101 Dogecoin Tricks

chromatic

Contents

Preface

Dogecoin is the Internet's original meme-based cryptocurrency.

That sentence may read like a bunch of nerdy nonsense to you. Don't worry; it is! Fortunately, this book will make it clearer. It'll still be a bunch of nerdy nonsense, but you'll understand it and be able to have fun, do cool things, and show off your new knowledge to friends at parties[1].

Let's explain that first sentence.

Dogecoin is what we're talking about.

The Internet is a globally-distributed network of computers that lets people and machines communicate in words, pictures, video, audio, and all kinds of data.

A meme is an idea, usually clever or ironic or sarcastic or joking, that's easily spread between people. It often takes the form of an image or video that can be shared on the Internet.

Cryptocurrency is a mechanism to record digital transactions over the Internet between people who don't necessarily trust each other as individuals but, as a group, agree to specific ways to communicate that ideally make the entire network of people trustworthy enough to serve as a mechanism of financial exchange.

Whew.

The crypto stuff is interesting, and there's a lot to learn if you want, but most of it summarizes to this: you don't have to take someone's word that they're trustworthy. You should be able to verify it.

If that sounds like a lot of work with a bunch of serious nerds doing serious nerd stuff, remember that there's a friendly dog mascot with a bunch of silly memes and the whole point is to have fun.

That's what this book is about.

---

[1] Talk about other things too; we want you invited *back*.

What do I need?

This book assumes you have access to a computer somewhere (though some of the tricks will work on mobile devices such as phones and tablets), have access to the Internet, and can download and install and run software.

At the time of this writing, you can visit dogecoin.com for links and information about how to download and run the official Dogecoin Core. Don't take our word for it though; check online if that's still the official home of the official Core. Test that against other sites such as https://github.com/dogecoin/dogecoin, the Dogecoin Reddit at https://reddit.com/r/dogecoin/ and other locations. If something seems off to you ask questions.

Credits

This work, when finished, will turn out not to have been possible without the assistance of many people. You will find credits listed here.

Any remaining errors are the fault of the sometimes-too-clever author.

CHAPTER 1

Running Your Own Node

In this chapter, we'll talk about interesting things you can do when you run your own node.

Tip #1 Set Your Node Comment

If you look at the network peers tab in your core wallet (Help -> Debug -> Peers, in the Qt GUI as of 1.14.6), you'll see a list of other nodes you've connected to. This list shows things like their IP address (IPv4 or IPv6), the amount of data sent and received, and more.

That additional data includes a node version string. By default (it's in the source code defined as a value called `CLIENT_NODE`; I just checked), every Dogecoin Core reports itself as `Shibetoshi/version`. This goes the same whether you're running `dogecoind` as a headless network service or `dogecoin-qt` as a GUI application or some other way of running the Core that we haven't invented yet.

The Core allows you to *append* text to this version string to customize your node and, potentially, brighten someone's day.

This is a feature called `uacomment`.

Setting `uacomment`

To customize this feature, you need to edit your configuration file and add an entry:

```
uacomment=My Cool Node
```

Of course, you can and should pick something more interesting than that. As it currently stands, you have 256 characters to play with here[1].

Save this file, then launch or restart your node. If you're running the Qt GUI, look at the debug information window under “User Agent” (Help -> Debug -> Information). The text you see there will be broadcast across the entire Dogecoin network whenever your node makes a connection, so this is your chance to make a mark and do something meaningful.

Setting a *Secure* UA Comment

Of course, this is your node broadcasting something potentially specific to you, so remember that you're broadcasting it to an entire Internet full of people, some of whom may not have your best interests at heart.

When you're choosing what to say, keep at least two important points in mind:

---

[1] This is defined as `MAX_SUBVERSION_LENGTH` in the source code. Check the source for the version you're running to see if this has changed.

- Be Kind
- Be Safe

If you're thinking of setting a comment that might hurt someone's feelings or cause drama like “Patrick is a curmudgeon![2]“, think twice. There are human beings on the other side of the Internet, and our funny dog money only works if we all work together and cooperate.

If you're thinking of setting a comment that might reveal personal or private information about you or someone else, like “Proudly hosted at 1 Chome-2 Dōgenzaka, Shibuya City”, think twice, then twice more. Personal information can make you or anyone else a target, and a visit in person or electronically from someone who doesn't have everyone's best interest at heart goes against the spirit of fun and collaboration.

What makes a *good* comment? If you follow those three rules (the other rule is the 256 character limit), then anything goes.

You could use a dopey joke, like “Why does the Swedish Chef love Dogecoin? Because it goes bork bork bork!”

You could advertise your Doge-related project, like “Check out IfDogeThenWow!”

You could brag a little about your uptime, like “Proudly serving the network since 2013.”

The sky is the limit, as long as you're smart about it.

---

[2] He'll probably agree, so this is the worst thing I wanted to put in print.

CHAPTER 2

Network Services with Dogecoin

Dogecoin is more than just a network of nodes. It's a low-friction payment and tipping system that can integrate with the wider Internet. This chapter explores several ways to do so, both as a publisher and a consumer.

Tip #2 Add a Wallet Address to a DNS Record

If you have a website, like https://ifdogethenwow.com/, how can people find you and send you a tip or a payment for something awesome you've done?

You need some way to associate a Dogecoin address with an Internet property.

Fortunately, the Internet already has a well-understood mechanism to add this kind of metadata to a domain name. It's the same way any device can turn that domain name into an IP address or addresses to look up a web page, send email, make an SSH connection, or do other things.

