

# Celestia-node RPC CLI tutorial

In this tutorial, we will cover how to use the celestia-node RPC API to submit and retrieve data (blobs) from the data availability layer by their namespace.

## Introduction

### Blobs

Data is posted to Celestia's DA layer by using `MsgPayForBlobs` transactions to the core network. Read [more about MsgPayForBlobs](#).

### Namespaces

Celestia partitions the block data into multiple namespaces, one for every application. This allows applications to only download their data, and not the data of other applications. Read [more about Namespaced Merkle trees \(NMTs\)](#).

TIP

If you already have a running and funded node, you can skip to the [RPC CLI guide section](#). WARNING

The gateway endpoints have been deprecated and will be removed in the future. If you would like to use them anyway, you can [find more details on GitHub](#).

## Hardware requirements

The following minimum hardware requirements are recommended for running a light node:

- Memory: 500 MB RAM (minimum)
- CPU: Single Core
- Disk: 50 GB SSD Storage
- Bandwidth: 56 Kbps for Download / 56 Kbps for Upload

## Setting up dependencies

This portion of the tutorial will go over setting up your development environment to run Celestia software. This environment can be used for development, building binaries, and running nodes.

In your terminal, set up dependencies needed to install and build celestia-node.

1. If you are on Ubuntu, first update and upgrade your OS:
2. `APT`
3. `YUM`
4. `bash`
5. `sudo`
6. `apt`
7. `update`
8. `&&`
9. `sudo`
10. `apt`
11. `upgrade`
12. `-y`
13. `sudo`
14. `apt`
15. `update`
16. `&&`
17. `sudo`
18. `apt`
19. `upgrade`
20. `-y`
21. `bash`
22. `sudo`
23. `yum`
24. `update`
25. `sudo`
26. `yum`

27. update
28. Install essential packages that are necessary to execute many tasks like downloading files, compiling, and monitoring the node:
29. APT
30. YUM
31. Mac
32. bash
33. sudo
34. apt
35. install
36. curl
37. tar
38. wget
39. clang
40. pkg-config
41. libssl-dev
42. jq
43. \
44. build-essential
45. git
46. make
47. ncdu
48. -y
49. sudo
50. apt
51. install
52. curl
53. tar
54. wget
55. clang
56. pkg-config
57. libssl-dev
58. jq
59. \
60. build-essential
61. git
62. make
63. ncdu
64. -y
65. bash
66. sudo
67. yum
68. install
69. curl
70. tar
71. wget
72. clang
73. pkg-config
74. libssl-dev
75. jq
76. \
77. build-essential
78. git
79. make
80. ncdu
81. -y
82. sudo
83. yum
84. install
85. curl
86. tar
87. wget
88. clang
89. pkg-config
90. libssl-dev
91. jq
92. \
93. build-essential

```
94. git
95. make
96. ncdu
97. -y
98. bash
```

99. **these commands are for installing Homebrew, wget and jq**

100. **follow the instructions from the output after running this command**

```
101. /bin/bash
102. -c
103. "(
104. curl
105. -fsSL
106. https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

107. **then install wget & jq**

```
108. brew
109. install
110. wget
111. &&
112. brew
113. install
114. jq
```

115. **these commands are for installing Homebrew, wget and jq**

116. **follow the instructions from the output after running this command**

```
117. /bin/bash
118. -c
119. "(
120. curl
121. -fsSL
122. https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

123. **then install wget & jq**

```
124. brew
125. install
126. wget
127. &&
128. brew
129. install
130. jq
```

## Install Golang

celestia-node is written in Golang so we must install Golang to build and run our node.

