Version Notes

This eBook is updated from time to time as more data develops and a more refined version of the project's goals becomes clear. We shall try to keep archived copies as time progresses (*starting with version 1.0.0*)

Smartnode Setup Guide: v1.0

Created: June 5th, 2025

Intended distribution models include:

- Self Hosted ePub PDF
- Apple Bookstore
- Amazon Bookstore (They make us charge $0.99)
- Draft2Digital

Note about Amazon Marketplace

Amazon Kindle Direct Publishing (KDP) will not let us give away this whitepapaer, or any ePub free of charge. as such all royalties from the sale of this book on Amazon Marketplace (or others in the future if they force us to sell for more than $0.00 USD shall annually donated to St. Judes Child Cancer Research Center

Things to Consider

This guide assumes you will be running the Smartnode on a Linux-based system. While it is technically possible to use Windows, I have observed increasing issues with Smartnodes running on Windows. It simply does not meet the typical requirements of a reliable Smartnode setup.

I cannot, in good faith, recommend installing a Smartnode on a home internet connection—even if you have business-class service. In my experience supporting this software and other Ghostrider coins (i.e., DASH derivatives), home internet tends to suffer from uptime issues. As a result, you are more likely to receive high `POS_BAN` scores. We strongly encourage you to use a VPS. Our support channel on Discord can provide recommendations.

Although it is technically possible to install the node under the `root` user, this is not advisable. The recommended approach is to follow this user structure:

Users

- root
- sudouser
- bitoreum

root – A system-level, protected user. This is the "big boss" account, with unrestricted access to do anything on the system, including irreversible actions like deleting critical data. Use this user only if you absolutely know what you are doing.

sudouser – A user account with elevated (sudo) privileges. This is the preferred method for executing administrative tasks, rather than using the root account directly. While this user is capable of installing the Smartnode, doing so can be risky—if compromised, it can still harm the system. You may name this user whatever you like, unlike the reserved `root`.

bitoreum – This is the preferred non-privileged user account under which the Smartnode will be installed. Since it lacks elevated privileges, it is relatively safe, even if compromised.

Oracle VPS vs. Everyone Else

I always recommend Oracle VPS over other providers. With a bit of understanding of their Always Free tier, it's a solid choice for anyone wanting to try out a Smartnode without a significant financial commitment. However, you will need a valid credit card for verification; prepaid debit or credit cards are not accepted.

Keep in mind that Oracle is not designed for entry-level users, hobbyists, or even small businesses. Their platform primarily serves enterprise customers. Concepts like protected instances, internal port mapping, software firewall exclusions, console usage, and mandatory SSH key login can make Oracle intimidating, even for intermediate users. That said, it's still free (at the time of writing), which is a compelling advantage.

Finally, Oracle VPS often does not support common firewall tools like `firewalld` or `ufw`. It is strongly encouraged to use `iptables`, which this guide recommends.

Resources

The absolute bare minimum VPS instance is a single-core, 1GB RAM setup, with a minimum 2GB `swapfile`. While this configuration will work, it will likely struggle during the initial sync and may only be capable of supporting one other low-resource Ghostrider coin.

A good, safe baseline is: 1–2GB RAM, 25GB SSD, 1Gbps network bandwidth, with Ubuntu 22.04 or 24.04 (*recommended*) installed. In general, the more resources, the better.

For Oracle, I recommend a single Ampere core (from the Always Free tier) with 6GB of RAM and 47GB of high-speed boot volume block storage. This configuration uses the ARM architecture, so make sure to match the

architecture to the Crystal Bitoreum wallet. Oracle also offers an x86 instance type, but it is limited to 2 cores and a maximum of 1GB RAM—not ideal, but workable.

VULTR (affiliate link) offers a Cloud Compute instance for about $6 USD per month. It includes a single core, 1GB RAM, and both an IPv4 and IPv6 address. This setup can handle one Smartnode—or possibly one other Ghostrider coin—as long as you add a 2–4GB `swapfile`.