At the inagural Dogecoin hackathon[1], Timothy Stebbing showed off this idea.

Here's how to make it work.

How to Add a TXT Record

First, generate a new, unique Dogecoin address. Use the Dogecoin Core, `libdogecoin`, or a non-custodial wallet you trust. Keep the private key safe, as always.

Second, take a domain name you control. Maybe that's `mycoolsite.example` or something else. You've registered this through a domain registrar, and you have the ability to configure DNS.

Keep Your Domain Running!

If you didn't set up your own DNS, talk to the person or people who set it up for you. If you edit this on your own and make a mistake, you could break your website, email, or other network services associated with that domain.

Now go into your DNS configuration and add a ‘TXT‘ record. This is an arbitrary text string associated with your domain name and propagated through the globally-accessible, cached DNS system.

For now, stick with the defaults of *which* DNS entry, TTL settings, et cetera.

The contents of this new record should be the literal string ‘dogecoin:‘ (yes, include the colon) plus the new wallet address you generated, with no spaces in

[1] https://foundation.dogecoin.com/announcements/2022-09-08-dogeathon-downunder/

between. This conforms to BIP 21[2], so other applications can use this. Hold onto that thought.

Save the configuration and publish it and you're good to go.

Specific DNS Configuration Details See... Elsewhere

Looking for a visual explanation of how to do this? The process varies depending on your domain registrar and DNS host. Again, this has the potential to render your domain *temporarily* unusable, so this tip won't go into much more detail.
This might change in the future.

When this change makes its way through DNS, anyone will be able to look up your domain name, find this new record, and do something with this wallet address.

Understand the Risks

At this point in the book, you should be a little bit wary of the words *anyone* and *something*, because publishing an address like this means anyone can do *anything* with both the address and the knowledge the address is associated with the domain name.

DNS entries often contain contact information for the person managing the domain. While many registrars allow domain security and privacy to hide your personal information, some don't. Before you add this record, check your DNS settings with a DNS lookup tool like the `whois` command-line tool or a reputable website.

You *can* of course add this record even if you have contact information recorded, but be mindful of the fact that any transaction to and from this address can be associated with the domain contact permanently.

You should also be aware that DNS as originally designed has potential security flaws, including spoofing. If you're concerned about this potential flaw, make sure you've configured DNSSEC correctly.

Finally, if you fail to renew your domain or someone snatches it up out from under you, they can swap their address for yours in their configuration, in the same way that they could swap their website for yours if you lose control of your domain name.

---

[2] https://github.com/bitcoin/bips/blob/master/bip-0021.mediawiki

What Can You Do With This?

Apart from being an interesting forehead-slapper of a trick[3], the value of this trick is in the integrations it enables.

Imagine a web browser extension that could show you a Dogecoin tip address whenever you visited, if the site owner had one configured. If you like the site, throw a few Doge their way, right from your browser or device.

Or imagine we repurposed this record to point to a text file or structured data file on the server, served over HTTPS, that gave access information, purchase data, or other suggestions, like "If you send 10 Doge to this address, you'll get 24 hours of access" for a news site or something.

Consider also an alternate approach. `ifdogethenwow.com` has, as of this writing, the main domain and a well-known[4] subdomain of `blog.ifdogethenwow.com`. Any and every subdomain could have its own TXT record with a unique wallet address.

An enterprising shibe could start a site called `shibetips.whatever` and, for a small fee or out of the goodness of their hearts, allow users *optionally* to associate an address with a subdomain. As always, keep in mind the security and privacy implications, but think of the possibilities.

---

[3] In the sense that, "Oh yeah, you could totally do this!"

[4] Okay okay, hear me out though!

CHAPTER 3

Getting Data from a Local Node

When you run your own node, you can do more than just mine blocks and keep the network healthy. You get insight into the current, past, and future state of the network.

This is powerful. You do not have to rely on someone else to tell you what's happening, what might happen, or what has happened. You can get that from your node yourself.

Here's how.

Tip #3 Authenticate RPC Securely

A running Dogecoin node holds a lot of information. If you ask it nicely, it can give you this information: what's in your wallet, what happened in a given block, the current difficulty, the contents of the mempool, and more.

If you read “your wallet” and think “Wait, I don't want just anyone to have access to my wallet”, that's good. You probably don't. That means you need to restrict access to your node and especially the way you connect to it to send it commands.

Fortunately, the Dogecoin Core has a way to ensure that only the people who are supposed to connect to a node can do so. All you have to do is configure it securely.

Understanding Secure Authentication

To authenticate to a Core node to send RPC commands, you need to prove your identity in two ways:

- The node recognizes the user you claim to be
- The node confirms that you are that user

As you can imagine, the straightforward approach is to give the node a list of authorized users and some way of confirming the identity of those users. In other words, if the server knows about `ralph`, `nelson`, and `milhouse` and someone tries to connect as `jimbo`, the node can easily deny that connection.

Of course, if Jimbo gets smart and tries to connect as `milhouse`, the node also needs to check `milhouse`'s password. If Jimbo tries `bartrules`, `imissmymom`, and `pantsme`, he'll have trouble. If he manages to guess `thrillho`[1], he's in.

You can see the benefits and flaws of this approach, especially because Milhouse wrote `thrillho` on his hand so he wouldn't forget it.

How does the Core know `ralph`, `nelson`, and `milhouse` are all valid users and to reject invalid passwords?