1. Set the version for your desired network:

2. Mainnet Beta
3. Mocha
4. Arabica
5. bash
6. ver
7. =
8. "1.21.1"
9. ver
10. =
11. "1.21.1"
12. bash
13. ver
14. =
15. "1.21.1"
16. ver
17. =
18. "1.21.1"
19. bash
20. ver
21. =
22. "1.21.1"
23. ver
24. =
25. "1.21.1"
26. Download and install Golang:
27. Ubuntu (AMD)
28. Ubuntu (ARM)
29. Mac (Apple)
30. Mac (Intel)
31. bash
32. cd
33. HOME
34. wget
35. "https://golang.org/dl/go
36. ver
37. .linux-amd64.tar.gz"
38. sudo
39. rm
40. -rf
41. /usr/local/go
42. sudo
43. tar
44. -C
45. /usr/local
46. -xzf
47. "go
48. ver
49. .linux-amd64.tar.gz"
50. rm
51. "go
52. ver
53. .linux-amd64.tar.gz"
54. cd
55. HOME
56. wget
57. "https://golang.org/dl/go
58. ver
59. .linux-amd64.tar.gz"
60. sudo
61. rm
62. -rf
63. /usr/local/go
64. sudo
65. tar
66. -C
67. /usr/local
68. -xzf
69. "go

```
70. ver
71. .linux-amd64.tar.gz"
72. rm
73. "go
74. ver
75. .linux-amd64.tar.gz"
76. bash
77. cd
78. HOME
79. wget
80. "https://golang.org/dl/go
81. ver
82. .linux-arm64.tar.gz"
83. sudo
84. rm
85. -rf
86. /usr/local/go
87. sudo
88. tar
89. -C
90. /usr/local
91. -xzf
92. "go
93. ver
94. .linux-arm64.tar.gz"
95. rm
96. "go
97. ver
98. .linux-arm64.tar.gz"
99. cd
100. HOME
101. wget
102. "https://golang.org/dl/go
103. ver
104. .linux-arm64.tar.gz"
105. sudo
106. rm
107. -rf
108. /usr/local/go
109. sudo
110. tar
111. -C
112. /usr/local
113. -xzf
114. "go
115. ver
116. .linux-arm64.tar.gz"
117. rm
118. "go
119. ver
120. .linux-arm64.tar.gz"
121. bash
122. cd
123. HOME
124. wget
125. "https://golang.org/dl/go
126. ver
127. .darwin-arm64.tar.gz"
128. sudo
129. rm
130. -rf
131. /usr/local/go
132. sudo
133. tar
134. -C
135. /usr/local
136. -xzf
137. "go
```

```
138. ver
139. .darwin-arm64.tar.gz"
140. rm
141. "go
142. ver
143. .darwin-arm64.tar.gz"
144. cd
145. HOME
146. wget
147. "https://golang.org/dl/go
148. ver
149. .darwin-arm64.tar.gz"
150. sudo
151. rm
152. -rf
153. /usr/local/go
154. sudo
155. tar
156. -C
157. /usr/local
158. -xzf
159. "go
160. ver
161. .darwin-arm64.tar.gz"
162. rm
163. "go
164. ver
165. .darwin-arm64.tar.gz"
166. bash
167. cd
168. HOME
169. wget
170. "https://golang.org/dl/go
171. ver
172. .darwin-amd64.tar.gz"
173. sudo
174. rm
175. -rf
176. /usr/local/go
177. sudo
178. tar
179. -C
180. /usr/local
181. -xzf
182. "go
183. ver
184. .darwin-amd64.tar.gz"
185. rm
186. "go
187. ver
188. .darwin-amd64.tar.gz"
189. cd
190. HOME
191. wget
192. "https://golang.org/dl/go
193. ver
194. .darwin-amd64.tar.gz"
195. sudo
196. rm
197. -rf
198. /usr/local/go
199. sudo
200. tar
201. -C
202. /usr/local
203. -xzf
204. "go
205. ver
```

```
206. .darwin-amd64.tar.gz"
207. rm
208. "go
209. ver
210. .darwin-amd64.tar.gz"
211. Add your/usr/local/go/bin
212. directory to yourPATH
213. if you have not already:
214. bash
215. zsh
216. bash
217. echo
218. "export PATH=
219. PATH
220. :/usr/local/go/bin:
221. HOME
222. /go/bin"
223.
```

```
224. HOME
225. ~/.bash_profile
226. source
227. HOME
228. ~/.bash_profile
229. echo
230. "export PATH=
231. PATH
232. :/usr/local/go/bin:
233. HOME
234. /go/bin"
235.
```

```
236. HOME
237. ~/.bash_profile
238. source
239. HOME
240. ~/.bash_profile
241. bash
242. echo
243. "export PATH=
244. PATH
245. :/usr/local/go/bin:
246. HOME
247. /go/bin"
248.
```

```
249. HOME
250. ~/.zshrc
251. source
252. HOME
253. ~/.zshrc
254. echo
255. "export PATH=
256. PATH
257. :/usr/local/go/bin:
258. HOME
259. /go/bin"
260.
```

```
261. HOME
262. ~/.zshrc
263. source
264. HOME
265. ~/.zshrc
266. To verify that the correct version of Go was installed correctly run:
267. bash
268. go
269. version
270. go
```

271. version

The output will show the version installed.

## Celestia-node

### Install celestia-node

Installing celestia-node for Arabica devnet or Mocha testnet means installing a specific version to be compatible with the network.

Install the celestia-node binary by running the following commands:

1. Remove any existing copy of celestia-node, clone the repository, and change into the directory.
2. bash
3. cd
4. HOME
5. rm
6. -rf
7. celestia-node
8. git
9. clone
10. <https://github.com/celestiaorg/celestia-node.git>
11. cd
12. celestia-node/
13. cd
14. HOME
15. rm
16. -rf
17. celestia-node
18. git
19. clone
20. <https://github.com/celestiaorg/celestia-node.git>
21. cd
22. celestia-node/
23. Check out to the desired version, based on the network you will use:
24. Mainnet Beta
25. Mocha
26. Arabica
27. bash
28. git
29. checkout
30. tags/v0.12.4
31. git
32. checkout
33. tags/v0.12.4
34. bash
35. git
36. checkout
37. tags/v0.13.1
38. git
39. checkout
40. tags/v0.13.1
41. bash
42. git
43. checkout
44. tags/v0.13.1
45. git
46. checkout
47. tags/v0.13.1
48. Build the celestia
49. binary:
50. a. Standard build
51. bash
52. make
53. build
54. make
55. build



56. b. Experimental build
57. OPTIONAL
58. If you're a node operator comfortable with experimental features and seeking optimal performance with minimal RAM usage, this option is recommended for you.
59. bash
60. make
61. build-jemalloc
62. make
63. build-jemalloc
64. This build option enables CGO, and downloads and installs [jemalloc](#)
65. [.Learn more about the build command](#)
66. .
67. Install the binary:
68. Ubuntu
69. Mac
70. bash
71. make
72. install
73. make
74. install
75. bash
76. make
77. go-install
78. make
79. go-install
80. Build thecel-key
81. utility:
82. bash
83. make
84. cel-key
85. make
86. cel-key
87. Verify that the binary is working and check the version:
88. bash
89. celestia
90. version
91. celestia
92. version

The output will show the semantic version of celestia-node, commit hash, build date, system version, and Golang version.