RackNerd (non-affiliate link) frequently runs great deals throughout the year. During the Christmas holiday season, I secured a long-term service VPS for about $29.99 per year—4 cores and 4GB RAM for life. Their deals fluctuate, so it's worth checking regularly.

IONOS (non-affiliate link) is an emerging low-cost VPS provider. I don't have enough direct experience to offer a recommendation, but they've been favorably mentioned by others—take that as you will.

And finally, the elephant in the room: Hetzner (non-affiliate link). Officially, they prohibit the use of their VPS services for anything related to cryptocurrency. The original Bitoreum lead developer was banned for running a TRON wallet, and I was personally removed for running both a Bitoreum and Yerbas Smartnode. Since their position has not changed, I cannot recommend them at this time—although they remain inexpensive for non-US customers.US customers be advised there are tarrifs that you will be responsible for as of the time of writing.

Connecting for the First Time

Your VPS instance provider will give you an IPv4 address, a username (`root` or `ubuntu`, typically), and either a password or an SSH key. To connect with a password, open the `terminal` application and use the following format:

```
ssh usernameProvided@ip#.ip#.ip#.ip#
```

The system will prompt you for a password. Enter the provided password, and you should see a bash prompt like this:

```
username@server-hostname:~$
```

To log in using an SSH key file, there are a few important considerations. Windows doesn't enforce file permissions, but Linux and macOS do. First, you need to `chmod` the key file before you can use it properly:

```
chmod 600 /path/to/ssh.key
```

Type `chmod 600` first, then drag and drop the key file into the terminal to auto-fill the path. Once done, the SSH command is ready:

```
ssh -i /path/to/ssh.key usernameProvided@ip#.ip#.ip#.ip#
```

Using an SSH key allows you to log in without entering a password, as the key file serves as your private authentication. **DO NOT SHARE OR REVEAL THIS FILE TO OTHERS!**

First Boot

Depending on your instance, your first login will be as either `root` or `ubuntu`. The first thing we'll do is create a new sudo-enabled user:

```
sudo adduser sudouser
```

The system will prompt you to enter the password twice—ensure they match and are secure but memorable. It will then ask for details like full name, phone, etc. These fields are less important. When prompted to confirm the details with `[Y/n]`, type `Y` to finish. Example:

```
info: Adding user `sudouser' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `sudouser' (1006) ...
info: Adding new user `sudouser' (1006) with group `sudouser
(1006)' ...
info: Creating home directory `/home/sudouser' ...
info: Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for sudouser
Enter the new value, or press ENTER for the default
    Full Name []: Super User for the Smartnode VPS
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] Y
info: Adding new user `sudouser' to supplemental / extra groups
`users' ...
info: Adding user `sudouser' to group `users' ...
```

Now, add the user to the `sudo` group:

```
sudo adduser sudouser sudo
```

This will return:

```
info: Adding user `sudouser' to group `sudo' ...
```

Next, create the `bitoreum` user the same way, but do **not** add this user to the `sudo` group. Example:

```
sudo adduser bitoreum

info: Adding user `bitoreum' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `bitoreum' (1008) ...
info: Adding new user `bitoreum' (1008) with group `bitoreum
(1008)' ...
info: Creating home directory `/home/bitoreum' ...
info: Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for bitoreum
Enter the new value, or press ENTER for the default
    Full Name []: Bitoreum Smartnode
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] Y
info: Adding new user `bitoreum' to supplemental / extra groups
`users' ...
info: Adding user `bitoreum' to group `users' ...
```

Do not add the `bitoreum` user to the `sudo` group. The following section is for users continuing with SSH key-based login (recommended for better security).

