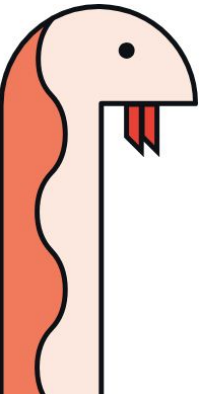
Build the fastest miner!

Rigel Di Scala

PyCon Italia ‘25



PYCON(IT) 25

Rigel Di Scala
Head of Development
@ Vedrai

VEDRAI

Choices I'd make again

(psst... we are hiring!)


AND NOW FOR
SOMETHING
COMPLETELY
DIFFERENT

Outline

1. Intro
2. Asyncio 101
3. IRC
4. Workshop
    a. build an irc client
    b. implement the mining algorithm
    c. build an irc miner
5. Final challenge: develop the fastest miner

Workshop objectives

1. Learn asynchronous programming

    How it works, what it's good for.

2. Learn the elegant algorithms of the Bitcoin protocol

    Mining, peer-to-peer networks, etc.

3. Have fun challenging each other

Bitcoin

r/AskReddit • 10 mo. ago
favioe1

If you could go back 10 years, what would you change?

Archiv

AdNegative9457 • 1y ago

Buy bitcoin.

572 2 Award Share

Buy Bitcoin when it launches, convince my parents to invest in Apple and Amazon

1 Share

Sort by: Best

Search Comr

whymartino1 • 10mo ago

I would buy Bitcoin

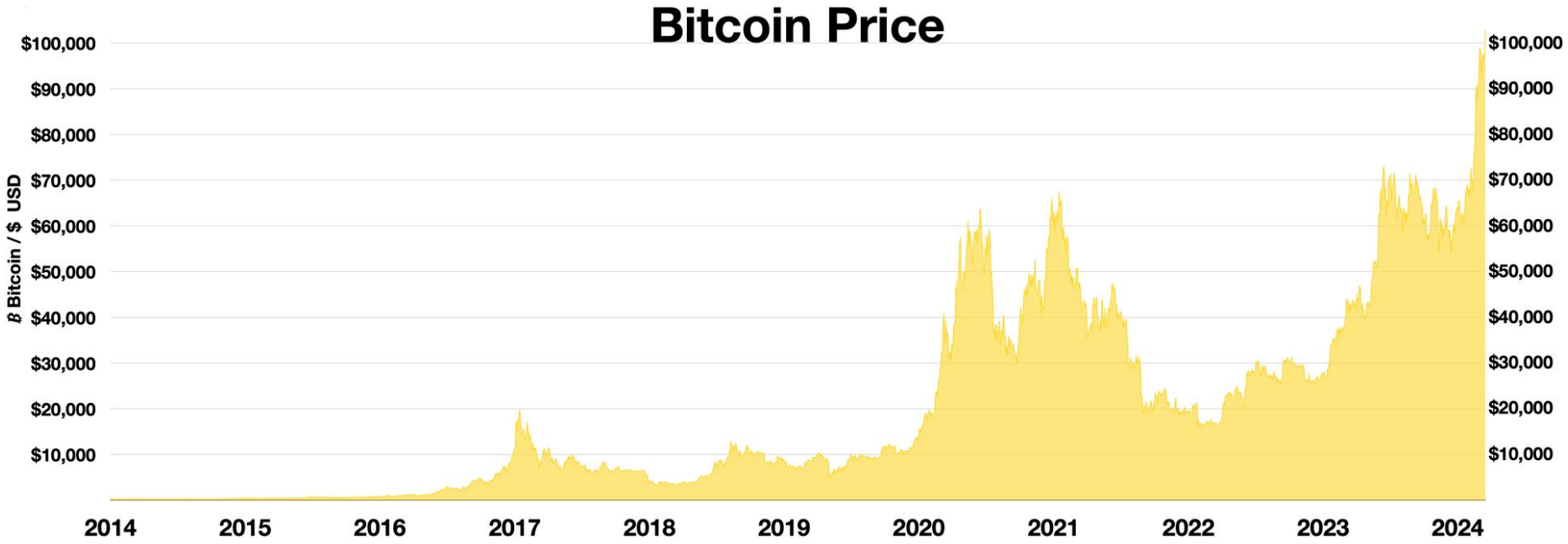
866 Award Share

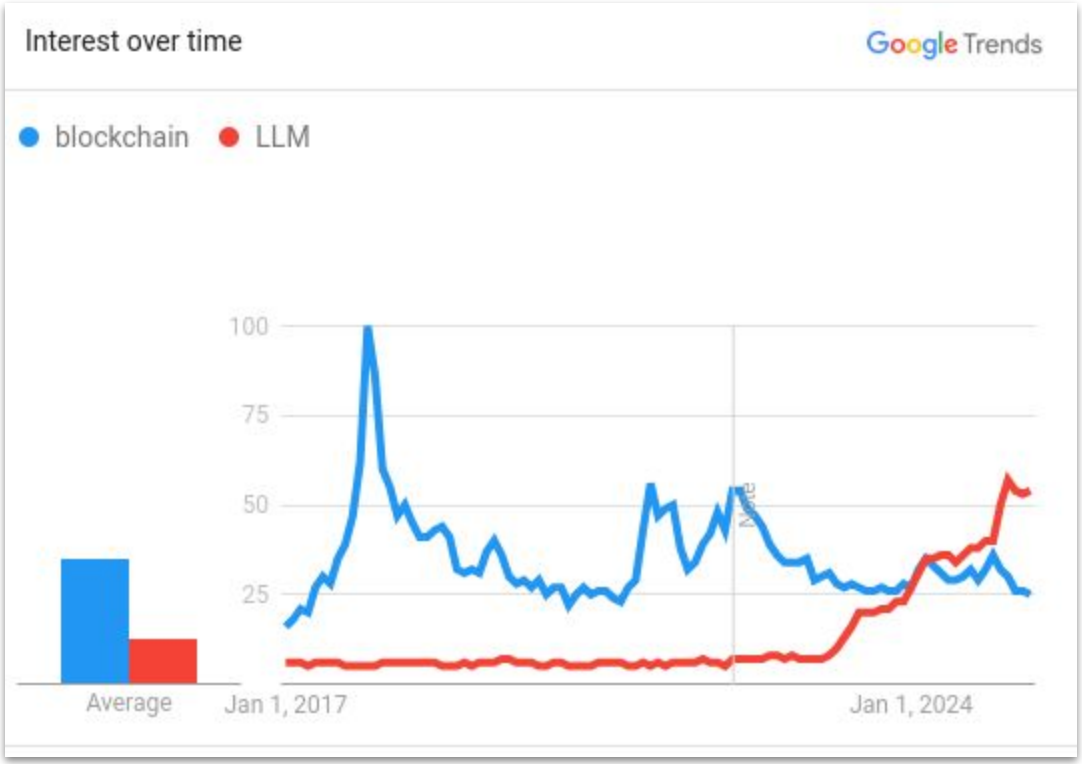
66 more replies

Dramatic_Mulberry142 • 3mo ago

C is the best language to learn CS, but please buy Bitcoin

12 Reply Award Share


Bitcoin Price
₿ Bitcoin / $ USD
$100,000
$90,000
$80,000
$70,000
$60,000
$50,000
$40,000
$30,000
$20,000
$10,000
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024


Interest over time
Google Trends
blockchain
LLM
100
75
50
25
Note
Average
Jan 1, 2017
Jan 1, 2024

Nvidia (2018)

WILLIAM SUBERG MAR 30, 2018

Nvidia CEO: Blockchain Will Stay ‘For Long Time’ Thanks To ‘Low Friction’ Cryptocurrency

