pair ship desk) and is used only in

> the definition of 'culm', a tagged union in which three of the four

> branches are commented out (i.e. are unused).

>

> This commit deletes 'scup' and 'culm' and refactors what little code

> made use of them.

Note that the short description is prefixed by `zuse:`, which is what the

commit touches. Otherwise it just includes a summary of the change.

Here's another example:

> build: give arvo a high priority

>

> 0bdced981e4 introduced the 'arvo-ropsten' derivation. Attempting to

> install both 'arvo' and 'arvo-ropsten' via nix-env will result in a

> priority error; this assigns a higher priority to 'arvo' to resolve the

> conflict.

>

> Fixes #1912.

Note that it cites a previous relevant commit, `0bdced981e4`, in its summary,

and also points at the issue that it resolves.

If you're in doubt about how to format your commit descriptions, take a look at

the recent history and try to mimic the style that you can see others broadly

follow there.

When we say commits should be "atomic", we mean with respect to some distinct

logical unit, e.g. a type definition used across many files, or a single file,

or just a single function in a single file. Commits should be atomic at the

level of \*code\*, not of entire features. You don't have to squash your commits

into a single one that captures everything you're trying to do -- the history

will never make for pleasant bedtime reading, so focus instead on making your

commits useful for tools like `git-blame` and `git-bisect`.

Your contribution must apply cleanly to `master` in order to be considered

mergeable. You may want to regularly [rebase your changes][reba] onto `master`

in order to both clean up any intermediate "development" commits you make and

to ensure that you're up to date.

If you're making a GitHub pull request, it's good practice to make it from a

topic branch, rather than `master`, on your fork.

### Pills

Any contribution that touches the kernel (i.e., anything in `pkg/arvo/sys`),

should be accompanied by an updated [solid pill](#the-kernel-and-pills). Pills

are tracked in the repository via [git LFS][git-lfs].

Whenever you make a contribution to the kernel, please create a new solid pill

via:

```

sh/update-solid-pill

```

and include it along with your contribution.

Historically, we've sometimes included these updated pills in separate,

standalone commits (you will see plenty of "pills: update solid" and similar

commits if you look through the history), but this practice is considered to be

deprecated -- you should usually just include the updated pill in the same

commit that updates the source.

## Releases

We typically create releases by tagging appropriate commits on `master`, so any

given commit in `master` may not actually be present in the latest release.

We perform updates by pushing releases over-the-air to `~zod` approximately

once per week, so any contribution that can be deployed OTA will usually find

its way onto the network pretty rapidly.

If you want to propose a hotfix (i.e. a small, OTA-updateable change, usually a

bugfix, to some currently-deployed release) then simply make it clear that your

contribution -- whether it be a pull request, patch, or whatever -- is intended

to be a hotfix. A maintainer can then deploy it to the network outside of the

normal release schedule.

Less frequently we release new Vere versions, which requires users to download

new binaries, and occasionally, while Urbit is still in early development, we

breach network continuity in order to release large changes that are difficult

to push out over-the-air. Contributions to Vere, or non-OTA-able updates to

Arvo, will find their way into releases before terribly long.

## Code style

The Urbit project uses two-space indentation and avoids tab characters.

In C code, it should not be too difficult to mimic the style of the code

around you, which is just fairly standard K&R with braces on every

compound statement. One thing to watch out for is top-level sections in

source files that are denoted by comments and are actually indented one

level.

Hoon will be a less familiar language to many contributors. We've published

some [style guidelines for Hoon][hoon], but above all you should try to mimic

the style of the code around you. With regards to the style used throughout

the codebase: the more recently the code was written, the more standard and

accepted its style is likely to be.

## Kernel development

Working on either C or non-kernel Hoon should not bring any surprises, but the

Hoon kernel (anything under [`pkg/arvo/sys/`][sys]) is bootstrapped from a

so-called \*pill\*, and must be recompiled if any changes are made. This should

happen automatically when you make changes, but if it doesn't, the command to

manually recompile and install the new kernel is `|reset` in `dojo`. This

rebuilds from the `sys` directory in the `home` desk in `%clay`.

Currently, `|reset` does not reload apps like `dojo` itself, which will still

reference the old kernel. To force them to reload, make a trivial edit to their

main source file (under the `app` directory) in `%clay`.

[arvo]: https://github.com/urbit/urbit/tree/master/pkg/arvo

[sys]: https://github.com/urbit/urbit/tree/master/pkg/arvo/sys

## The kernel and pills

Urbit bootstraps itself using a binary blob called a pill (you can see it being

fetched from `bootstrap.urbit.org` on boot). This is the compiled version of

the kernel (which you can find in the `sys` directory of [Arvo][arvo]), along

with a complete copy of the Arvo source.

The procedure for creating a pill is often called "soliding." It is somewhat

similar to `|reset`, but instead of replacing your running kernel, it writes

the compiled kernel to a file. The command to solid is:

```

> .urbit/pill +solid

```

When the compilation finishes, your pill will be found in the

`[pier]/.urb/put/` directory as `urbit.pill`.

You can boot a new ship from your local pill with `-B`:

```

$ urbit -F zod -B path/to/urbit.pill my-fake-zod

```

Release pills, i.e. those corresponding to vere releases, are cached at

`https://bootstrap.urbit.org` and are indexed by the vere version number, e.g.

`urbit-0.8.2.pill`.

Pills are also cached in version control via [git LFS][git-lfs]. You can find

the latest solid pill, as well as the latest so-called \*brass\* and \*ivory\*

pills, in the `bin/` directory at the repository root. Note that you'll need

to initialise git LFS in order to check these pills out:

```

$ git lfs init

$ git lfs pull

```

[git-lfs]: https://git-lfs.github.com

## Issues

The [GitHub tracker][issu] is our canonical source of truth around issues,

bugs, performance problems, feature requests, and so on. If you encounter any

issues when developing on Urbit, feel free to submit a report about it here.

A good bug report, description of a crash, etc., should ideally be

\*reproducible\*, with clear steps as to how another developer can replicate and

examine your problem. That said, this isn't always possible -- some bugs

depend on having created a complicated or unusual state, or can otherwise

simply be difficult to trigger again (say, you encountered it in the last

continuity era).

Your issue should thus at a minimum be \*informative\*. The best advice here is

probably "don't write bad issues," where "bad" is a matter of judgment and

taste. Issues that the maintainers don't judge to be sufficiently useful or

informative may be closed.

Feature requests are welcome, but they should include sufficient detail and

explanation, as well as a discussion of perceived benefits one could expect

from them. "It would be cool if.." probably does not, in itself, constitute a

good feature request; instead, try to be specific about what you're requesting,

and what your desired feature would accomplish.

## Staying in touch

Questions or other communications about contributing to Urbit can go to

[support@urbit.org][mail].

[mail]: mailto:support@urbit.org

[list]: https://groups.google.com/a/urbit.org/forum/#!forum/dev

[repo]: https://github.com/urbit/urbit

[reba]: https://git-rebase.io/

[issu]: https://github.com/urbit/urbit/issues

[hoon]: https://urbit.org/docs/learn/hoon/style/