Contributing to UraniumX
==========

The UraniumX project operates an open contributor model where anyone is welcome to contribute towards development in the form of peer review, testing and patches. This document explains the practical process and guidelines for contributing.

Firstly in terms of structure, there is no particular concept of "Core developers" in the sense of privileged people. Open source often naturally revolves around meritocracy where longer term contributors gain more trust from the developer community. However, some hierarchy is necessary for practical purposes. As such there are repository "maintainers" who are responsible for merging pull requests as well as a "lead maintainer" who is responsible for the release cycle, overall merging, moderation and appointment of maintainers.

Contributor Workflow

The codebase is maintained using the "contributor workflow" where everyone without exception contributes patch proposals using "pull requests". This facilitates social contribution, easy testing and peer review.

To contribute a patch, the workflow is as follows:

- Fork repository
- Create topic branch
- Commit patches

The project coding conventions in the [developer notes](doc/developer-notes.md) must be adhered to.

In general [commits should be atomic](https://en.wikipedia.org/wiki/Atomic_commit#Atomic_commit_convention) and diffs should be easy to read. For this reason do not mix any formatting fixes or code moves with actual code changes.

Commit messages should be verbose by default consisting of a short subject line (50 chars max), a blank line and detailed explanatory text as separate paragraph(s), unless the title alone is self-explanatory (like "Corrected typo in init.cpp") in which case a single title line is sufficient. Commit messages should be helpful to people reading your code in the future, so explain the reasoning for your decisions. Further explanation [here](http://chris.beams.io/posts/git-commit/).

If a particular commit references another issue, please add the reference, for example `refs #1234`, or `fixes #4321`. Using the `fixes` or `closes` keywords will cause the corresponding issue to be closed when the pull request is merged.

Please refer to the [Git manual](https://git-scm.com/doc) for more information about Git.

- Push changes to your fork
- Create pull request

The title of the pull request should be prefixed by the component or area that the pull request affects. Valid areas as:

- *Consensus* for changes to consensus critical code
- *Docs* for changes to the documentation
- *Qt* for changes to uraniumx-qt
- *Mining* for changes to the mining code
- *Net* or *P2P* for changes to the peer-to-peer network code
- *RPC/REST/ZMQ* for changes to the RPC, REST or ZMQ APIs
- *Scripts and tools* for changes to the scripts and tools
- *Tests* for changes to the uraniumx unit tests or QA tests
- *Trivial* should **only** be used for PRs that do not change generated executable code. Notably, refactors (change of function arguments and code reorganization) and changes in behavior should **not** be marked as trivial. Examples of trivial PRs are changes to:
  - comments
  - whitespace
  - variable names
  - logging and messages
- *Utils and libraries* for changes to the utils and libraries
- *Wallet* for changes to the wallet code

Examples:

```
Consensus: Add new opcode for BIP-XXXX OP_CHECKAWESOMESIG
Net: Automatically create hidden service, listen on Tor
Qt: Add feed bump button
Trivial: Fix typo in init.cpp
```

If a pull request is specifically not to be considered for merging (yet) please prefix the title with [WIP] or use [Tasks Lists](https://help.github.com/articles/basic-writing-and-formatting-syntax/#task-lists)
in the body of the pull request to indicate tasks are pending.

The body of the pull request should contain enough description about what the patch does together with any justification/reasoning. You should include references to any discussions (for example other tickets or mailing list discussions).

At this stage one should expect comments and review from other contributors. You can add more commits to your pull request by committing them locally and pushing to your fork until you have satisfied all feedback.

Squashing Commits

If your pull request is accepted for merging, you may be asked by a maintainer to squash and or [rebase](https://git-scm.com/docs/git-rebase) your commits before it will be merged. The basic squashing workflow is shown below.

```
git checkout your_branch_name
git rebase -i HEAD~n
# n is normally the number of commits in the pull
# set commits from 'pick' to 'squash', save and quit
# on the next screen, edit/refine commit messages
# save and quit
```

```
git push -f # (force push to GitHub)
```

If you have problems with squashing (or other workflows with `git`), you can alternatively enable "Allow edits from maintainers" in the right GitHub sidebar and ask for help in the pull request.

Please refrain from creating several pull requests for the same change. Use the pull request that is already open (or was created earlier) to amend changes. This preserves the discussion and review that happened earlier for the respective change set.

The length of time required for peer review is unpredictable and will vary from pull request to pull request.

Pull Request Philosophy

Patchsets should always be focused. For example, a pull request could add a feature, fix a bug, or refactor code; but not a mixture. Please also avoid super pull requests which attempt to do too much, are overly large, or overly complex as this makes review difficult.

Features

When adding a new feature, thought must be given to the long term technical debt and maintenance that feature may require after inclusion. Before proposing a new feature that will require maintenance, please consider if you are willing to maintain it (including bug fixing). If features get orphaned with no maintainer in the future, they may be removed by the Repository Maintainer.

Refactoring

Refactoring is a necessary part of any software project's evolution. The following guidelines cover refactoring pull requests for the project.