Cryptocurrency “will be here for a long time” thanks to its ability to exchange value without “friction,” Nvidia’s CEO has said.

27212 Total views 495 Total shares

Also Nvidia (2023)

Cryptocurrencies add nothing useful to society, says chip-maker Nvidia

Tech chief says the development of chatbots is a more worthwhile use of processing power than crypto mining

The first version ChatGPT was trained on a supercomputer made up of about 10,000 Nvidia graphics cards. Photograph: Tyrone Siu/Reuters

The US chip-maker Nvidia has said cryptocurrencies do not "bring anything useful for society" despite the company's powerful processors selling in huge quantities to the sector.

Michael Kagan, its chief technology officer, said other uses of processing power such as the artificial intelligence chatbot ChatGPT were more worthwhile than mining crypto.

Notarchain (2019 – 2024?)

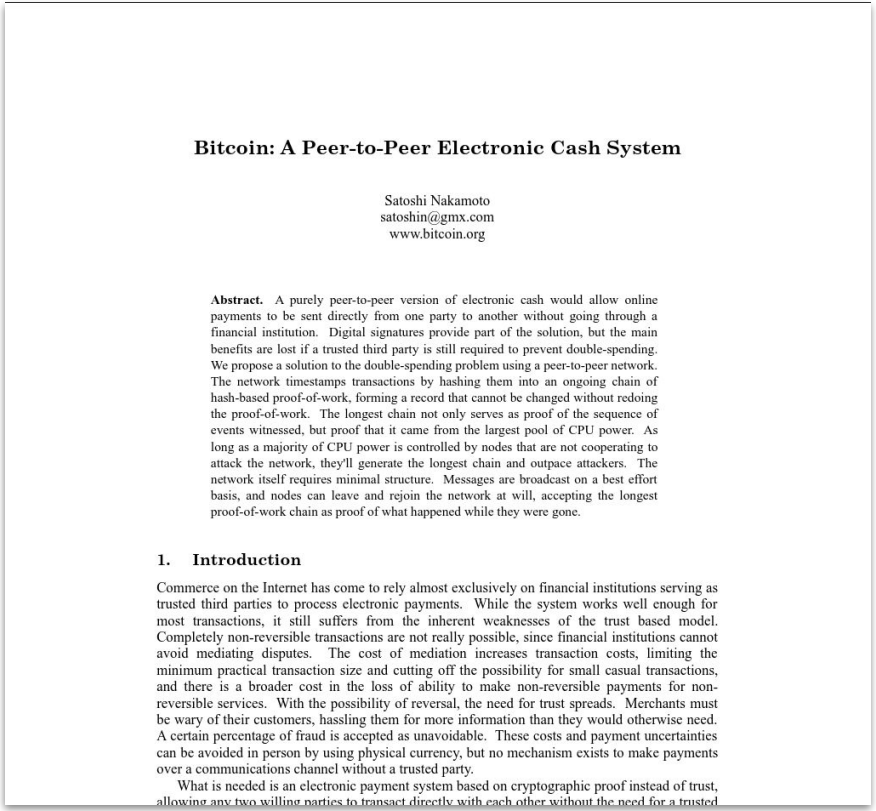
Notai di pregio internazionale, sia italiani che spagnoli, hanno di recente analizzato benefici e costi di un affidamento totale alla tecnologia in un'epoca in cui si parla quasi solo di blockchain, di smart contracts e di NFT (non fungible token). A prescindere dal fatto che la moneta di scambio per transazioni anche immobiliari sia ufficiale o meno, valuta reale o crypto, il nodo del dibattito risiede nella certezza che può essere attribuita alla blockchain e nella sicurezza che può essere attribuita ad un meccanismo informatico decentralizzato e disintermediato piuttosto che ad un sistema materiale basato su professionisti che conoscono le regole e che le applicano sotto diversi livelli di controllo, a tutela degli operatori economici.