Add Users to Your Config File

The answer is in your *dogecoin.conf* file and a program from the Dogecoin Core repository at *share/rpcuser/rpcuser.py*[2].

---

[1] See https://www.youtube.com/watch?v=nbbKsAZatao.

[2] See https://github.com/dogecoin/dogecoin/blob/master/share/rpcuser/rpcuser.py, for example.

You'll need the Python programming language installed to run this program. Run it like this:

```
$ python3 share/rpcuser/rpcuser.py lisa

String to be appended to bitcoin.conf:
rpcauth=lisa:b0a414f0d217c5bdec8db24ff340223$...
Your password:
...=
```

I've removed the relevant password contents, but trust me–they're there.

You'll need to keep the two pieces of output it provides. First, grab the line starting with `rpcauth` and add it to your *dogecoin.conf* file, then save the file. Start or restart your node.

Second, take the password and store it somewhere else, securely.

Don't Like the Password?

You could, of course, modify the code to provide your own password, but the random value you get here is going to be difficult to guess. Bitcoin has an updated version of this code (at least newer than the version in Dogecoin Core 1.14.6), so if you look in their repository, you might have more options when generating passwords.

When your node starts, test your authentication by connecting with your preferred RPC mechanism. I like to use the `curl` binary, because it has nothing to do with Dogecoin or any other cryptocurrency, so if I can get it to work, I know I have everything configured correctly.

```
$ curl --data-binary '{
    "jsonrpc":"1.0",
    "id":"curltext",
    "method":"getdifficulty",
    "params":[]}' \
    http://lisa:...@127.0.0.1:22555
{"result":8101199.900972591,
 "error":null,
 "id":"curltext"}
```

If you get the authentication wrong, you'll instead receive error output.

How Does This Work?

If you run the Python code again, you'll get a different result, both the password you use and the string you need to add to your configuration file. The code picks some random values for you, with the password it provides and the salted value.

Why does this work?

Neither the configuration file nor the Core store or know your password. The information you add with the `rpcauth` line is two things: a random salt and the hashed value of the combination of the salt and your password.

To prove that you're `lisa` or `milhouse` or `nelson` or anyone else the node knows about, you have to provide your username and your password. When the node receives both values, it finds the relevant auth line for your username, splits the rest of the line into the salt and hash, then hashes your password with the salt and checks the results against the value you added to your configuration file.

Then (importantly) it throws away the password because it knows who you are.

Salt is Healthy in Moderation

You may hear about an alternate authentication approach where you add username/password pairs to your *dogecoin.conf* file directly. Beware of this; it's much less secure than the approach described here. If someone were to read the contents of your configuration file, they would be able to read your passwords.
With this salted approach, you still want to keep your configuration file safe, but an attacker will have much more difficulty figuring out your password.

If your password were directly available in memory or on disk, it'd be available to attackers. If the server always used the same salt, then an attacker could try a bunch of potential passwords with the same salt to find something that lets them in. By using a different random salt for every user, attackers have to do a lot more work.

Of course, changing your password (and salt) every now and then is probably a good idea.

Understand the Risks

You can do a lot of interesting things with RPC commands and a node that has a wallet connected, but anyone who can authenticate and send RPC commands to

your node can potentially do things with and to your wallet. Unless you *really* need a full wallet connected to your node, you're safer running in `-disablewallet` mode.

If you *do* need a wallet, use layers of security such as a good firewall, binding your node only to trusted network interfaces, et cetera.

Please also note that there's no reason you have to run the Python code on the same computer as where you're running your node. In fact, it's a lot safer if you don't. That way your password and the salted, hashed password aren't on the node at all, so they're not both available to attackers.

If you need to store your password somewhere, for example with an automated process, be sure to store it securely in a way that it's not also exposed to attackers. This is the weakest part of the entire security model, so if you're going to keep it in plain text in a file on your server in the cloud somewhere, realize what you've exposed yourself to.

Tip #4 Take Actions on New Blocks

Because Dogecoin is an ever-increasing network of transactions, you can think of interesting ideas if you realize that a network of transactions is a series of events.

Payment processing is an easy and obvious idea. When a block gets mined, look for any payments to any address you're interested in, then do something based on the source address, the destination, the amount, any script in the transaction, or whatever.

You could also monitor the health of the network as a whole by looking at time between blocks, number of transactions in a block, number of coins transferred in a block, difficulty change in blocks over time, or any other piece of data available.

Dogecoin Core gives you options to treat these events as events, so you can do the interesting things you have in mind.

Configuring `blocknotify`

Dogecoin inherited a Bitcoin feature called `blocknotify`. This is a configuration option available from the command line or set in *dogecoin.conf* which allows you to ask the Core to launch an external command whenever the Core processes a new block.

You can pass two options to this command: the *number* or *height* of the block and the *hash* of the block.

With that, you have everything you need to do much, much more.

How do you make this work?

First, configure your node for RPC. Be aware of the risks of doing so, and follow all the security guidelines to your degree of comfort and safety.

Second, write a command that does something useful.

Third, launch or re-launch your node.

That's it.