SSH-Keygen

The easiest way to switch users is via `root` access. If you're not already root, run:

```
sudo bash
su sudouser
ssh-keygen -t ed25519 -a 100 -C "your.email@address.com"
```

Example output:

```
Generating public/private ed25519 key pair.
Enter file in which to save the key
(/home/sudouser/.ssh/id_ed25519):
Created directory '/home/sudouser/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in
/home/sudouser/.ssh/id_ed25519
Your public key has been saved in
/home/sudouser/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:Yh6TN9wEwAsjBFAdt40h5dYcKCCeg71HAiR9oa1Z3jc
your.email@address.com
The key's randomart image is:
+--[ED25519 256]--+
|O*+o+==oo        |
|+++o*+oB o       |
|.++o++=.+ .      |
|    .O o.o o     |
| + o B E .       |
| . o * o         |
|  .              |
|                 |
|                 |
+----[SHA256]-----+
```

Next, change to the new (hidden) .ssh folder:

```
cd ~/.ssh
ls
```

Expected output:

```
sudouser@bitoreum-smartnode:~/.ssh$
id_ed25519   id_ed25519.pub
```

Ubuntu 22.04 and 24.04 use the `authorized_keys` system. To set it up, run:

```
mv id_ed25519.pub authorized_keys
cat id_ed25519
```

Output example:

```
-----BEGIN OPENSSH PRIVATE KEY-----
someRandomtext&numbers0123456789
someRandomtext&numbers0123456789
someRandomtext&numbers0123456789
someRandomtext&numbers0123456789
someRandomtext&numbers0123456789
-----END OPENSSH PRIVATE KEY-----
```

This is the private key you will use to access your VPS. Create a new file called `bitoreum_Smartnode_sudouser.key` on your local machine and paste this exact content. Then, secure it with:

```
chmod 600 /path/to/bitoreum_Smartnode_sudouser.key
```

Back on the instance remove the private key from the server:

```
rm id_ed25519
```

Congratulations! You now have a fully secured SSH key that enables safe remote login to your `sudouser` account on your VPS! Now to test the secured connection with the new keyon you local machine use your `terminal` app:

```
ssh -i /path/to/bitoreum_Smartnode_sudouser.key
sudouser@ip#.ip#.ip#.ip#
```

You should be greated with the following prompt, or something similar:

```
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-1026-oracle
aarch64)
```

```
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/pro

This system has been minimized by removing packages and content
that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sat Jun 7 19:15:45 2025 from your.origin.ip4.number

sudouser@bitoreumsmartnode:~$
```

While you can create an SSH key for the `bitoreum` user, it is largely unnecessary. Once logged in, you can always switch users using the command:

```
su <username>
```

You will however, need the respective user's password to proceed.

Oracle

Oracle often has towo major issues that we have to overcome. First you have to log into your Oracle Console and navigate to Virtual Cloud Network > VNC Prefix > security > Security List > Security Rules -> Add Ingress Rule:

```
Stateless: off
Source Type: CIDR
Source CIDR: 0.0.0.0/0
IP Protocol: TCP
Source Port Range:
Destination Port Range: 15168
Description: Bitoreum Network
```

Save or Add Ingress rule, then switch back over to your VPS because now we need to modify the IP tables:

```
sudo nano /etc/iptables/rules.v4
```

Look for the following line:

```
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
```

// and directly below it enter

```
-A INPUT -p tcp -m state --state NEW -m tcp --dport 15168 -j ACCEPT
CNTL+O
CNTL+X
sudo iptables-restore </etc/iptables/rules.v4
```

This applies the new IPv4 `iptable` rules, in some cases you might have to reboot the instance, particulatly if this is the first time adding an ingress rule

on the Oracle Console.

Every Other VPS

If `ufw` is not installed then install it first by:

```
sudo apt install ufw -y #If already installed ignore this
ufw default deny incoming
ufw default allow outgoing
ufw allow ssh
ufw allow 15168/tcp
ufw enable
```

We now have a working firewall either route you are taking. Look you could just use iptables for all instances... but whyyyyyy?

Update & Upgrade Your Instance

Even though we now have a fresh and clean install, we still need to update the repositories, preinstalled packages, and the OS itself to ensure we're running the best version of the instance. This may sound complicated, but it's largely an automated process:

```
sudo apt update && sudo apt dist-upgrade -y
```

There is always some controversy around using the `-y` flag, which automatically answers "yes" to all prompts. In this specific use case, if `-y` breaks your system, there are likely bigger issues at play. For our limited scope and usage, it should be perfectly fine.