Ministero della Giustizia
Amministrazione degli Archivi Notarili

Guida ai servizi
degli
Archivi Notarili

2024

Satoshi Nakamoto's Bitcoin whitepaper



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

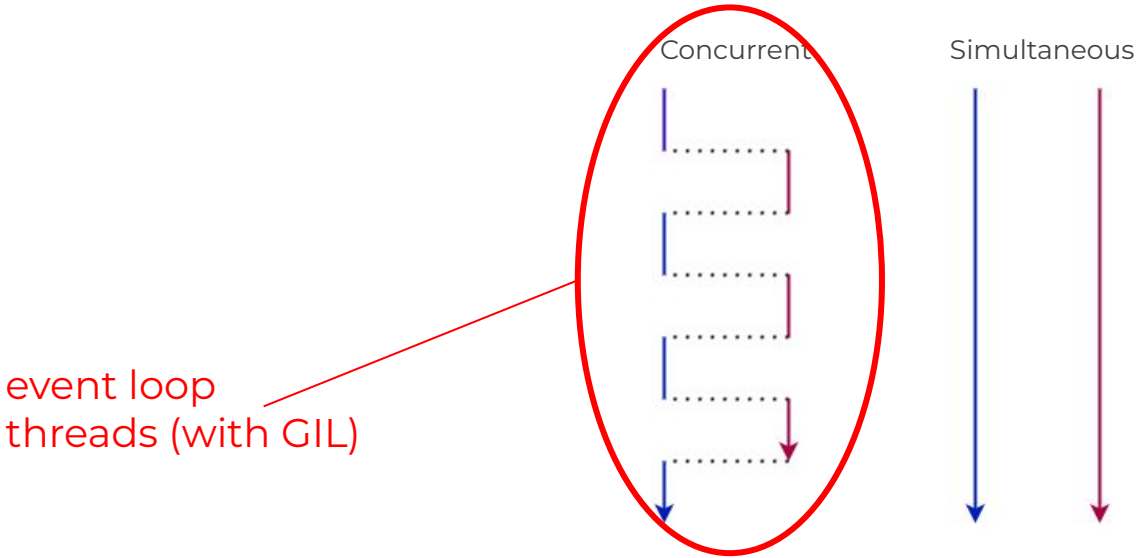
What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted

Asyncio 101

Let’s start with generators

Asynchronous programming

A programming paradigm that manages multiple **concurrent** paths of execution.



Asynchronous programming with asyncio

Based on the Event Loop.

Good for **I/O (network, fs, etc.)** intensive applications.

Bad for **computation** intensive applications.

Asynchronous programming with asyncio

Main concepts:

- Coroutine objects
- Async functions
- The Event Loop
- “Awaiting”
- Tasks

Coroutine

A computation that can be paused and resumed.

Vaguely related to Generators.

`await` and `yield` both pause the execution and give back control... *but to what?*

Generator vs. Coroutine

`yield`

gives back control to the

calling function

`await`

gives back control to the

event loop

The Event Loop

An internal Python interpreter facility that monitors and manages coroutines:

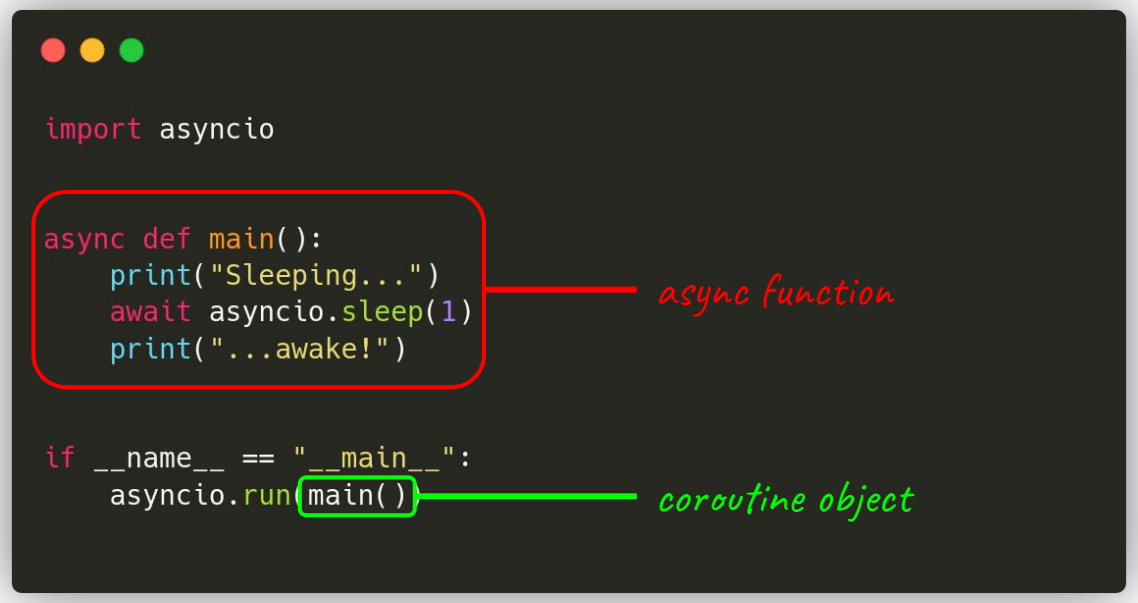
- executes in the main thread (by default)
- registers and monitors related file descriptors
- resumes coroutines when new I/O has occurred

A Task is an abstraction that represents a coroutine managed by the Event Loop.

Asynchronous function

A function that produces a Coroutine.

Creating and awaiting a coroutine



asyncio.run()

1. Is there a currently running event loop?
    a. If there is, raise an exception to prevent nested loops
    b. If there isn't, proceed
2. Create a new loop and set it as the current one
3. Run the passed coroutine until it returns
4. Cleanup by cancelling any running tasks
5. Close the event loop

