Verifying Code Updates

How to verify code updates of smart contracts deployed on Libre.

Code Verification Instructions

About Code Verification

The purpose here is to be able to read and even deploy locally both the current version and the proposed changes for testing.

Ideally, you should know how to use git, how to run a diff on code changes, and how to calculate a sha256sum. That being said, we have written it all out for you anyway.

Cookies
This site uses cookies to deliver its service and to analyse traffic. By browsing this site, you accept the cookie policy.
Reject all

1. Verify currently deployed code locally
2. Verify proposed code locally
3. Compare proposed changes to deployed code (can be done in Gitlab UI or VS Code)

Steps are listed below and we are working on a video walkthrough for the future so basically anyone with a terminal and a keyboard will be able to do this verification.

Steps to verify:

1. Clone repo
2. Get sha256sum hash of currently deployed contract `libre.sh get code stake.libre` - compare to table above “currently deployed”
3. Git checkout commit of currently deployed contract (provided in proposal)
4. Build WASM (docker if avail) `./docker-build.sh` (or older repos) `./build.sh`
5. Verify sha256sum of built wasm matches deployed contract (from step 2) sha256sum $PATH_TO_WASM (provided in proposal)

Cookies

This site uses cookies to deliver its service and to analyse traffic. By browsing this site, you accept the cookie policy.

Reject all

7. Build WASM of the proposed update `./docker-build.sh` (or older repos) `./build.sh`
8. Check sha256sum of wasm you built `sha256sum $PATH_TO_WASM` (provided in proposal)
9. If it matches the sha256sum in the multisig on-chain, then you are looking at the correct version of code! (only the sha256sum of the wasm matters here - abi will not match, but you can download it from the multisig proposal and review if you like)
10. Inspect code changes between proposed update and deployed version using the Code Comparison link (provided in proposal) or by running a diff between commits (source: proposed, target: deployed)

Multisig Execution Instructions

How to Approve

1. Use Anchor and login to local.blocks.io and click "Approve" (recommended)
2. Use these CLI multisig commands below (2

Cookies

This site uses cookies to deliver its service and to analyse traffic. By browsing this site, you accept the cookie policy.

Reject all

versions that require **cleos** to be installed)

CLI Example 1 - use libre.sh

Create libre.sh for CLI transactions on Linux

```
#!/bin/bash
CLEOS=/usr/opt/eosio/2.1.0-r
API=https://api.libre.eosswe
$CLEOS -u $API "$@"
```

Replace "sweden" with your account name in the "actor" key below:

```
./libre.sh multisig approve quar
p sweden@active
```

CLI example 2 - simple cleos

Replace sweden with your producer name in all 4 actor keys:

```
cleos -u http://lb.libre.org
  "delay_sec": 0,
  "max_cpu_usage_ms": 0,
  "actions": [
    {
      "account": "eosio.msig
      "name": "approve",
      "data": {
        "proposer": "quantum
        "proposal_name": "fo
        "level": [
          "sweden",
          "active"
        ]
      },
```

Cookies

This site uses cookies to deliver its service and to analyse traffic. By browsing this site, you accept the cookie policy.

Reject all

```
      "authorization": [
        {
          "actor": "sweden",
          "permission": "act
        }
      ]
    }
  ]
}'
```


Technical Details - Previous
Staking Details

Next
Deployed Smart Cont...

Last modified 3mo ago

Cookies

This site uses cookies to deliver its service and to analyse traffic. By browsing this site, you accept the cookie policy.

Reject all