Processing a New Block

Let's show this off by writing a simple command that shows basic block statistics, such as the number of transactions, difficulty, time, and size of a block. As it turns out, I already have some code in Perl that performs authenticated RPC, so I'll reuse that and gloss over some of the details.

```
#!/usr/bin/env perl
```

```
use v5.036;

use JSON;
use Path::Tiny;
use RPCAgent;

exit main( @ARGV );

sub main( $height, $hash ) {
    my $config = get_config( $ENV{CONFIG_FILE} );
    my $rpc    = create_rpc( $config );
    my $block  = $rpc->get_block_by_hash( $hash );

    my $stats  = analyze_block( $block );

    say <<~END_HERE;
    Found block $height
    Processed at $stats->{time}
    Contains $stats->{num_tx} transactions
    Size of $stats->{size}
    Difficulty of $stats->{difficulty}
    END_HERE

    return 0;
}

sub analyze_block( $block ) {
    my $num_tx = $block->{tx}->@*;
    my $time   = localtime $block->{time};

    return {
        num_tx     => $num_tx,
        time       => $time,
        difficulty => $block->{difficulty},
        size       => $block->{size},
    };
}

sub get_config( $config_file ) { ... }
sub create_rpc( $config )      { ... }
```

If you haven't read much Perl, don't fret. It's relatively easy to follow despite some of the syntactic details looking a little odd. The first handful of lines set the version of Perl (the latest stable release, as of this writing, so I can use a few newer features) and load a couple of useful modules, including `RPCAgent`, the code I alluded to earlier. All *that* code does is wrap calls to the Core's RPC listener in a Perlish interface, so I can write `$rpc->get_block_by_hash( $hash )` and not have to think about the details of providing the right HTTP headers. Everything's already set up.

The interesting work is in `main()` and `analyze_block()`.

`main()` starts the action. I've set up a configuration file you'll read more about in a moment. That configuration gives me enough data to set up an object that performs the RPC requests.

When this code runs, it gets two arguments, the height of the new block and the hash of the new block. The RPC call gets all of the block's data with that hash. The Core returns that data as a JSON data structure, but the `RPCAgent` code turns that into a Perl data structure (a nested hash, or a dictionary as you might call it in Python, or a hashmap in Java, or an object in JavaScript, or…).

From there, `analyze_block()` takes that data structure and pulls out a few pieces of data. Difficulty and size are obvious fields. Obvious transformation is obvious there. The block's time is in a field named `time`, but that records seconds since the epoch, so I use Perl's `localtime` to turn that into a textual representation. Finally, the *number* of transactions is interesting, so I use a standard Perl idiom to look at the array of items in the `tx` field (representing transactions) and get a scalar value, representing the count of items in the array.

But I Don't Read Perl!

"Context? Scalar? That's unique!" *De gustibus non est disputandum*, but I wrote a book called *Modern Perl* that explains all this stuff and more, so head over to http://modernperlbooks.com/ and download a free copy. You'll have one more interesting tool in your toolkit!

When `main()` gets the results, it then prints a description of that transaction. It will look something like this:

```
Found block 4485057
Processed at Wed Nov 23 15:15:28 2022
```

```
    Contains 12 transactions
    Size of 3979
    Difficulty of 12696685.226509
```

I've elided the configuration file setup and the `RPCAgent` object creation. If you have much programming experience, you can probably imagine what they look like anyhow.

Launching Your Program

How does this get executed? My personal preference is to modify my *dogecoin.conf* file to add this command. I wrote a little shell wrapper to set up the environment correctly[3]:

```
  #!/bin/bash

  cd "${HOME}/dogecoin-tricks-book/"
  export CONFIG_FILE="./chapter_3/env.json"

  perl -Ilib bin/show_block_stats.pl $*
```

This file sets the path to my RPC credentials configuration file and makes sure Perl knows which paths to search to load my `RPCAgent` module.

All that's left is to add a single line to *dogecoin.conf*[4]:

```
blocknotify=/bin/bash \
  "${BOOK_HOME}/chapter_3/bin/launch_listener.sh" \
  "%i" "%s" >> blocks.log
```

Again, this looks a lot like what you'd use to set up a cron job on a Unix system.

The `%i` parameter is the height of the new block and the `%s` parameter is the hash of the new block. Whenever the Core processes a new block, it'll launch a system process and pass those two values as variables. In this case, the system will invoke the `bash` shell which will itself invoke `perl`, and away things go.

If you make this change, you'll have to restart your node before the Core will start executing this command.

---

[3] As you might do for a `cron` command.

[4] I've added linebreaking for the book's formatting.

Understand the Risks

As with any use of RPC against a local Core, you should enable authentication and authorization (see Authenticate RPC Securely, pp. 10) and keep unauthorized activity away from your node. This is especially true if you have a wallet attached to your node (so try to avoid doing that). Anyone who can connect to your node via RPC can get a lot of data you might not want to expose and take network actions you might not want to permit.

Second, be aware that you could write buggy code[5]. I wrote the example code in such a way that I could test my program in isolation by providing block heights and hashes I already had without having to test the integration in my *dogecoin.conf* file. This let me fix bugs without launching and relaunching the Core.

Okay, I *didn't* do that the first couple of times, but I wish I had, so I recommend you learn from my mistake and do it right the first time.

If you do get your program into a crash loop, you might harm the stability of the system and the experience for you and others on the machine. Don't let that stop you from doing interesting things but do be careful to think about what could go wrong.

What Can You Do With This?

I already suggested a couple of interesting ideas, but the world is open at this point.

Perhaps you can light up a lava lamp whenever you make or receive a transaction.

You could update a counter on a webpage or post a message to Slack or Discord with every new block mined.

You could plot the size and fullness of blocks over time.

You could keep a list of the busiest wallets over the past day, week, or month.

I'm sure you'll come up with a few more clever ideas that aren't on this list either.

---

[5] I sure did, the first couple of times I tried this.