Next, we will install the software we're likely to need throughout the rest of this guide. Once again, whether or not you use the `-y` flag is up to you:

```
sudo apt update && sudo apt install nano unzip fail2ban wget
net-tools -y
```

Temporarily Suppress `sudo` Password

For the remainder of this project, we'll be calling commands that require elevated `sudo` privileges. Entering the password repeatedly will become tedious, so we'll temporarily suppress the password requirement. **DO NOT** leave this suppression in place once you're done—doing so is like leaving your car unlocked, running, and unattended.

```
sudo visudo
```

Now scroll to the bottom of the file and look for the following line:

```
# See sudoers(5) for more information on "@include" directives:

@includedir /etc/sudoers.d

# See sudoers(5) for more information on "@include" directives:

@includedir /etc/sudoers.d
```

Add the following line directly below:

```
sudouser ALL=(ALL) NOPASSWD:ALL
```

The updated section should now look like this:

```
# See sudoers(5) for more information on "@include" directives:
@includedir /etc/sudoers.d
sudouser ALL=(ALL) NOPASSWD:ALL
```

To resume requiring a password for elevated privileges, simply comment out the line by prefixing it with `#`. The final section of the `visudo` file should then look like this:

```
# See sudoers(5) for more information on "@include" directives:
@includedir /etc/sudoers.d
# sudouser ALL=(ALL) NOPASSWD:ALL
```

Press `CTRL+O` to save the file, then `CTRL+X` to cleanly exit `visudo`.

Adding SWAP

Adding swap is always a good idea especially to lower resoure instances. First we are going to do this from root, as it has less of a chance of failure or misconfiguration. First if you have swap already set up (some instances do) you can check for it :

```
free -h
              total
```

```
Mem:           1Gi
Swap:          0Gi
```

This means there is no swap present and only 1 GB of physical RAM, lets help this system out a bit:

```
sudo bash
dd if=/dev/zero of=/swapfile bs=1k count=2048k
chmod 600 /swapfile
mkswap /swapfile
swapon /swapfile
echo "/swapfile    swap    swap    auto    0 0"| tee -a
/etc/fstab
sysctl -w vm.swappiness=10
echo vm.swappiness = 10 | tee -a /etc/sysctl.conf
exit
```

This returns us back to the previous user.

Note: swappiness = 10 tells system only to use it if absolutely needed...

Prevent Brute Force Attacks

if given an infinate amount of time and chances a hacker can brute force their way into any system no matter how secure. Earlier we installed a piece software that will help prevnet this called `fail2ban`. Its installed and active but now we need to configure it.

```
sudo nano /etc/fail2ban/jail.local
```

Past the following:

```
[sshd]
enabled = true
port = 22
filter = sshd
logpath = /var/log/auth.log
maxretry = 3
```

Save and close this file using

`CNTL+O`

`CNTL+X`

Now restart the service:

```
sudo systemctl restart fail2ban
```

you now have a hardened and secure VPS, after 3 failed login attempts and fail2ban will blacklist the offending IP. So be carefule not to enter your own password wrong too many times. You can force reboot the instance to clear fail2ban's memory in case this happens

Install & Configure Crystal Bitoreum

At the time of writing, Crystal Bitoreum v4.0.0.0 is not publicly available, but v3.0.2 is. Even with its known flaws, v3.0.2 will work fine for our purposes. Assuming you are still operating as the `sudouser`, return to your home directory with:

```
cd
```

We need the appropriate version of the core wallet to match your system. Use the command below that corresponds to your instance type:

Ampere (Oracle) / ARM:

```
wget
https://github.com/bitoreum/bitoreum/releases/download/v3.0.2/b
itoreum-arm-oracle-ubuntu20-3.0.2.01.tar.gz
```

x86:

```
wget
https://github.com/bitoreum/bitoreum/releases/download/v3.0.2/b
itoreum-3.0.2.jammy.x86_64.tar.gz
```

Next, we need the most recent known bootstrap. It os still quite old, and a newer one will be released with v4.0.0.0. For now, use the following:

```
wget
https://github.com/bitoreum/bitoreum/releases/download/v3.0.2/b
ootstrap-16feb2023.zip
```