## Instantiate a Celestia light node

Now, let's instantiate a Celestia Light node:

TIP

RPC endpoints are exposed in all celestia-node types such as light, bridge and full nodes. Mainnet Beta

Mocha

Arabica bash celestia

light

init celestia

light

init bash celestia

light

init

--p2p.network

mocha celestia

light

```
init
--p2p.network
mocha bash celestia
light
init
--p2p.network
arabica celestia
light
init
--p2p.network
```

arabica Instantiating (or initializing) the node means setting up a node store on your machine. This is where the data and your keys will be stored.

## Connect to a public core endpoint

Let's now run the Celestia Light node with a gRPC connection to an example public core endpoint.

Note: You are also encouraged to find a community-run API endpoint and there are several in the Discord. This one is used for demonstration purposes. Check out the [Mocha testnet page](#) , or [Arabica devnet page](#) .

```
Mainnet Beta
Mocha
Arabica bash celestia
light
start
--core.ip
< UR I
    celestia
light
start
--core.ip
< UR I
    bash celestia
light
start
--core.ip
< UR I
--p2p.network
mocha celestia
light
start
--core.ip
```

< UR I

--p2p.network

mocha bash celestia

light

start

--core.ip

< UR I

--p2p.network

arabica celestia

light

start

--core.ip

< UR I

--p2p.network

arabica TIP

The --core.ip gRPC port defaults to 9090, so if you do not specify it in the command line, it will default to that port. You can add the port after the IP address or use the --core.grpc.port flag to specify another port if you prefer.

Refer to [the ports section of the celestia-node troubleshooting page](#) for information on which ports are required to be open on your machine. For example, your command along with an RPC endpoint might look like this:

Mainnet Beta

Mocha

Arabica bash celestia

light

start

--core.ip

consensus.lunaroasis.net celestia

light

start

--core.ip

consensus.lunaroasis.net bash celestia

light

start

--core.ip

rpc-mocha.pops.one

--p2p.network

mocha celestia

light

start

```

--core.ip
rpc-mocha.pops.one
--p2p.network
mocha bash celestia
light
start
--core.ip
validator-1.celestia-arabica-11.com
\ --p2p.network
arabica celestia
light
start
--core.ip
validator-1.celestia-arabica-11.com
\ --p2p.network
arabica

```

## Keys and wallets

You can create your key for your node by running the following command from the celestia-node directory:

TIP

You do not need to declare a network for Mainnet Beta. Refer to [the chain ID section on the troubleshooting page for more information](#)

```

bash ./cel-key
add
< key-name
--keyring-backend
test
--node.type
light
\ --p2p.network
< network
./cel-key
add
< key-name
--keyring-backend
test
--node.type
light
\ --p2p.network
< network

```

You can start your light node with the key created by running the following command:

Mainnet Beta

Mocha

Arabica bash celestia

light

start

--core.ip

< UR I

--keyring.accname

< key-nam e

celestia

light

start

--core.ip

< UR I

--keyring.accname

< key-nam e

bash celestia

light

start

--core.ip

< UR I

--keyring.accname

< key-nam e

\ --p2p.network

mocha celestia

light

start

--core.ip

< UR I

--keyring.accname

< key-nam e

\ --p2p.network

mocha bash celestia

light

start

--core.ip

< UR l

--keyring.accname

< key-name

\ --p2p.network

arabica celestia

light

start

--core.ip

< UR l

--keyring.accname

< key-name

\ --p2p.network

arabica Once you start the light node, a wallet key will be generated for you. You will need to fund that address with Mocha testnet or Arabica devnet tokens to pay for PayForBlobs transactions.

You can find the address by running the following command in the celestia-node directory:

```
bash ./cel-key
```

list

--node.type

light

--keyring-backend

test

--p2p.network

< network

```
./cel-key
```

list

--node.type

light

--keyring-backend

test

--p2p.network

< network

If you would like to fund your wallet with testnet tokens, head over to either the #mocha-faucet or #arabica-faucet channels on the [Celestia Discord](#).

You can request funds to your wallet address using the following command in Discord:

text request request Where is the celestia1\*\* address generated when you created the wallet.

With your wallet funded, you can move on to the next step.

## RPC CLI guide

This section of the tutorial will teach you how to interact with a Celestia node's [remote procedure call \(RPC\) API](#) using the

command line interface (CLI).

You will need to [setup dependencies, install, and run celestia-node](#) if you have not already.