Awaiting sequentially

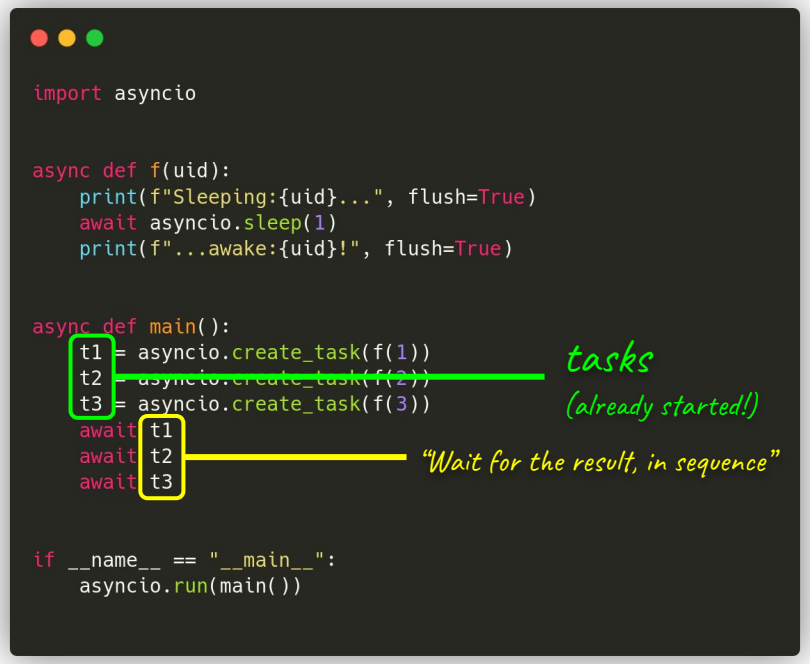
```python
import asyncio


async def f(uid):
    print(f"Sleeping:{uid}...", flush=True)
    await asyncio.sleep(1)
    print(f"...awake:{uid}!", flush=True)


async def main():
    await f(1)
    await f(2)
    await f(3)


if __name__ == "__main__":
    asyncio.run(main())
```

Awaiting concurrently*

```python
import asyncio


async def f(uid):
    print(f"Sleeping:{uid}...", flush=True)
    await asyncio.sleep(1)
    print(f"...awake:{uid}!")


async def main():
    await asyncio.gather(
        f(1),
        f(2),
        f(3)
    )


if __name__ == "__main__":
    asyncio.run(main())
```

*using tasks under the hood

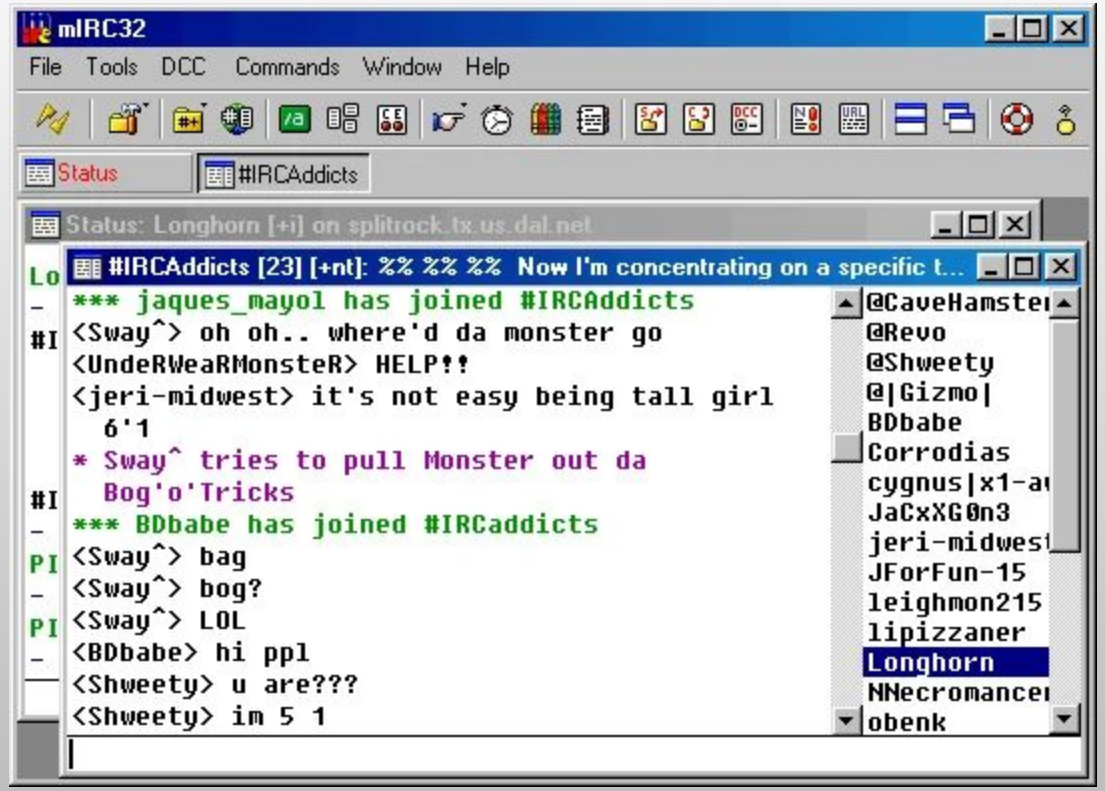
Creating and awaiting tasks



Internet Relay Channel

Khaled Mardam-Bey
(author of mIRC)


About mIRC
mIRC® v5.31 32bit
An Internet Relay Chat Client
Copyright © 1995-1998
Khaled Mardam-Bey & mIRC Co. Ltd.
All Rights Reserved.
Author!
Visit the mIRC website for the latest version, hints and tips, and general IRC information.
http://www.mirc.co.uk
Visit
How to register
mIRC is Shareware... if you like it, please register. Thanks :)
Licensed to:
Unregistere
License
Introduction


mIRC32
File Tools DCC Commands Window Help
Status
#IRCAddicts
Status: Longhorn [+i] on splitrock.tx.us.dal.net
#IRCAddicts [23] [+nt]: %% %% %% Now I'm concentrating on a specific t...
*** jaques_mayol has joined #IRCAddicts
<Sway^> oh oh.. where'd da monster go
<UndeRWeaRMonsteR> HELP!!
<jeri-midwest> it's not easy being tall girl
6'1
* Sway^ tries to pull Monster out da
Bog'o'Tricks
*** BDbabe has joined #IRCaddicts
<Sway^> bag
<Sway^> bog?
<Sway^> LOL
<BDbabe> hi ppl
<Shweety> u are???
<Shweety> im 5 1
@CaveHamster
@Revo
@Shweety
@|Gizmo|
BDbabe
Corrodias
cygnus|x1-a
JaCxXG0n3
jeri-midwest
JForFun-15
leighmon215
lipizzaner
Longhorn
NNecromancer
obenk