We are going to globally install the Bitoreum Core so first we either:

```
tar -xvf bitoreum-arm-oracle-ubuntu20-3.0.2.01.tar.gz
```

or

```
tar -xvf bitoreum-3.0.2.jammy.x86_64.tar.gz
```

What you are left with is a folder called `bitoreum-build`:

```
cd bitoreum-build && sudo cp bitoreumd bitoreum-cli /usr/bin
```

Now any user can run the core without refferencing an absoulte path such as:

```
bitoreumd
bitoreum-cli [function]
```

Now we need to start configuring the bitoreum user Bitoreum/Crystal Bitoreum keep core files natively in a hiden folder called .bitoreumcore, we create the directory, propigate it with the bootstrap, and make the last two files we need:

```
sudo unzip bootstrap-16feb2023.zip -d
/home/bitoreum/.bitoreumcore
sudo touch /home/bitoreum/.bitoreumcore/debug.log
sudo nano/home/bitoreum/.bitoreumcore/bitoreum.conf
```

Now we are in the conf file that controls the function of the Crystal/Bitorem Smartnode here is a template:

```
#ProTXHash=
#SmartnodePublicKey=
# Basic settings
server=1
daemon=1
listen=1
rpcbind=127.0.0.1
rpcallowip=127.0.0.1
```

```
# RPC credentials
rpcuser=randomlettERSaNDNumbers0995995
rpcpassword=randomlettERSaNDNumbers0995995

# Listen and port settings
rpcport=8900 // Do Not duplicate this port use elsewhere

#Smartnode Settings
smartnodeblsprivkey=

#Define IP
externalip=x.x.x.x:15168 // Public IP provided by your VPS
provider

#Preffered Nodeodes, speeds up sync process
addnode=116.203.130.246:15168
addnode=146.235.232.125:15168
addnode=159.69.111.55:15168
addnode=45.32.230.238:15168
addnode=188.34.194.136:15168
addnode=137.131.52.195:15168
```

There are some lines missing notably:

```
#ProTXHash=
#SmartnodePublicKey=
smartnodeblsprivkey=
```

Ignore the ProTXHash line for now, we will come back to that later. The public and private key can be obtained from a running core, either from the GUI, launch **'tools > debug console'** and type:

```
bls generate
```

or from command line:

```
bitoreum-cli bls generate
```

Either way, the output will look like:

```
{
"secret": "uniqekeyrandomstringofchars", // smartnodeblsprivkey
```

```
"public": "uniqekeyrandomstringofchars" // SmartnodePublicKey
}
```

Add these in the appropriateplaces in the conf and then we save and exit the file:

`CNTL+O`

`CNTL+X`

Finally we need to return ownership of all these thinngs we have unziped and created back to the `bitoreum` user:

```
sudo chown -R bitoreum:bitoreum /home/bitoreum
```

Crystal Bitoreum As A `systemd` Service

this is kind of unneeded and can be achived with a `cron` job, but systemd allows for rebooting the instance in case it crashes, which we can test later. Make sure we are logged in as the `sudouser` user.

```
sudo nano /etc/systemd/system/bitoreum.service
```

Copy and paste the following text:

```
################################################################
###
## Bitoreum Smartnode Systemd service alway alive ##
################################################################
###
[Unit]
Description=Bitoreum Node Daemon
After=network.target

[Service]
Type=forking
User=bitoreum
RuntimeDirectory=/home/bitoreum/.bitoreumcore
#PIDFile=/run/bitoreumd.pid
Restart=on-failure
```

```
ExecStart=/usr/bin/bitoreumd \
   -datadir=/home/bitoreum/.bitoreumcore/ \
   -conf=/home/bitoreum/.bitoreumcore/bitoreum.conf \
   -daemon
ExecStop=/usr/bin/bitoreum-cli \
   -datadir=/home/bitoreum/.bitoreumcore/ \
   -conf=/home/bitoreum/.bitoreumcore/bitoreum.conf \
    stop

# Recommended hardening

# Provide a private /tmp and /var/tmp.
PrivateTmp=true

# Mount /usr, /boot/ and /etc read-only for the process.
ProtectSystem=full

# Disallow the process and all of its children to gain
# new privileges through execve().
NoNewPrivileges=true

# Use a new /dev namespace only populated with API pseudo
devices
# such as /dev/null, /dev/zero and /dev/random.
PrivateDevices=true

# Deny the creation of writable and executable memory mappings.
MemoryDenyWriteExecute=true

[Install]
WantedBy=multi-user.target
```