Tip #5 Add an Action Launcher

When you start developing things that require changing your *dogecoin.conf* file, you'll inevitably run into situations where you need to restart your Core node to test your changes.

The wait for a node to restart can be annoying on a fast local machine, where you have to wait several seconds to validate all the blocks you have downloaded. On a remote machine in the cloud with a full node, you could wait *minutes* or more between restarts.

We can do better.

One approach is to add features to the Core that let you configure its parameters through RPC calls. Another approach is what software developers happily refer to as "yet another layer of indirection".

Fortunately, there's one really good opportunity to make your life a lot easier.

Launch a Launcher for New Block Actions

The tip for Take Actions on New Blocks, pp. 14 showed off a Dogecoin feature to launch an arbitrary program for every new block accepted on the network. Whether that's one program or a thousand you want to launch, it's all the same to the Core; it's just a single configuration like in the *dogecoin.conf* file.

The contents of this file are completely arbitrary. You could do something, but you could also do entirely nothing. You could do a dozen things. All the things you can do could change.

There's only one you *can't* change without restarting your Core node: the command to invoke.

The previous tip had a configuration like this:

```
blocknotify=/bin/bash \
  "${BOOK_HOME}/chapter_3/bin/launch_listener.sh" \
  "%i" "%s" >> blocks.log
```

. . . but if it changed to something like this:

```
blocknotify=/bin/bash \
  "${BOOK_HOME}/chapter_3/bin/launch_loader.sh "%i" "%s"
```

. . . then the file `launch_loader.sh` might be:

```
#!/bin/bash

HEIGHT=$1
HASH=$2

# launch the first block listener
cd "${HOME}/dogecoin-tricks-book/"
export CONFIG_FILE="./chapter_3/env.json"

perl -Ilib bin/show_block_stats.pl $HEIGHT $HASH \
    >> show_block_stats.log

cd

# launch the second block listener
bash bin/show-desktop-notifications $HEIGHT \
    >> show_desktop_notifications.log

# launch the third block listener
ruby bin/some_cool_code.rb $HASH \
    >> some_cool_code.log

# post something to a webhook
curl -X POST ...
```

In plain English, all this file has to do is prepare the arguments the Dogecoin Core provides (height of the newly-mined block and its hash), do whatever environment manipulation is necessary, and launch a bunch of commands.

You can write this launcher in whatever language you like: shell, Python, PowerShell, Rust, whatever you like. If you want to launch all of these processes independently in parallel, you can. If you want to launch them one after another, you can.

Best of all, you can make changes to the loader without restarting your node. You can even test this by providing your own arguments to it; it's just a program, launched by the Core automatically but a program runnable however you want.

Understand the Risks

There's only one risk of this technique not already addressed in other examples: anyone who can edit this file can make your Core node launch programs.

If you put authentication information for Core RPC calls in this launcher, they're available to other programs. If you have authentication information for other systems

or services, such as a database or a cloud machine or a web service, they may be available to other programs launched from this launcher too.

It's not easy to predict every way someone might find to interfere with your system, but keep this file locked down as tightly as you would your core configuration yourself.

Tip #6 Add Desktop Notifications

Now that you know how to do things when the Dogecoin network accepts new blocks (Take Actions on New Blocks, pp. 14), it's time to get creative. You could go the Internet of Things route and turn on a lava lamp, play a trumpet salute through the office speakers, or flash a message on the electronic reader board outside your school[6].

Let's start with something simple. Let's pop up a desktop notification on Linux, Windows, or Mac OS X with interesting data you can get for each block.

A Simple Desktop Notification for Block Difficulty

Like other Bitcoin-related cryptocurrencies, Dogecoin adjusts the difficulty of mining a block based on the average time it takes to mine a block. If more miners have more power on the network, the difficulty will increase. If fewer miners are running (they have solar power and it's cloudy), the difficulty will decrease. This variable difficulty is an attempt to keep the amount of time it takes to mine a block to about a minute no matter how much or little mining capacity is working.

If you're a miner, this is interesting because mining difficulty and hashrate affects the likelihood that you will mine any single block.

If you're not a miner, it's an interesting number to track to gauge what's going on in the network. It's also easy but not trivial to get from a Core node, which makes it a good example.

Let's think about what we want to happen. In simple terms:

- **when** a new block is mined
- **get** the difficulty of that block
- **display** a message with interesting information

We already know the first part. It's time for some code.

Display Latest Block Difficulty Notifications

What does this code need to do? The Core will launch it every time a new block gets mined. It needs to receive a single argument: the height of that block.

[6] Get permission before doing any of these things, of course!

To get the difficulty of the new block, we can use the RPC call `getdifficulty`. This call takes no arguments and returns the current difficulty.

That's also easy enough; we know how to make RPC calls and get the results. All that's left is popping up a desktop notification.

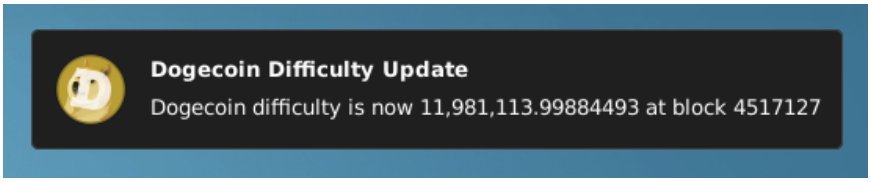

Figure 3.1: Dogecoin Difficulty Desktop Notification

On Linux, the `notify-send` command program (from `libnotify`) handles all of the details. You can provide it a title for the notification, text for the notification, and even an icon. The result looks like Dogecoin Difficulty Desktop Notification, pp. 23.