PING? PONG!
CONNECTED SINCE '95

The workshop

Step 0: setting up

Clone the repo

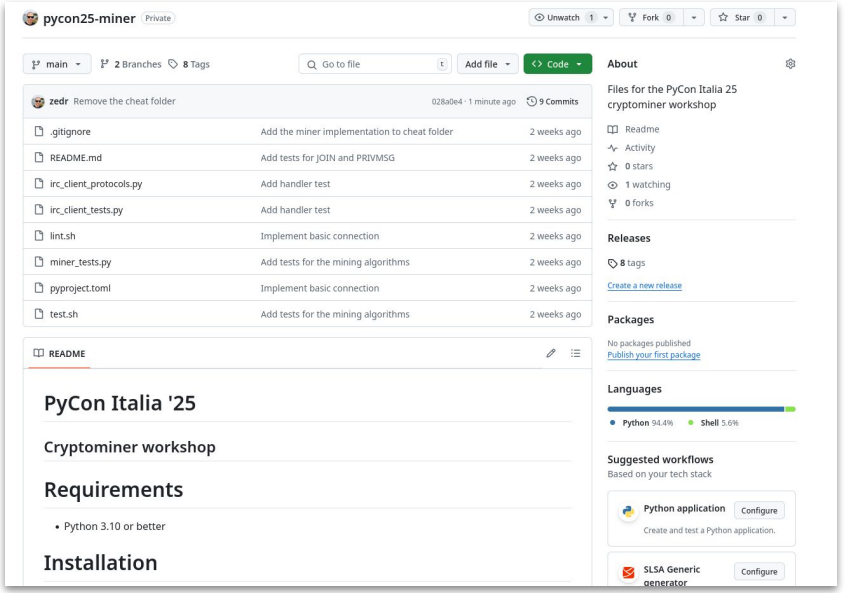
Visit:
https://github.com/zedr/pycon25-miner/

```
git clone
git@github.com:zedr/pycon25-miner.git
```

