Contributing

The following information provides a set of guidelines for contributing to the Osmo chain dev repo. Use your best judgment, and, if you see room for improvement, please propose changes to this document.

The contributing guide for Osmosis explains the branching structure, how to use the SDK fork, how to make / test updates to SDK branches and how to create release notes.

Contributions come in the form of writing documentation, raising issues / PRs, and any other actions that help develop the Osmo protocol documentation.

First steps

The first step is to find an issue you want to fix. To identify issues we think are good for first-time contributors, we add thegood first issue label.

If you have a feature request, please use thefeature-request repo

Once you find an existing issue that you want to work on or if you have a new issue to create, continue below.

Proposing changes

To contribute a change proposal, use the following workflow:

1. Fork the repository
2. .
3. Add an upstream
4. so that you can update your fork.
5. Clone your fork to your computer.
6. Create a branch and name it appropriately.
7. Work on only one major change in one pull request.
8. Make sure all tests are passing locally.
9. Next, rinse and repeat the following:
10.
    1. Commit your changes. Write a simple, straightforward commit message. To learn more, seeHow to Write a Git Commit Message
11.
    1. .
12.
    1. Push your changes to your remote fork.
13.
    1. Create a PR on the Osmo repository. There should be a PR template to help you do so.
14.
    1. Wait for your changes to be reviewed. If you are a maintainer, you can assign your PR to one or more reviewers. If you aren't a maintainer, one of the maintainers will assign a reviewer.
15.
    1. After you receive feedback from a reviewer, make the requested changes, commit them to your branch, and push them to your remote fork again.
16.
    1. Once approval is given, feel free to squash & merge!

Working with the SDK

Updating dependencies for builds

Vendor is a folder that go automatically makes if you run go mod vendor, which contains the source code for all of your dependencies. Its often helpful for local debugging. In order to update it...

Commit & push to the Cosmos-SDK fork in a new branch (see above steps for more details), and then you can grab the commit hash to do:

go get github.com/osmosis-labs/cosmos-sdk@{my commit hash}

You get something like:

go get: github.com/osmosis-labs/cosmos-sdk@v0.33.2 updating to github.com/osmosis-labs/cosmos-sdk@v0.42.10-0.20210829064313-2c87644925da: parsing go.mod: module declares its path as: github.com/cosmos/cosmos-sdk but was

required as: github.com/osmosis-labs/cosmos-sdk

Then you can copy paste thev0.42.10-0.20210829064313-2c87644925da part and replace the corresponding section of go.mod

Then dogo mod vendor , and you're set.

Changing things in vendor for local builds / local testing

In whichever folder you're running benchmarks for, you can test via:

go test -benchmem -bench DistributionLogicLarge -cpuprofile cpu.out -test.timeout 30m -v

Then once that is done, and you get the short benchmark results out, you can do:

go tool pprof -http localhost:8080 cpu.out

and take look at the graphviz output!

Note that if you are doing things that are low-level / small, the overhead of cpuprofile may mess with cache effects, etc. However for things like epoch code, or relatively large txs, this totally works!

Branch structure of releases on v7, v6, v4

People still need those versions for querying old versions of the chain, and syncing a node from genesis, so we keep these updated!

For v6.x, and v4.x, most PRs to them should go to main and get a "backport" label. We typically use mergify for backporting. Backporting often takes place after a PR has been merged to main

How to build proto files. (rm -rf vendor/ && make build-reproducible once docker is installed)

You can do rm -rf vendor and make build-reproducible to redownload all dependencies - this should pull the latest docker image of Osmo. You should also make sure to do make proto-all to auto-generate your protobuf files. Makes ure you have docker installed. If you get something likeW0503 22:16:30.068560 158 services.go:38] No HttpRule found for method: Msg.CreateBalancerPool feel free to ignore that. Make sure to also do make all to run all the linting tests before you commit and push, as well asgofmt -ing the file you've modified or added to make sure everything still abides by the standards. Edit this page Previous IBC Relayer List Next IBC Protocol * First steps * Proposing changes * Working with the SDK * * Updating dependencies for builds * * Changing things in vendor for local builds / local testing* * Branch structure of releases on v7, v6, v4 * * How to build proto files. (rm -rf vendor/ && make build-reproducible once docker is installed)