That's the concept, at least. What does the code look like?

```
#!/bin/bash

HEIGHT=$1
DIFFICULTY=$(dogecoin-cli getdifficulty)
DIFFICULTY_FORMATTED=$(
    numfmt --grouping ${DIFFICULTY_RAW}
)

notify-send \
     -i "$HOME/.icons/dogecoin.png" \
     "Dogecoin Difficulty Update" \
     "Dogecoin difficulty is now "
     "${DIFFICULTY_FORMATTED} at block ${HEIGHT}"
```

That's not so bad, but there's a little bit going on behind the scenes.

The `HEIGHT` assignment gives a readable name to the one parameter this script gets, obviously the height of the freshly-mined block.

The `DIFFICULTY_RAW` assignment uses the `dogecoin-cli` utility to make the `getdifficulty` RPC call. Using the CLI program directly does two things: avoid

having to deal with authentication for a call on the local machine and get the result directly as a plain number, rather than JSON output.

The `DIFFICULTY_FORMATTED` assignment uses the GNU `numfmt` utility to turn a big number into something formatted more appropriately for display. In my current locale, that means adding commas to separate thousands. Big numbers are easier to read this way.

But I Don't Read Bash!

You can write this in whatever programming language you prefer, even Perl[a].

[a] Admit it, you thought it might be in Perl, didn't you?

Finally, the call to `notify-send` takes three arguments. First, the `-i` argument uses a path to an icon. In this example, I took the Dogecoin logo from the core (*src/qt/res/icons/dogecoin.png*) and put it in a directory of my own making. The second argument is the title of the notification. The third interpolates the nicely-formatted block difficulty and the block height into the notification body.

That's it.

This message will pop up on my screen every minute or so and gradually fade away after a couple of seconds.

Linux… on the Desktop? Not for me yet!

Never fear; you can modify this for Mac or Windows too. Windows has a port of `notify-send` currently available from https://github.com/vaskovsky/notify-send. Check the source code and provenance before you install, of course.
Mac OS X users can use the `display notification` feature[a]. You'll have to run the command through `osacript -e "..."`.
If you don't have `numfmt` installed, install the `coreutils` package on Linux or via Homebrew on Mac OS X. This may require a different approach on Windows.

[a] See https://developer.apple.com/library/archive/documentation/AppleScript/Conceptual/AppleScriptLangGuide/reference/ASLR_cmds.html#//apple_ref/doc/uid/TP40000983-CH216-SW224 for more details.

Understand the Risks

What can go wrong here?

You don't have to have a full node running locally, or even a node with a wallet. You can have a slim node or a headless node or a wallet-free node, but the code as written requires a node running locally.

To connect to a node on another machine securely, be sure to set up authentication appropriately (Authenticate RPC Securely, pp. 10) and provide the appropriate connection arguments to the `dogecoin-cli` call.

If you install third-party code (such as the `notify-send` port on Windows or `coreutils` on Mac OS X), beware that the code you install is what you intend to install, that you get it from a reputable place, and that anything you download has risks of exposing your information.

If you use this code to affect a third-party machine or system (such as playing the chorus of Queen's “Another One Bites the Dust” on every block mined), be aware that your coworkers, roommates, fellow commuters, and everyone around you may not fully appreciate your musical taste[7] and you should be kind to them because they don't share your excitement about block mining.

[7] Sure, it's not “Radio Gaga”, but it's also not “Crazy Little Thing Called Doge”!

What Can You Do With This?

Monitoring difficulty may not be super interesting, but reifying actions on the chain into events untethered to the network can do a lot of things.

You could update stats to display on your web page.

You could estimate time between payments settling if you take Doge for in-person transactions or e-commerce transactions.

You could bring additional mining capacity online if you see the difficulty rate or hashrate drop.

Or you can leave this as it is and bask in the warm glow that you're part of a community that processes transactions in big blocks, day and night, every sixty seconds or so.

Tip #7 Enhance RPC Calls

Because Dogecoin has Bitcoin, Litecoin, and LuckyCoin in its pedigree–and in its code–you can often find guides and documentation written for those coins that work for Dogecoin.

Sometimes you can't, however.

For example, the current major version of Bitcoin (version 24.x) supports several RPC calls that use and manipulate and query address labels.

The current major version of Dogecoin (version 1.14.x) doesn't support these features.

That doesn't mean you're out of luck. That means you have to be a little creative.

Proxy RPC Calls

Think of a label as a nickname. Any operation you want to do on an address, you can do on a label instead. You just need the ability to associate a label with an address and you're off to the races.

What do you need to make this work? Well you *could* install a toolkit named `Finance::Dogecoin::Utils` from CPAN[8] or you could do the work yourself.

What do you need?

Associate Labels with Addresses

First, you need a way to associate a label with an address. Bitcoin has an RPC call named `setlabel` which does this in the current wallet. It takes two arguments, the `address` to associate and the `label` to use.

This already suggests a lot of the implementation.

In `Finance::Dogecoin::Utils`, I use this information to populate a JSON file stored in the appropriate configuration directory (*~/.local/share/dogeutils/* on a Linux system). This JSON file contains an object that maps a label to an address. For example:

```
{
  "Dogecoin Book Tips":
      "DAY5wNkebzEyqUXCkN9koKNBuzXRKRTjcL"
}
```

[8] Use the command `cpanm Finance::Dogecoin::Utils` if you have Perl installed. Otherwise....