Then reset back to **step-0**

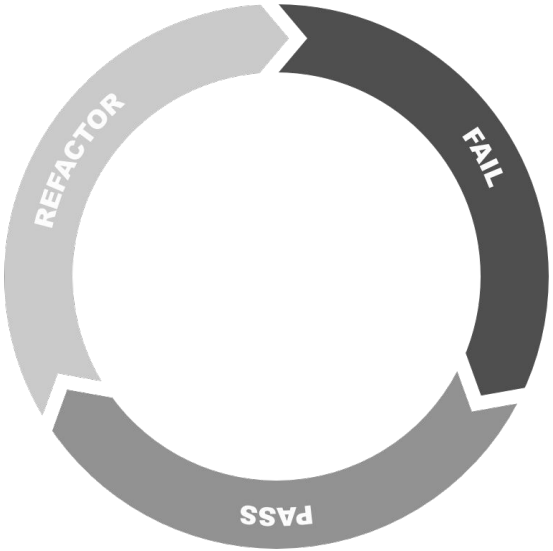
```
git reset --hard step-0
```

Create a new file for your client and name it
`irc_client.py`



TDD - Test Driven Development

1. Write a test that fails
2. Implement the code so that the test passes
3. Refactor



Step 1: connecting

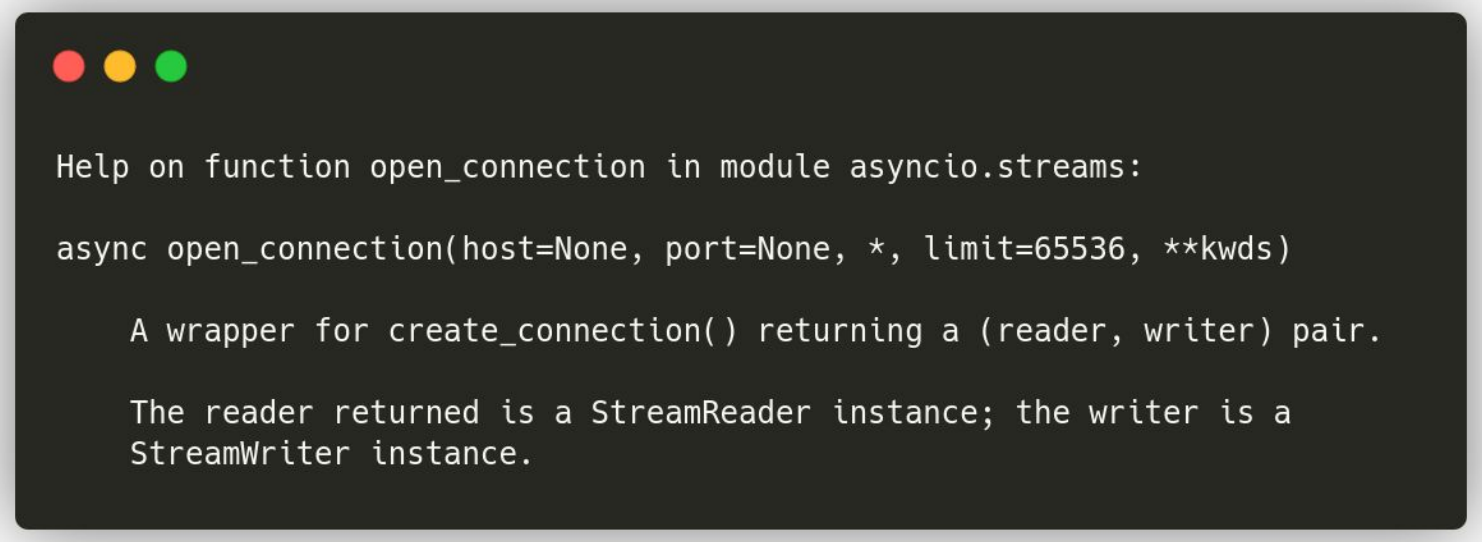
```
git reset --hard step-1
```

Cheat by copying `cheat/irc_client.py`

The async IRC client protocol

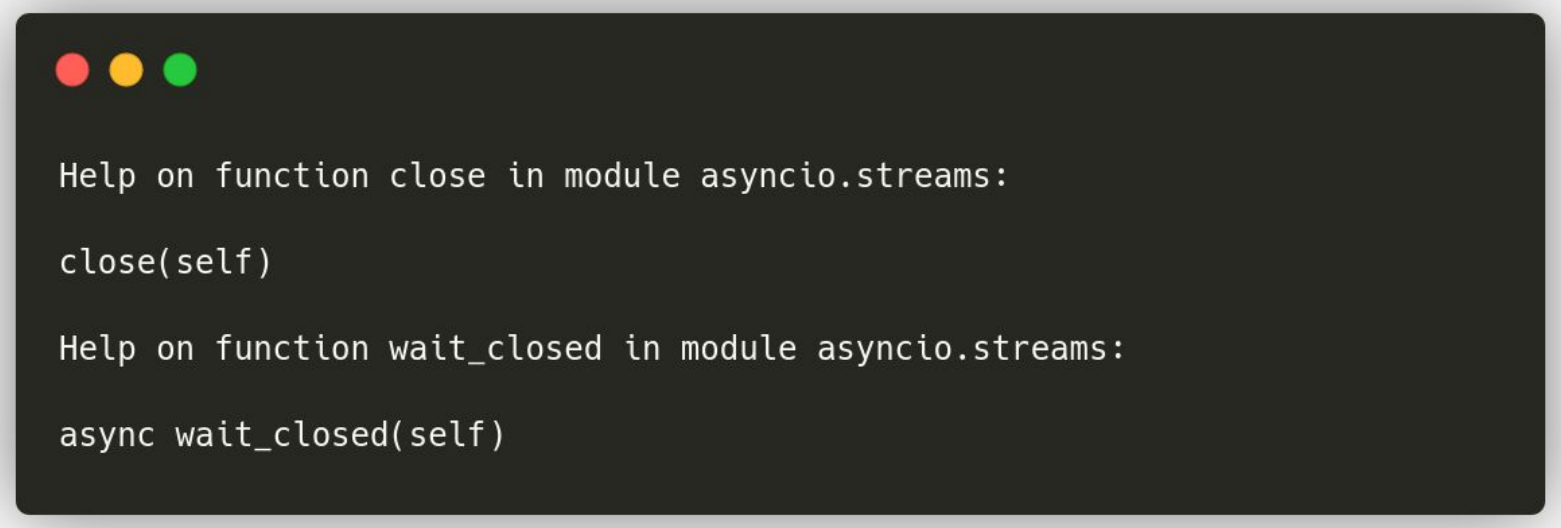
```python
class AsyncIrcClientProtocol(Protocol):
    """The protocol of an asynchronous IRC client."""

    writer: StreamWriter
    reader: StreamReader

    async def connect(
            self,
            server_host: str,
            server_port: int
    ) -> None:
        """Connects to an IRC server."""

    async def disconnect(self) -> None:
        """Disconnects from an IRC server."""
```

Opening the connection



```
Help on function open_connection in module asyncio.streams:

async open_connection(host=None, port=None, *, limit=65536, **kwds)

    A wrapper for create_connection() returning a (reader, writer) pair.

    The reader returned is a StreamReader instance; the writer is a
    StreamWriter instance.
```

Closing the connection (cleanly)



```
Help on function close in module asyncio.streams:

close(self)

Help on function wait_closed in module asyncio.streams:

async wait_closed(self)
```

main() is asynchronous

```
async def main():
    client = IrcClient()
    await client.connect()
    await client.disconnect()


if __name__ == "__main__":
    asyncio.run(main())
```

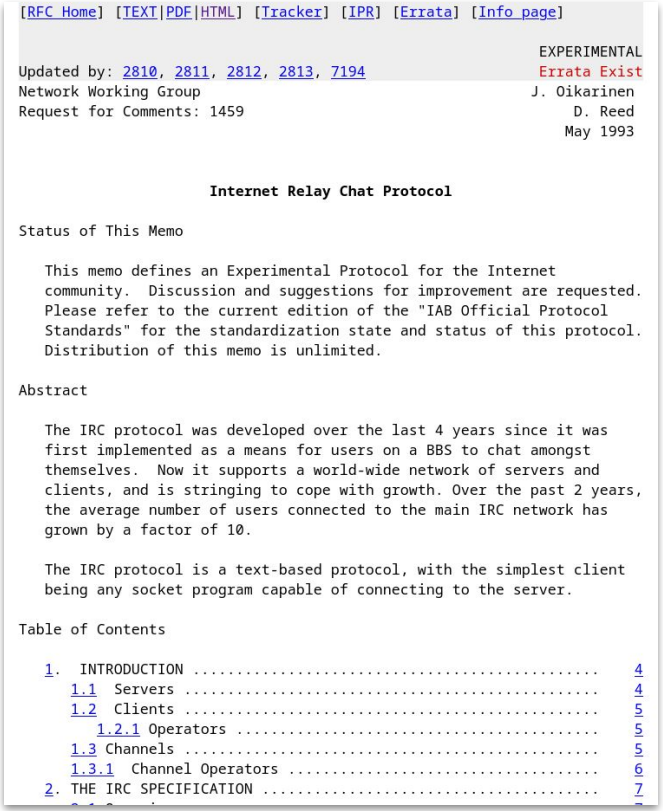
Installing the script

Installing the [dev] dependencies

Step 2: Sending INFO

```
git reset --hard step-2
```

The IRC Protocol RFC (1459)



[RFC Home] [TEXT|PDF|HTML] [Tracker] [IPR] [Errata] [Info page]

EXPERIMENTAL
Updated by: 2810, 2811, 2812, 2813, 7194 Errata Exist
Network Working Group J. Oikarinen
Request for Comments: 1459 D. Reed
May 1993

Internet Relay Chat Protocol

Status of This Memo

This memo defines an Experimental Protocol for the Internet community. Discussion and suggestions for improvement are requested. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

The IRC protocol was developed over the last 4 years since it was first implemented as a means for users on a BBS to chat amongst themselves. Now it supports a world-wide network of servers and clients, and is stringing to cope with growth. Over the past 2 years, the average number of users connected to the main IRC network has grown by a factor of 10.