## Command formatting

The format for interacting with the RPC CLI methods is as follows:

```
bash celestia
```

```
< modul e
```

```
< metho d
```

```
    [args...] [flags...] celestia
```

```
< modul e
```

```
< metho d
```

```
    [args...] [flags...] Where:
```

- celestia
- is the main command to interact with the node.
- 
- is the specific module in the node you want to interact with, such as [blob](#)
- [,state](#)
- [,p2p](#)
- , etc.
- 
- is the specific method within the module that performs the action you want, such as [\\$blob.Submit](#)
- [,state.AccountAddress](#)
- [,p2p.Info](#)
- , etc.
- [args...]
- represents any additional arguments that the method might require.
- [flags...]
- are parameters that modify the behavior of the command. They start with--
- (e.g.,--node.store
- ,--token
- , or--url
- ).

For example, to submit a blob to Celestia, you can use this command once your node store is set:

```
bash celestia
```

```
blob
```

```
submit
```

```
0x42690c204d39600fddd3
```

```
'gm'
```

```
--node.store NODE_STORE celestia
```

```
blob
```

```
submit
```

```
0x42690c204d39600fddd3
```

```
'gm'
```

--node.store NODE\_STORE Alternatively, you could use the--token flag to set your auth token:

```
bash celestia
```

```
blob
```

```
submit
```

```
0x42690c204d39600fddd3
```

```
'gm'
```

```
--token AUTH_TOKEN celestia
```

```
blob
```

```
submit
```

```
0x42690c204d39600fddd3
```

```
'gm'
```

--token AUTH\_TOKEN Before you try that out, let's go over the basic flags that you will need to use when interacting with the RPC CLI. We'll also cover how to set your auth token and how to use the node store to set it.

## Basic flags

All RPC CLI commands have basic flags that can be used to interact with the API.

These include:

- --node.store string
  - the path to root/home directory of your celestia-node store
- --token string
  - authorization token for making requests
- --url string
  - the address of the RPC, default is `http://localhost:26658`

When running RPC CLI commands, you will need to set either the [authentication token](#) or set the [node store](#) , so the auth token can be retrieved from the store.

The RPC CLI handles these flags in the following order:

1. If user passes auth token, auth token is used.
2. If user doesn't pass auth token, check node store flag, create token from node store, and use auth token from node store.

## Auth token

In order to interact with the API using RPC CLI, you will need to set the authentication token.

The --token string flag sets the authentication token. If a token is not found, authentication will not be set. And if authentication is not set, the request will fail.

To set your authentication token, you can use the following command. Be sure to replace with the type of node and with the network that you are running your node on:

```
bash export AUTH_TOKEN = ( celestia
```

```
< node-type
```

```
auth admin --p2p.network
```

```
< network
```

```
) export AUTH_TOKEN = ( celestia
```

```
< node-type
```

```
auth admin --p2p.network
```

```
< network
```

```
) Here's an example of how to set your auth token on a light node on Arabica:
```

```
bash export AUTH_TOKEN = ( celestia light auth admin --p2p.network arabica) export AUTH_TOKEN = ( celestia light auth admin --p2p.network arabica)
```



## Node store

In order to interact with the API using RPC CLI, you can also use your node store to set your auth token. This will allow you to interact with the API without setting an authentication token directly.

To set your node store for a light node on mocha-4, you can use the following command:

`bash export NODE_STORE = HOME /.celestia-light-mocha-4 export NODE_STORE = HOME /.celestia-light-mocha-4` Then, set the `--node.store` flag to the `NODE_STORE` variable to set the auth token from your node store:

```
bash celestia
```

```
< modul e
```

```
< metho d
```

```
    [args...] --node.store NODE_STORE celestia
```

```
< modul e
```

```
< metho d
```

```
    [args...] --node.store NODE_STORE
```

### Auth token on custom or private network

This section is for users who are using a `CELESTIA_CUSTOM` or private network.

#### TIP

If you are using a private and custom network, you will need to set the location of the node store in your auth command.  
`bash --node.store HOME /.celestia-light-private ) --node.store HOME /.celestia-light-private )` The above is an example from the following custom network set up with:

```
bash CELESTIA_CUSTOM = private
```

```
celestia
```

```
light
```

```
init CELESTIA_CUSTOM = private
```

```
celestia
```

```
light
```

```
init or
```

```
bash celestia
```

```
light
```

```
init
```

```
--p2p.network
```

```
private celestia
```

```
light
```

```
init
```

```
--p2p.network
```

private As an example, this is what a completely custom network would look like:

```
bash
```

## Initialize node store

# CELESTIA\_CUSTOM

robusta-22

celestia

light

init

## Set auth token

```
export AUTH_TOKEN = ( celestia light auth admin --p2p.network private \ --node.store
```

```
HOME /.celestia-light-robusta-22)
```

## Initialize node store

# CELESTIA\_CUSTOM

robusta-22

celestia

light

init

## Set auth token

```
export AUTH_TOKEN = ( celestia light auth admin --p2p.network private \ --node.store
```

```
HOME /.celestia-light-robusta-22)
```

### Submitting data

In this example, we will be submitting a blob to the network with [blob.Submit](#) transaction with our light node.