Writing this as a plain text file has several benefits: you can move it between machines, you can edit by hand if you want, and it's not tied to any wallet format. You can read or write it with anything that understands JSON.

The drawback is that it's not attached to your wallet itself, although that may be a benefit. If someone gets their hands on this file, they'll know you have interest in these addresses (and know what you called them), but they won't get your wallet itself.

With the `dogeutils` command provided by the Perl code mentioned earlier, you could achieve by using the command line:

Proxy RPC Calls to a Node

The next thing you need is a way to make calls to a running Core node. For this, be sure to set up authentication (see Authenticate RPC Securely, pp. 10). With a username and password, you can use any HTTP client or library such as `curl` or Python's `requests` or whatever you prefer in your preferred language to make requests of the node and relay back the responses.

Handling passwords securely can be tricky. You *could* use an authcookie solution. I decided to use a username and password approach[9] where another JSON file in the same configuration directory associates names and passwords.

When the `dogeutils` program starts, it looks for a username provided on the command line (or in an environment variable) and then looks up the password for that username in the authentication file. If found, any call it makes to the actual Core node will send that username and password.

With authentication set up, all you need to do is know how to call a method via HTTP.

In the example of Bitcoin's `setlabel` call, you need to:

- Connect to the node's IP address on the right port
- Pass your username and password with HTTP basic authentication
- Make a `POST` request
- Provide a JSON body with the appropriate format

If you have a Core running on your local machine, connect to `localhost:22555`. Make sure the URL also includes your URL and password, so that it looks like

[9] At least until someone suggests something better!

`http://username:password@localhost:22555/`. Set a `Content-Type` header of `application/json` and pass a message body of the encoded JSON:

```
{
  "jsonrpc": "1.0",
  "id": "some identifier here",
  "method": "setlabel",
  "params": [
    "DAY5wNkebzEyqUXCkN9koKNBuzXRKRTjcL",
    "Dogecoin Book Tips"
  ]
}
```

Note how close that looks to the `dogeutils` command earlier.

Wrap Calls That Don't Exist

What good is this?

Besides `setlabel`, Bitcoin provides an RPC call called `getreceivedbylabel`, which returns the amount of coins received by the address(es) associated with a label. This does the same thing as `getreceivedbyaddress` except its argument is a label, not an address.

That's fine; now we've gone through all the pieces necessary to make this work.

`dogeutils` has a little bit of glue code to understand it should provide both `setlabel` (which doesn't talk to a Core node) and `getreceivedbylabel` (which does). The *implementation* of the thing that looks like a `getreceivedbylabel` call is entirely within the `dogeutils` program. Given a label, it looks up the associated address in the JSON file, then calls `getreceivedbyaddress` with the address and returns the resulting JSON verbatim.

That's it.

As long as you have some way of communicating with a Core node, some way of managing user authentication, and some type of local storage (hard-coded data, a JSON file, an SQLite database, whatever), you have lots of options.

Understand the Risks

RPC against a Dogecoin Core node needs authentication. Enable this. Manage it securely.

The authentication approach I describe here has some flaws; leaving your password in the clear even in a file you control means anyone who can get access to that file has your password. It'd be more secure to store it in a local keychain somewhere.

Alternately, you could force the secure entry of your password on the command line every time you run a command like `dogeutils`.

This is probably secure enough if you're running this program against a Core node running on the same machine, because that's just as vulnerable to anyone who can modify your *dogecoin.conf* file and add password entries or read the authentication cookie. If you're running your utility over the network (or connecting to a Core over a network), ensure that the network is secure.

What Can You Do With This?

You can use this to provide RPC calls not yet supported by the current Dogecoin Core, of course, but you can provide other features as well.

Want to get all of the transactions at a specific block height? Chain together several RPC commands and make your own utility call.

Want to list blocks in reverse order? You can do that.

Want to turn addresses into labels in the output? You can do that too.

You're not limited to *modifying* calls either. You could provide an access control mechanism where certain user accounts can access specific types of calls. For example, you might have read-only accounts that can read general blockchain data such as looking at blocks or transactions and provide other accounts that can create transactions or work with wallets.

CHAPTER 4

Inside the Dogecoin Core

Dogecoin is a lot of things. One of those things is the Dogecoin Core wallet. While anyone can write or run their own client, this code is the first client among equals.

This is good, in that it's tested, observed, analyzed, and deployed widely. It's risky, in that Dogecoin belongs to everyone who participates. Fortunately, the code is open and available for anyone to read, study, adapt, modify on their own, and suggest changes.

That includes you.

Tip #8 Follow Core Development

How do you manage a global network where anyone can participate and the success or failure of your personal financial transactions relies on your ability to trust the work of countless strangers?

You have to trust that everyone's playing by the same rules. To do that, you have to verify that you understand the rules and that someone's not trying to sneak something past you.

In the world of Dogecoin and cryptocurrency, this means that we must:

- follow rigorous, tested, and well-understood mathematical principles and formulas
- adhere to well-defined rules and parameters for participation
- trust the software we use to implement all of these things correctly

That's one of the reasons the Dogecoin Core is so important. The source code is available, the released built from it are done so transparently, and development and organization occurs in the open.

Where is the Core?

At the time of this writing, the source code is available on a site called GitHub, at https://github.com/dogecoin/dogecoin[1].