CTRL+O
CTRL+X

```
echo "pid=/run/bitoreum.pid" >>
/home/bitoreum/.bitoreumcore/bitoreum.conf
```

```
sudo systemctl daemon-reload
sudo systemctl enable bitoreum
sudo systemctl start bitoreum && tail -f
/home/bitoreum/.bitoreumcore/debug.log
```

This will install a `systemd` service called `bitoreum`, which launches the `bitoreumd` daemon under the user named `bitoreum`. If `bitoreumd` crashes under that user, it will automatically attempt to relaunch—again and again. One important thing to note: with this setup, you can't stop the core using the traditional method.

Before we continue, monitor the `tail` log for errors and to confirm that the node is syncing with other Smartnodes. It may take a few minutes for the peers list to propagate and begin pulling data from the network. As mentioned earlier, the most recent bootstrap is quite old and starts far behind the current chain height, but once peers are connected, the sync process typically accelerates quickly.

To test this service and simulate a crash state, start by switching to the `bitoreum` user while the service is running:

```
su bitoreum
cd ~/.bitoreumcore
bitoreum-cli stop && tail -f debug.log
```

This simulates a crash state. You'll see `bitoreumd` shut down and immediately restart. This means you should not use `bitoreum-cli stop` directly—it will simply trigger a restart loop. Instead, use the following commands to manage the service:

```
sudo systemctl stop bitoreum
sudo systemctl start bitoreum
```

There are two major benefits to this setup:

1. Automatic recovery from a crash without intervention.
2. Automatic startup on system reboot—unless you disable it.

If you ever need to disable autostart on reboot, run:

```
sudo systemctl disable bitoreum
```

Collateral & Registration

Now that our Smartnode is synced and running in the background, we need to set up the collateral in the core wallet. If this is your first time using a core wallet derived from DASH, you'll need to enable certain features. Under the **Settings** menu, select **Options**. In the popup, go to the **Wallet** tab. The two critical settings we need to enable are: `Enable Coin Control Features` and `Show Smartnodes Tab`. Enabling both will trigger a warning to restart your wallet. Go ahead and close and reopen it. After the restart, you should see an `Inputs` button in the Send tab and a new Smartnodes tab available.

If you have never encrypted your wallet, do so now and set a password that is both secure and easy to remember. This action will also prompt another wallet restart.

Now, open the debug console from the **Tools** menu. To perform the next steps, your wallet must be unlocked. In the console input field, type:

```
walletpassphrase "youractualpasswordhere" 100000
```

The number is the duration (in seconds) the wallet will remain unlocked if the password is correct. Leaving it blank is the same as entering zero, which immediately re-locks the wallet. `100000` seconds is approximately 1.16 days. Once that time expires, the wallet automatically relocks and a password will be required again. You can manually lock it anytime by entering:

```
walletlock
```

With your wallet unlocked, go to the **Receive** tab. In the `Label` field, type `Collateral 01` and click the **Request Payment** button. This creates a new address labeled **Collateral 01** in your wallet. Now send exactly **1,800,000 $BTRM** to this address. **WARNING:** Make sure the checkbox labeled `Subtract fee from amount` is **unchecked**—the amount must be exactly 1800000 (1,800,000 sylized) $BTRM or the registration process will fail.

A popup window with a three-second delay on the `OK` button will show the transaction details. Nothing is sent until you click `OK`, and you can cancel by clicking `cancel` at any time.