Some things to consider:

- The endpoint takes innamespace
- anddata
- values.\* The commitment will be generated by the node.
- - Share version is set by the node.
- Namespace should be 10 bytes, prefixed by0x
- if hex; otherwise use base64
- Data can be hex-encoded (0x...
- ), base64-encoded ("..."
- ), or a plaintext string which will be encoded to base64 ('Hello There!'
- )
- Optionally, user can provide a gas fee and gas limit.

We use the followingnamespace of0x42690c204d39600fddd3 and thedata value of0x676d .

Here is an example of the format of theblob.Submit transaction:

```
bash celestia
```

```
blob
```

```
submit
```

```
< hex-encoded
```

```
namespace
```

```
< hex-encoded
```

```
data
```

```
\ [optional: fee] [optional: gasLimit] [node store
```

```
|
```

```
auth
```

```
token] celestia
```

```
blob
```

```
submit
```

```
< hex-encoded
```

```
namespace
```

```
< hex-encoded
```

```
data
```

```
\ [optional: fee] [optional: gasLimit] [node store
```

```
|
```

```
auth
```

```
token] We run the following to submit a blob to the network in hexadecimal format:
```

```
bash celestia
```

```
blob
```

```
submit
```

```
0x42690c204d39600fddd3
```

```
0x676d
```

```
\ --node.store NODE_STORE celestia
```

```
blob
```

```
submit
```

```
0x42690c204d39600fddd3
```

```
0x676d
```

```
\ --node.store NODE_STORE We get the following output:
```

```
json { "result" : { "height" : 252607 , "commitment" : "0MFhYKQUi2BU+U1jxPzG7QY2BVV1lb3kiU+zAK7nUiY=" } } { "result" : { "height" : 252607 , "commitment" : "0MFhYKQUi2BU+U1jxPzG7QY2BVV1lb3kiU+zAK7nUiY=" } } We can also use a string of text as the data value, which will be converted to base64. Here is an example of the format:
```

```
bash celestia
```

```
blob
```

```
submit
```

```
< hex-encoded
```

```
namespace
```

```
< 'data'
```

```
\ [optional: fee] [optional: gasLimit] [node store
```

```
|
auth
token] celestia
blob
submit
< hex-encoded
namespace
< 'data'
\ [optional: fee] [optional: gasLimit] [node store
```

```
|
auth
token] And an example to submit "gm" as the plain-text data:
bash celestia
blob
submit
0x42690c204d39600fddd3
'gm'
```

```
--node.store HOME /.celestia-light-mocha-4/ celestia
```

```
blob
submit
0x42690c204d39600fddd3
'gm'
```

```
--node.store HOME /.celestia-light-mocha-4/ Output:
```

```
json { "result" : { "height" : 252614 , "commitment" : "IXg+08HV5RsPF3Lle8PH+B2TUGsGUsBiseflxh6wB5E=" } } { "result" :
{ "height" : 252614 , "commitment" : "IXg+08HV5RsPF3Lle8PH+B2TUGsGUsBiseflxh6wB5E=" } } If you notice from the
above output, it returns a result of 252614 which we will use for the next command. The result corresponds to the height of the
block in which the transaction was included.
```

### Optional: Submit with curl

Refer to the [submitting a blob using curl](#) section.

### Retrieving data

After submitting your PFB transaction, upon success, the node will return the block height for which the PFB transaction was included. You can then use that block height and the namespace ID with which you submitted your PFB transaction to get your message shares (data) returned to you. In this example, the block height we got was 252614 which we will use for the following command. Read more about shares in the [Celestia Specs](#) .

Here is what an example of the format of the get command looks like:

```
bash celestia
blob
get
< block
height
```

< hex-encoded

namespac e

\ < commitment

from

output

abov e

< node

store

|

auth> celestia

blob

get

< block

heigh t

< hex-encoded

namespac e

\ < commitment

from

output

abov e

< node

store

|

auth> Here is an example command to retrieve the data from above, onarabica-11 :

bash celestia

blob

get

252614

0x42690c204d39600fddd3

IXg+08HV5RsPF3Lle8PH+B2TUGsGUsBiseflxh6wB5E=

\ --node.store NODE\_STORE celestia

blob

get

252614

0x42690c204d39600fddd3

IXg+08HV5RsPF3Lle8PH+B2TUGsGUsBiseflxh6wB5E=

\ --node.store NODE\_STORE Will generate the following output:

blob

--node.store NODE\_STORE Which will return:

## Examples

## Check your balance

Let's query our node for the balance of its default account (which is the account associated with the CELESTIA\_NODE\_AUTH\_TOKEN key we generated above):

```
bash celestia
```

```
state
```

```
balance
```

```
--node.store NODE_STORE celestia
```

```
state
```

```
balance
```

```
--node.store NODE_STORE The response will look similar to:
```

```
json { "jsonrpc" : "2.0" , "result" : { "denom" : "utia" , "amount" : "172118057" }, "id" : 1 } { "jsonrpc" : "2.0" , "result" : { "denom" : "utia" , "amount" : "172118057" }, "id" : 1 }
```