There are three categories of refactoring, code only moves, code style fixes, code refactoring. In general refactoring pull requests should not mix these three kinds of activity in order to make refactoring pull requests easy to review and uncontroversial. In all cases, refactoring PRs must not change the behaviour of code within the pull request (bugs must be preserved as is).

Project maintainers aim for a quick turnaround on refactoring pull requests, so where possible keep them short, uncomplex and easy to verify.

"Decision Making" Process

The following applies to code changes to the UraniumX project (and related projects such as libsecp256k1), and is not to be confused with overall UraniumX Network Protocol consensus changes.

Whether a pull request is merged into UraniumX rests with the project merge maintainers and ultimately the project lead.

Maintainers will take into consideration if a patch is in line with the general principles of the project; meets the minimum standards for inclusion; and will judge the general consensus of contributors.

In general, all pull requests must:

- Have a clear use case, fix a demonstrable bug or serve the greater good of the project (for example refactoring for modularisation);
- Be well peer reviewed;
- Have unit tests and functional tests where appropriate;
- Follow code style guidelines;
- Not break the existing test suite;
- Where bugs are fixed, where possible, there should be unit tests demonstrating the bug and also proving the fix. This helps prevent regression.

Patches that change UraniumX consensus rules are considerably more involved than normal because they affect the entire ecosystem and so must be preceded by extensive mailing list discussions and have a numbered BIP. While each case will be different, one should be prepared to expend more time and effort than for other kinds of patches because of increased peer review and consensus building requirements.

Peer Review

Anyone may participate in peer review which is expressed by comments in the pull request. Typically reviewers will review the code for obvious errors, as well as test out the patch set and opine on the technical merits of the patch. Project maintainers take into account the peer review when determining if there is consensus to merge a pull request (remember that discussions may have been spread out over GitHub, mailing list and IRC discussions). The following language is used within pull-request comments:

- ACK means "I have tested the code and I agree it should be merged";
- NACK means "I disagree this should be merged", and must be accompanied by sound technical justification (or in certain cases of copyright/patent/licensing issues, legal justification). NACKs without accompanying reasoning may be disregarded;
- utACK means "I have not tested the code, but I have reviewed it and it looks OK, I agree it can be merged";
- Concept ACK means "I agree in the general principle of this pull request";
- Nit refers to trivial, often non-blocking issues.

Reviewers should include the commit hash which they reviewed in their comments.

Project maintainers reserve the right to weigh the opinions of peer reviewers using common sense judgement and also may weight based on meritocracy: Those that have demonstrated a deeper commitment and understanding towards the project (over time) or have clear domain expertise may naturally have more weight, as one would expect in all walks of life.

Where a patch set affects consensus critical code, the bar will be set much

higher in terms of discussion and peer review requirements, keeping in mind that mistakes could be very costly to the wider community. This includes refactoring of consensus critical code.

Where a patch set proposes to change the UraniumX consensus, it must have been discussed extensively on the mailing list and IRC, be accompanied by a widely discussed BIP and have a generally widely perceived technical consensus of being a worthwhile change based on the judgement of the maintainers.

Finding Reviewers

As most reviewers are themselves developers with their own projects, the review process can be quite lengthy, and some amount of patience is required. If you find that you've been waiting for a pull request to be given attention for several months, there may be a number of reasons for this, some of which you can do something about:

- It may be because of a feature freeze due to an upcoming release. During this time, only bug fixes are taken into consideration. If your pull request is a new feature, it will not be prioritized until the release is over. Wait for release.
- It may be because the changes you are suggesting do not appeal to people. Rather than nits and critique, which require effort and means they care enough to spend time on your contribution, thundering silence is a good sign of widespread (mild) dislike of a given change (because people don't assume *others* won't actually like the proposal). Don't take that personally, though! Instead, take another critical look at what you are suggesting and see if it: changes too much, is too broad, doesn't adhere to the [developer notes](doc/developer-notes.md), is dangerous or insecure, is messily written, etc. Identify and address any of the issues you find. Then ask e.g. on IRC if someone could give their opinion on the concept itself.
- It may be because your code is too complex for all but a few people. And those people may not have realized your pull request even exists. A great way to find people who are qualified and care about the code you are touching is the [Git Blame feature](https://help.github.com/articles/tracing-changes-in-a-file/). Simply find the person touching the code you are touching before you and see if you can find them and give them a nudge. Don't be incessant about the nudging though.
- Finally, if all else fails, ask on IRC or elsewhere for someone to give your pull request a look. If you think you've been waiting an unreasonably long amount of time (month+) for no particular reason (few lines changed, etc), this is totally fine. Try to return the favor when someone else is asking for feedback on their code, and universe balances out.

Release Policy

The project leader is the release manager for each UraniumX release.

Copyright

By contributing to this repository, you agree to license your work under the MIT license unless specified otherwise in `contrib/debian/copyright` or at the top of the file itself. Any work contributed where you are not the original author must contain its license header with the original author(s) and source.