After confirming, the wallet will take you to the **Transactions** screen, where you can monitor the transaction as it propagates across the network. Wait until the transaction receives at least one network confirmation before continuing.

At one Confirmation on the transaction alt mouse click on it to bring up a menu and we are looking for an option called `Copy Transaction ID`. Paste this into a text document as we need to assemble a fair amount of data.

As a child project of DASH coin we can use the EXACT command stucture for creating a masternde on DASH which are three commands

The first command is the ProTXRegister:

```
protx register_prepare "collateralHash" collateralIndex
"ipAndPort" "ownerKeyAddr" "operatorPubKey" "votingKeyAddr"
operatorReward "payoutAddress" "feeSourceAddress"
```

Second we need to sign the transaction:

```
signmessage "collateralAddress" "signMessage"
```

Then we need to submit the singed transaction:

```
protx register_submit "tx" "sig"
```

Starting with the first command the Transaction ID previously saved is the `collateralHash`. To get the `collateralIndex` head back into the wallet Debug Console and type `smartnode outputs`, the easiest way to confirm if you have multiple nodes is to match the last 4 of the collateralHash to the

line displayed, what you are looking for is the number to the right of the hash, its either going to be a 1 or zero.

For the `"ownerKeyAddr"` & `"votingKeyAddr"` in the Debug Console issue this command twice `getnewaddress`.

for the `operatorPubKey` remmeber we put it in the biroreum.conf file on the node VPS. From a the `sudouser` issue this command:

```
sudo cat /home/bitoreum/.bitoreumcore/bitoreum.conf
```

We are looking for the value we saved in the line:

```
#SmartnodePublicKey=somekeyvaluehere
```

Unless you are splitting a node's payouts with a friend or someone who gives you part of the collateral the `operatorReward` is going to be zero (`0`)

For the `payoutAddress` We are going to go back to the Wallet GUI and go back to the Recieve tab, In the `Label Field` lets call this Smartnode Payouts

Finally for the `(feeSourceAddress)`, go back into the debug console again and type`listaddressbalances` the list that propigates, we are looking for any balance that IS NOT our 1800000, remmeber we need that clean, it can be any other address with even tiny amounts of $BTRM we just need it to cover transaction fees which are tiny

With the first command now completed, head back to the debug console and past the whole completed command into the input field and submit (wallet needs to be unlocked). If the transaction is valid the consol will spit out a bunch of text tht is critical, copt the whole output and pzste it into a txt file. For the `"collateralAddress"` you are looking for that address in the output you got from the previous command. FOr the `"signMessage"` we are looking for that line as well it will be filed with a bunch of seemingly random text but we need the whole bit between the two " marks. Take this completed command and input it into the debug console. This will spit out a single line, take note of it, copy and past it into said txt file

For the last command we need the `"tx"` from the output of the first command. The `"sig"` is the single line from the previous output including

the = sign. Once you input this completed command into the debug console a few things should happen. One the collateral should now be locked to its address and unspendable, this is required if you want to recieve payments. Two a new transaction shows up in your transaction screen, once it gets to one confirmation your new Smartnode should now show up in the Smartnode tab

Congradulations, you should now have a fully functional Smartnode on the Crystal Bitoreum Network

Contacts

- Website: bitoreum.cc
- Explorer: explorer.bitoreum.cc
- Github: Crystal Bitoreum
- Discord: Crystal Bitoreum Network
- Bluesky: Ramen Wukong
- YouTube: @CrystalBitoreum

Useful Links

- Bitoreum Explorer: Coming Soon...
- Bitoreum Pools: Mining Pool Stats
- Recommended Exchange: TradeOgre
- Recommended Exchange: XeggeX wallet currently offline working with them to restore
- Coin Aggregator: IoVa Systems
- Coin Aggregator: CoinGecko
- Coin Aggregator: CoinPaprika
- Coin Aggregator: AltBuster
- Coin Aggregator: CoinCodex

Special thanks to everyone who has contributed to making the project, and a big shout-out to Marius. Without you, we wouldn't be here, for better or worse. You are missed, buddy, and I do hope you are alright somewhere...

Table of Contents