## Check the balance of another address

Here is an example of the format of the balance-for-address command:

```
bash celestia
```

```
state
```

```
balance-for-address
```

```
< address s
```

```
\ --node.store NODE_STORE celestia
```

```
state
```

```
balance-for-address
```

```
< address s
```

```
\ --node.store NODE_STORE Let's query our node for the balance of another address:
```

```
bash celestia
```

```
state
```

```
balance-for-address
```

```
celestia10rtd9lhel2cuh6c659l25ync16atcyt37umard
```

```
\ --node.store NODE_STORE celestia
```

```
state
```

```
balance-for-address
```

```
celestia10rtd9lhel2cuh6c659l25ync16atcyt37umard
```

```
\ --node.store NODE_STORE The response will be the balance of the address you queried:
```

```
json { "jsonrpc" : "2.0" , "result" : { "denom" : "utia" , "amount" : "1000000" }, "id" : 1 } { "jsonrpc" : "2.0" , "result" : { "denom" : "utia" , "amount" : "1000000" }, "id" : 1 }
```

## Get your node ID

This is an RPC call in order to get your node's peerId information:

```
bash celestia
```

```
p2p
```



info

```
--node.store NODE_STORE celestia
```

p2p

info

```
--node.store NODE_STORE The node ID is in theID value from the response:
```

```
json { "jsonrpc" : "2.0" , "result" : { "ID" : "12D3KooWFFhCaAqY56oEqY3pLZUdLsv4RYAfVWKATZRepUPdosLp" , "Addrs" :  
[ "/ip4/10.0.0.171/tcp/2121" , "/ip4/10.0.0.171/udp/2121/quic-v1" , "/ip4/71.200.65.106/tcp/25630" ,  
"/ip4/71.200.65.106/udp/25630/quic-v1" , "/ip6:::1/tcp/2121" , "/ip6:::1/udp/2121/quic-v1" ] } , "id" : 1 } { "jsonrpc" : "2.0" ,  
"result" : { "ID" : "12D3KooWFFhCaAqY56oEqY3pLZUdLsv4RYAfVWKATZRepUPdosLp" , "Addrs" : [  
"/ip4/10.0.0.171/tcp/2121" , "/ip4/10.0.0.171/udp/2121/quic-v1" , "/ip4/71.200.65.106/tcp/25630" ,  
"/ip4/71.200.65.106/udp/25630/quic-v1" , "/ip6:::1/tcp/2121" , "/ip6:::1/udp/2121/quic-v1" ] } , "id" : 1 }
```

## Get your account address

This is an RPC call in order to get your node's account address:

```
bash celestia
```

```
state
```

```
account-address
```

```
--node.store NODE_STORE celestia
```

```
state
```

```
account-address
```

```
--node.store NODE_STORE Response:
```

```
json { "jsonrpc" : "2.0" , "result" : "celestia1znk24rh52pgcd9z5x2x42jztjh6raaaphuvrt3" , "id" : 1 } { "jsonrpc" : "2.0" , "result" :  
"celestia1znk24rh52pgcd9z5x2x42jztjh6raaaphuvrt3" , "id" : 1 }
```

## Get block header by height

Here is an example of the format of theGetByHeight command:

```
bash celestia
```

```
header
```

```
get-by-height
```

```
< height
```

```
\ --node.store NODE_STORE celestia
```

```
header
```

```
get-by-height
```

```
< height
```

```
\ --node.store NODE_STORE Now, let's get the block header information.
```

Here we will get the header from Block 1:

```
bash celestia
```

```
header
```

```
get-by-height
```

```
1
```

```
\ --node.store NODE_STORE ````
```