The IRC protocol is a text-based protocol, with the simplest client being any socket program capable of connecting to the server.

Table of Contents

The IRC Protocol RFC (1459): messages

2.3 Messages

Servers and clients send eachother messages which may or may not generate a reply. If the message contains a valid command, as described in later sections, the client should expect a reply as specified but it is not advised to wait forever for the reply; client to server and server to server communication is essentially asynchronous in nature.

IRC messages are always lines of characters terminated with a CR-LF (Carriage Return - Line Feed) pair, and these messages shall not exceed 512 characters in length, counting all characters including the trailing CR-LF. Thus, there are 510 characters maximum allowed for the command and its parameters. There is no provision for continuation message lines. See section 7 for more details about current implementations.

\r\n

The IRC Protocol RFC (1459): INFO server message

4.3.8 Info command

```
      Command: INFO
   Parameters: [<server>]
```

The INFO command is required to return information which describes the server: its version, when it was compiled, the patchlevel, when it was started, and any other miscellaneous information which may be considered to be relevant.

```
17:26 -!-                    -/\- InspIRCd -\/-
17:26 -!-                  November 2002 - Present
17:26 -!-
17:26 -!- Core Developers:
17:26 -!-     Matt Schatz,          genius3000, <genius3000@g3k.solutions>
17:26 -!-     Sadie Powell,         SadieCat,   <sadie@witchery.services>
17:26 -!-
 [17:26] [zedr(+Riwx)] [1:libera (change with ^X)]
[(status)]
```

Sending an INFO message to the server

```
asyncio.StreamWriter.write(data: bytes) -> None
```

The method attempts to write the data to the underlying socket immediately.

If that fails, the data is queued in an internal write buffer until it can be sent.

The method should be used along with the drain() method:

```
stream.write(data)
await stream.drain()
```

Step 3: Setting USER and NICK

```
git reset --hard step-3
```

The IRC Protocol RFC (1459): USER and NICK

4.1.3 User message

```
      Command: USER
   Parameters: <username> <hostname> <servername> <realname>

   The USER message is used at the beginning of connection to specify
   the username, hostname, servername and realname of a new user.  It is
   also used in communication between servers to indicate new user
   arriving on IRC, since only after both USER and NICK have been
   received from a client does a user become registered.
```

4.1.2 Nick message

```
      Command: NICK
   Parameters: <nickname> [ <hopcount> ]

   NICK message is used to give user a nickname or change the previous
   one.  (...)

   If a NICK message arrives at a server which already knows about an
   identical nickname for another client, a nickname collision occurs.
   As a result of a nickname collision, all instances of the nickname
   are removed from the server's database, and a KILL command is issued
   to remove the nickname from all other server's database. If the NICK
   message causing the collision was a nickname change, then the
   original (old) nick must be removed as well.
```

The USER command

`USER <username> <hostname> <servername> :<realname>`

- USER → the command
- <username> → the chosen username of the client
- <hostname> → historical, ignored, set to `0`
- <servername> → historical, ignored, set to *
- <realname> → set to the same value as username

Example: `"USER zedr 0 * :zedr"`

The NICK command

`NICK <nickname>`

- NICK → the command
- <username> → the chosen nickname of the user

Example: `"NICK zedr"`

Step 4: Sending messages

```
git reset --hard step-4
```

The IRC Protocol RFC (1459): JOIN

4.2.1 Join message

```
      Command: JOIN
   Parameters: <channel>{,<channel>} [<key>{,<key>}]

   The JOIN command is used by client to start listening a specific
   channel. (...)
```

```
   Once a user has joined a channel, they receive notice about all
   commands their server receives which affect the channel.  This
   includes MODE, KICK, PART, QUIT and of course PRIVMSG/NOTICE.  The
   JOIN command needs to be broadcast to all servers so that each server
   knows where to find the users who are on the channel.  This allows
   optimal delivery of PRIVMSG/NOTICE messages to the channel.

   If a JOIN is successful, the user is then sent the channel's topic
   (using RPL_TOPIC) and the list of users who are on the channel (using
   RPL_NAMREPLY), which must include the user joining.
```

The JOIN command

`JOIN <channel_name>`

- JOIN → the command
- <channel_name> → the name of the channel

Note: channel names are prefixed by the '#' character.

Example: `"JOIN #pycon"`

The IRC Protocol RFC (1459): JOIN

4.4.1 Private messages

```
      Command: PRIVMSG
   Parameters: <receiver>{,<receiver>} <text to be sent>

   PRIVMSG is used to send private messages between users.  <receiver>
   is the nickname of the receiver of the message.  <receiver> can also
   be a list of names or channels separated with commas.
```

The PRIVMSG command

`PRIVMSG <receiver>`

- PRIVMSG → the command
- <receiver> → the name of the channel or nick

Example: `"JOIN #pycon"`

Step 5: Receiving messages

```
git reset --hard step-5
```

Sending an INFO message to the server

```
asyncio.StreamWriter.write(data: bytes) -> None
```

The method attempts to write the data to the underlying socket immediately.

If that fails, the data is queued in an internal write buffer until it can be sent.

The method should be used along with the drain() method:

```
stream.write(data)
await stream.drain()
```

Readling inbound streams

`asyncio.StreamReader.readline()`

Read one line, where “line” is a sequence of bytes ending with `\n`.

If EOF is received and `\n` was not found, the method returns partially read data.

If EOF is received and the internal buffer is empty, return an empty bytes object.