This website and the `dogecoin/dogecoin` project in particular allow people to collaborate in the development process, whether submitting bugs, asking for new features, improving documentation, translating text into multiple languages, adding new features, or having design discussions.

This is a good place to report a potential bug or ask for a new feature.

---

[1] Though it's unlikely this will change any time soon, if you're reading this in 2099 or even 2029, check to see if this is still true before blindly downloading anything.

Productive Discussions and Contributions

While anyone *can* participate, please be respectful of the time and effort of countless other people. Creating an issue saying "Devs please make price go up!!!" won't help anyone, because Core development deliberately resists any activity that could influence the price.

Navigating the Core

If you look at that page on its own, you may find it a little overwhelming at first (so many features!) and a little underwhelming (wait, where's all the activity?). To navigate all of this, you need to understand a little bit about Git and GitHub.

Git is a software tool that lets developers manage the source code: the instructions on what the Dogecoin Core does and how it does it. All of this source code is stored in a Git *repository*.

GitHub is a website that lets multiple developers share their Git repositories with each other. While Git itself is a distributed system that allows people to collaborate, it doesn't inherently enforce any "official" prime repository. GitHub does; only those people allowed to make changes in the `dogecoin/dogecoin` repository can change the code that makes up the software that you're running.

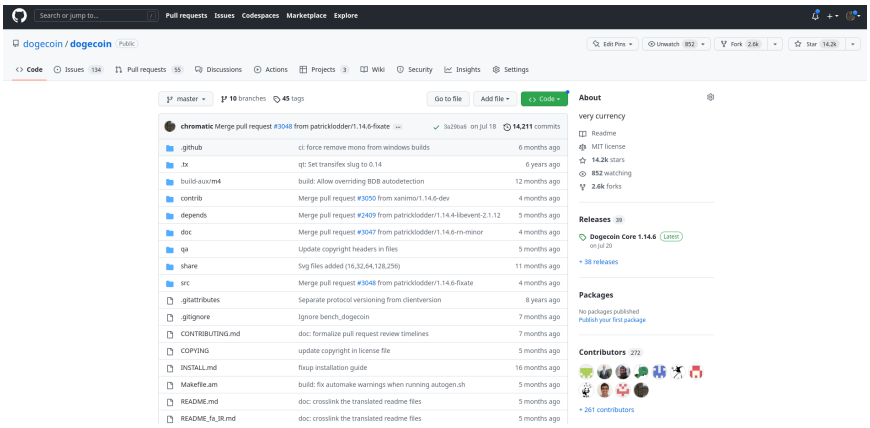

Figure 4.1: Dogecoin Core Github

However, anyone can *access* GitHub and create (or *fork*) their own repository

to make their own changes. Then, if they choose, they can request other people *pull* those changes into their own repositories (or even the main repository itself, `dogecoin/dogecoin`).

That forking metaphor is really important.

If you think of everyone's individual repository as a fork off of the main repository, you get the idea of a tree or a river or some sort of organic entity. This holds true in the main repository itself, but rather than forks it holds *branches*.

In the same way that a fork represents something that's different from the main repository, so branches can be different.

In the same way that a fork can produce a pull request (to merge changes back into the main repository), so can branches.

If you've never used Git or GitHub before, this isn't always obvious. However, now that you know, you know what to look for.

So where does development happen?

Dogecoin Development Process

If you only ever looked at the main branch (called `master`), you'd think not much ever happened. That's because all development occurs on other branches.

In Dogecoin Core Development Branch, pp. 34, you can see a little widget saying `1.14.7-dev`. This switches the web view to show a branch other than the `main` development branch. This particular version number names a branch that represents "the code that will be released as version 1.14.7".

You can switch that view yourself to see what *was* released in previous versions and what *will be* released in future versions. As of this writing, there are two future versions in active development, 1.14.7 and 1.21.

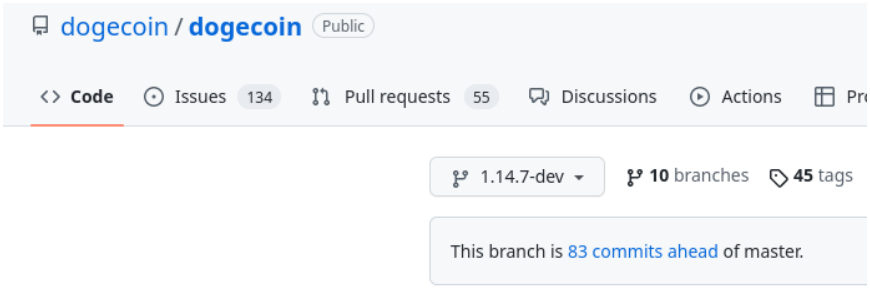

Figure 4.2: Dogecoin Core Development Branch

In this image, you can also see links for Issues and Pull Requests. The former is bugs or feature requests and the latter is code that's in progress. Pull requests always point to a specific branch, so that the developers can keep track of what they intend to merge where and when.

At any point, you can use the Code link, or the list of files and directories, to search through the code as it existed at any point in time (or, in the case of pull requests, code as it would exist if the pull request were merged to a branch).

This is really powerful stuff, even if you're not a developer.

What Can You Do With This?

There's a lot more you can do with GitHub and the code, even if you're not a developer. Feel free to click around, read issues, think about discussions, and look at pull requests. Contribution is open to anyone willing to put in the time and effort to help their fellow shibes.

As always, be respectful, understand the rules, and treat other people with respect. Remember that this tool is essential to the Core developers, so please be mindful of their time and resources.