It will output something like this:

```
`` json { "jsonrpc" :  
"2.0", "result" :  
{ "header" :  
{ "version" :  
{ "block" :  
"11", "app" :  
"1" }, "chain_id" :  
"arabica-11", "height" :  
"1", "time" :  
"2023-06-27T13:02:39.741743Z", "last_block_id" :  
{ "hash" :  
"", "parts" :  
{ "total" :  
0 , "hash" :  
"" } }, "last_commit_hash" :  
"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "data_hash" :  
"3D96B7D238E7E0456F6AF8E7CDF0A67BD6CF9C2089ECB559C659DCAA1F880353", "validators_hash" :  
"6363C68770C200FD794445668F9B18F5B1DD1125180D6E8D5AB004F7DD7A0F48", "next_validators_hash" :  
"6363C68770C200FD794445668F9B18F5B1DD1125180D6E8D5AB004F7DD7A0F48", "consensus_hash" :  
"048091BC7DDC283F77BFBF91D73C44DA58C3DF8A9CBC867405D8B7F3DAADA22F", "app_hash" :  
"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "last_results_hash" :  
"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "evidence_hash" :  
"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "proposer_address" :  
"91E04695CF9CF531BC0891E7B1D602B3E8022C86" }, "validator_set" :  
{ "validators" : [ { "address" :  
"91E04695CF9CF531BC0891E7B1D602B3E8022C86", "pub_key" :  
{ "type" :  
"tendermint/PubKeyEd25519", "value" :  
"9aNBAXno1B4X5LR2qY5qWqwrMNOzejkctXwzq9BExsg=" }, "voting_power" :  
"500000000", "proposer_priority" :  
"0" } ], "proposer" :  
{ "address" :  
"91E04695CF9CF531BC0891E7B1D602B3E8022C86", "pub_key" :  
{ "type" :  
"tendermint/PubKeyEd25519", "value" :  
"9aNBAXno1B4X5LR2qY5qWqwrMNOzejkctXwzq9BExsg=" }, "voting_power" :  
"500000000", "proposer_priority" :
```

```

"0" } }, "commit" :
{ "height" :
1 , "round" :
0 , "block_id" :
{ "hash" :
"7A5FABB19713D732D967B1DA84FA0DF5E87A7B62302D783F78743E216C1A3550", "parts" :
{ "total" :
1 , "hash" :
"D85C907CE660878A8203AC74BAA147CCC1F87114B45B568B72AD207B62AFE45E" } }, "signatures" : [ { "block_id_flag"
:
2 , "validator_address" :
"91E04695CF9CF531BC0891E7B1D602B3E8022C86", "timestamp" :
"2023-06-30T08:40:19.299137127Z", "signature" :
"qmaEzrnbgtEXCRYc8pCvGRbS+uMuknlBoRAE4qyE7oSgWCRwBVYS/oPReXQLg9ER1oEY1De4MkWvMjIFnQOOCg=="
} ] }, "dah" :
{ "row_roots" : [
"////////////////////7////////////////////huZWOTTDmD36N1F75A9BshxNIRasCnNpQiWqlhdVHcU" ,
"////////////////////5iieeroHBMfF+sER3JpvROleEJZjbY+TRE0ntADQLL3" ], "column_roots" :
[ "////////////////////7////////////////////huZWOTTDmD36N1F75A9BshxNIRasCnNpQiWqlhdVHcU" ,
"////////////////////5iieeroHBMfF+sER3JpvROleEJZjbY+TRE0ntADQLL3" ] } }, "id" :
1 } celestia
header
get-by-height
1
\ --node.store NODE_STORE ``
It will output something like this:
`` json { "jsonrpc" :
"2.0", "result" :
{ "header" :
{ "version" :
{ "block" :
"11", "app" :
"1" }, "chain_id" :
"arabica-11", "height" :
"1", "time" :
"2023-06-27T13:02:39.741743Z", "last_block_id" :
{ "hash" :
"", "parts" :
{ "total" :
0 , "hash" :

```

```
"" } }, "last_commit_hash" :
"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "data_hash" :
"3D96B7D238E7E0456F6AF8E7CDF0A67BD6CF9C2089ECB559C659DCAA1F880353", "validators_hash" :
"6363C68770C200FD794445668F9B18F5B1DD1125180D6E8D5AB004F7DD7A0F48", "next_validators_hash" :
"6363C68770C200FD794445668F9B18F5B1DD1125180D6E8D5AB004F7DD7A0F48", "consensus_hash" :
"048091BC7DDC283F77BFBF91D73C44DA58C3DF8A9CBC867405D8B7F3DAADA22F", "app_hash" :
"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "last_results_hash" :
"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "evidence_hash" :
"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "proposer_address" :
"91E04695CF9CF531BC0891E7B1D602B3E8022C86" }, "validator_set" :
{ "validators" : [ { "address" :
"91E04695CF9CF531BC0891E7B1D602B3E8022C86", "pub_key" :
{ "type" :
"tendermint/PubKeyEd25519", "value" :
"9aNBAXno1B4X5LR2qY5qWqwrMNOzejkctXwzq9BExsg=" }, "voting_power" :
"500000000", "proposer_priority" :
"0" } }, "proposer" :
{ "address" :
"91E04695CF9CF531BC0891E7B1D602B3E8022C86", "pub_key" :
{ "type" :
"tendermint/PubKeyEd25519", "value" :
"9aNBAXno1B4X5LR2qY5qWqwrMNOzejkctXwzq9BExsg=" }, "voting_power" :
"500000000", "proposer_priority" :
"0" } }, "commit" :
{ "height" :
1 , "round" :
0 , "block_id" :
{ "hash" :
"7A5FABB19713D732D967B1DA84FA0DF5E87A7B62302D783F78743E216C1A3550", "parts" :
{ "total" :
1 , "hash" :
"D85C907CE660878A8203AC74BAA147CCC1F87114B45B568B72AD207B62AFE45E" } }, "signatures" : [ { "block_id_flag" :
2 , "validator_address" :
"91E04695CF9CF531BC0891E7B1D602B3E8022C86", "timestamp" :
"2023-06-30T08:40:19.299137127Z", "signature" :
"qmaEzrnbgtEXCRYc8pCvGRbS+uMuknlBoRAE4qyE7oSgWCRwBVYS/oPReXQLg9ER1oEY1De4MkWwMjIFnQOOCg=="
} ] }, "dah" :
```

```
{ "row_roots" : [
"////////////////////7////////////////////huZWOTTDmD36N1F75A9BshxNIRasCnNpQiWqlhdVHcU" ,
"////////////////////5iieeroHBMfF+sER3JpvROleEJZjbY+TRE0ntADQLL3" ], "column_roots" :
[ "////////////////////7////////////////////huZWOTTDmD36N1F75A9BshxNIRasCnNpQiWqlhdVHcU" ,
"////////////////////5iieeroHBMfF+sER3JpvROleEJZjbY+TRE0ntADQLL3" ] } }, "id" :

1 }
```

### Combined commands

```
bash celestia

share

get-by-namespace

"( celestia header get-by-height 147105

\ --node.store

NODE_STORE

|

jq '.result.dah' -r )"

0x42690c204d39600fddd3

\ --node.store NODE_STORE celestia

share

get-by-namespace

"( celestia header get-by-height 147105

\ --node.store

NODE_STORE

|

jq '.result.dah' -r )"

0x42690c204d39600fddd3

\ --node.store NODE_STORE
```

### Get data availability sampler stats

```
bash celestia

das

sampling-stats

--node.store NODE_STORE celestia

das

sampling-stats

--node.store NODE_STORE
```

### Transfer balance of utia to another account

First, set your address as a variable:

```
bash export ADDRESS = celestia1c425ckmve2489attx022qpc02gxspa29wmh0d export ADDRESS =
celestia1c425ckmve2489attx022qpc02gxspa29wmh0d Then, transfer the amount of tokens that you would like, while
setting the recipient's address, gas fee, and gasLimit. This is what the format will look like:
```

bash celestia

state

transfer ADDRESS < amount

in

uti a

< gas

fee

in

uti a

< gas

fee

in

uti a

\ --node.store NODE\_STORE celestia

state

transfer ADDRESS < amount

in

uti a

< gas

fee

in

uti a

< gas

fee

in

uti a

\ --node.store NODE\_STORE Here is an example, sending 0.1 TIA, with a gas fee of 0.008 TIA, and a gas limit of 0.08:

bash celestia

state

transfer ADDRESS 100000

8000

80000

\ --node.store NODE\_STORE celestia

state

transfer ADDRESS 100000

8000

80000

\ --node.store NODE\_STORE If you'd just like to return the transaction hash, you can use jq:

```
bash celestia
```

```
state
```

```
transfer ADDRESS 100000
```

```
8000
```

```
80000
```

```
--node.store NODE_STORE |
```

```
jq
```

```
.result.txhash celestia
```

```
state
```

```
transfer ADDRESS 100000
```

```
8000
```

```
80000
```

```
--node.store NODE_STORE |
```

```
jq
```

```
.result.txhash
```

### **API version**

To query your node's API version, you can use the following command:

```
bash celestia
```

```
node
```

```
info
```

```
--node.store NODE_STORE celestia
```

```
node
```

```
info
```

```
--node.store NODE_STORE
```

### **Help**

To get help and view the CLI menu, use the following command:

```
bash celestia
```

```
--help celestia
```

--help To view help menu for a specific method, use the following command:

```
bash celestia
```

```
< modul e
```

```
< metho d
```

```
--node.store NODE_STORE --help celestia
```

```
< modul e
```

```
< metho d
```

```
--node.store NODE_STORE --help
```

## Advanced example

This example shows us using thejq command to parse the output of thecelestia header get-by-height method to get the extended header used incelestia share get-by-namespace :

```
bash celestia
```

```
share
```

```
get-by-namespace
```

```
\ "( celestia header get-by-height 252614
```

```
--node.store
```

```
NODE_STORE
```

```
|
```

```
jq '.result.dah' -r )" 
```

```
\ 0x42690c204d39600fddd3
```

```
--node.store NODE_STORE celestia
```

```
share
```

```
get-by-namespace
```

```
\ "( celestia header get-by-height 252614
```

```
--node.store
```

```
NODE_STORE
```

```
|
```

```
jq '.result.dah' -r )" 
```

```
\ 0x42690c204d39600fddd3
```

```
--node.store NODE_STORE
```

## Additional resources

### Submitting a blob using curl

In order to post a blob using curl, you will need a light node running with the--core.ip string flag, providing access to a consensus endpoint. The flag indicates node to connect to the given core consensus node. Examples:127.0.0.1 orsubdomain.domain.tld . Using either IP or DNS assumes RPC port 26657 and gRPC port 9090 as default unless otherwise specified.

1. In your terminal, set the auth token for the desired network. In this example, we will use Mainnet Beta.

```
bash export CELESTIA_NODE_AUTH_TOKEN = ( celestia light auth admin --p2p.network celestia) export  
CELESTIA_NODE_AUTH_TOKEN = ( celestia light auth admin --p2p.network celestia) 1. Post your blob with:
```

```
bash curl
```

```
-H
```

```
"Content-Type: application/json"
```

```
-H
```

```
"Authorization: Bearer CELESTIA_NODE_AUTH_TOKEN "
```

```
-X
```

```
POST
```

```
--data
```



sh "rpc error: code = NotFound desc = account celestia1krkle0n547u0znz3unnln8paf2d4qz3rznv86 not found" "rpc error: code = NotFound desc = account celestia1krkle0n547u0znz3unnln8paf2d4qz3rznv86 not found" It is possible that the account you are trying to submit aPayForBlobs from doesn't have testnet tokens yet. Ensure the testnet faucet has funded your account with tokens and then try again. [\[Edit this page on GitHub\]](#) Last updated: [Previous page](#) [Node API](#) [Next page](#) [Prompt Scavenger](#) [\[ \]](#)