The IrcMessageHandler

```
IrcMessageHandler = Callable[     # A callable object (e.g. function)
  [
      str,                        # Identity of the sender
      str,                        # Command
      list[str]                   # Sequence of strings containing message text
  ],
  Awaitable[                      # Returns an awaitable object (e.g. Coroutine or Task)
      Optional[                   # The awaitable object might return None
            bool                  # or a boolean value
      ]
  ]
]
```

Example: implementation

```python
async def echo(src: str, cmd: str, msgs: list[str]) -> None:
    print(src, cmd, msgs)
```

Example: execution

```
$ ./irc_client.py satoshi
INFO:root:(':92b5aba5d4a5', '001', ['satoshi', ':Hi,', 'welcome', 'to', 'IRC'])
INFO:root:(':92b5aba5d4a5', '002', ['satoshi', ':Your', 'host', 'is', '92b5aba5d4a5,', 'running', 'version', 'miniircd-2.3'])
INFO:root:(':92b5aba5d4a5', '003', ['satoshi', ':This', 'server', 'was', 'created', 'sometime'])

...

INFO:root:(':zedr_!zedr@192.168.1.67', 'PRIVMSG', ['#pycon', ':hi'])
```

Step 6: mining

```
git reset --hard step-6
```

How to combat spam (in the 90s)

“An anti-spam mechanism that required email senders to perform a small computational task, effectively proving that they expended resources (in the form of CPU time) before sending an email.”

HashCash (Adam Back, 1997)

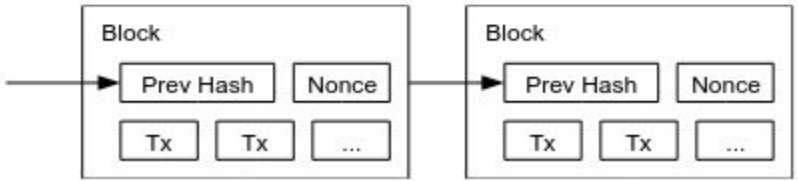
Wikipedia

Satoshi's Proof of Work (2008)

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The algorithm

1. Pick a number, the nonce, e.g. 42
2. `while True`:
3. Get the hash of "`{nonce}:{transaction_message}`"

   e.g.
   `0038f85f66bc64702320d8cb9cb702914e210fc4756e158660976cc9df664c0`

4. Does the hash begin with difficulty * number (2) of **zeros**?
5. If **yes**: break -> you mined the transaction! Broadcast immediately!
6. If **no**: increment the nonce by 1 and loop

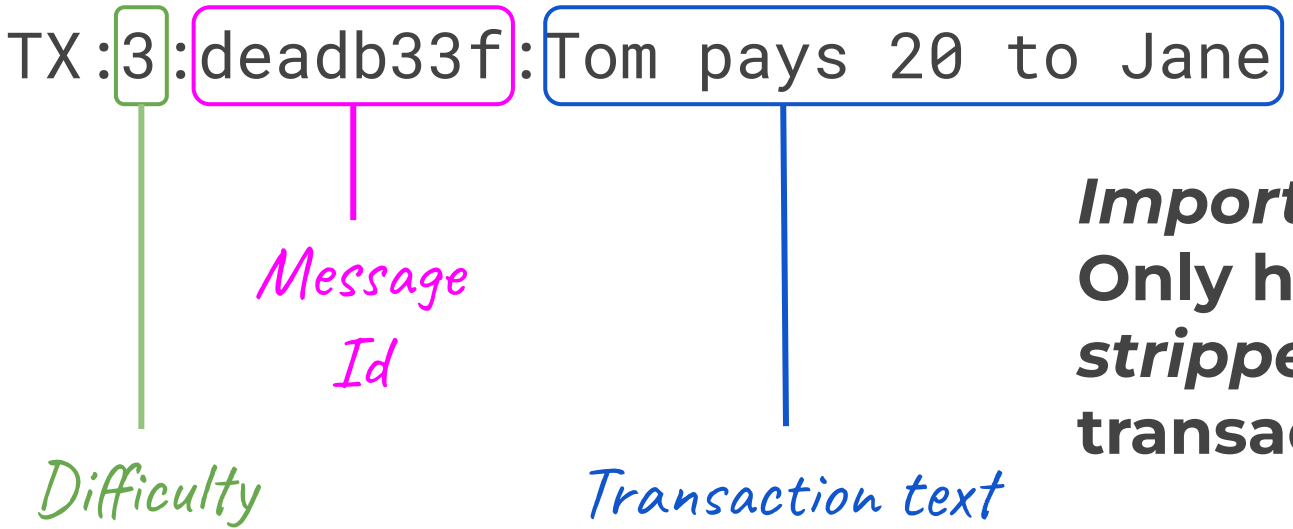
Final step: putting it all together

```
git reset --hard step-final
```

The broadcaster

An IRC client that *broadcasts* transaction messages.

Messages have this format:

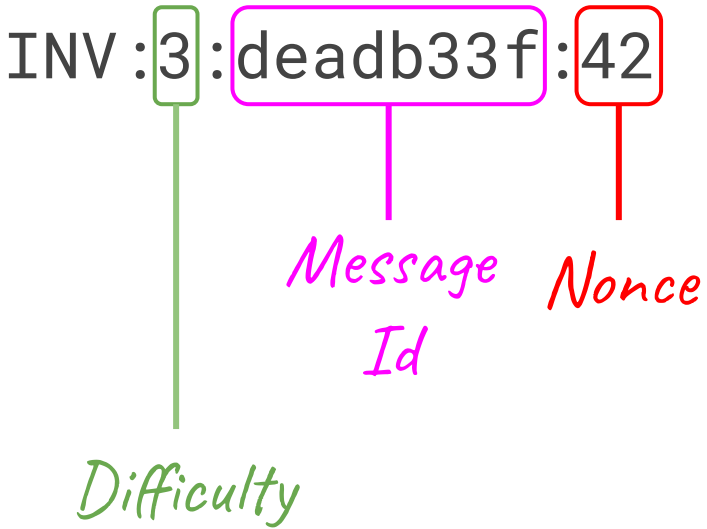


Important:
Only hash the *stripped* transaction text!

The miner

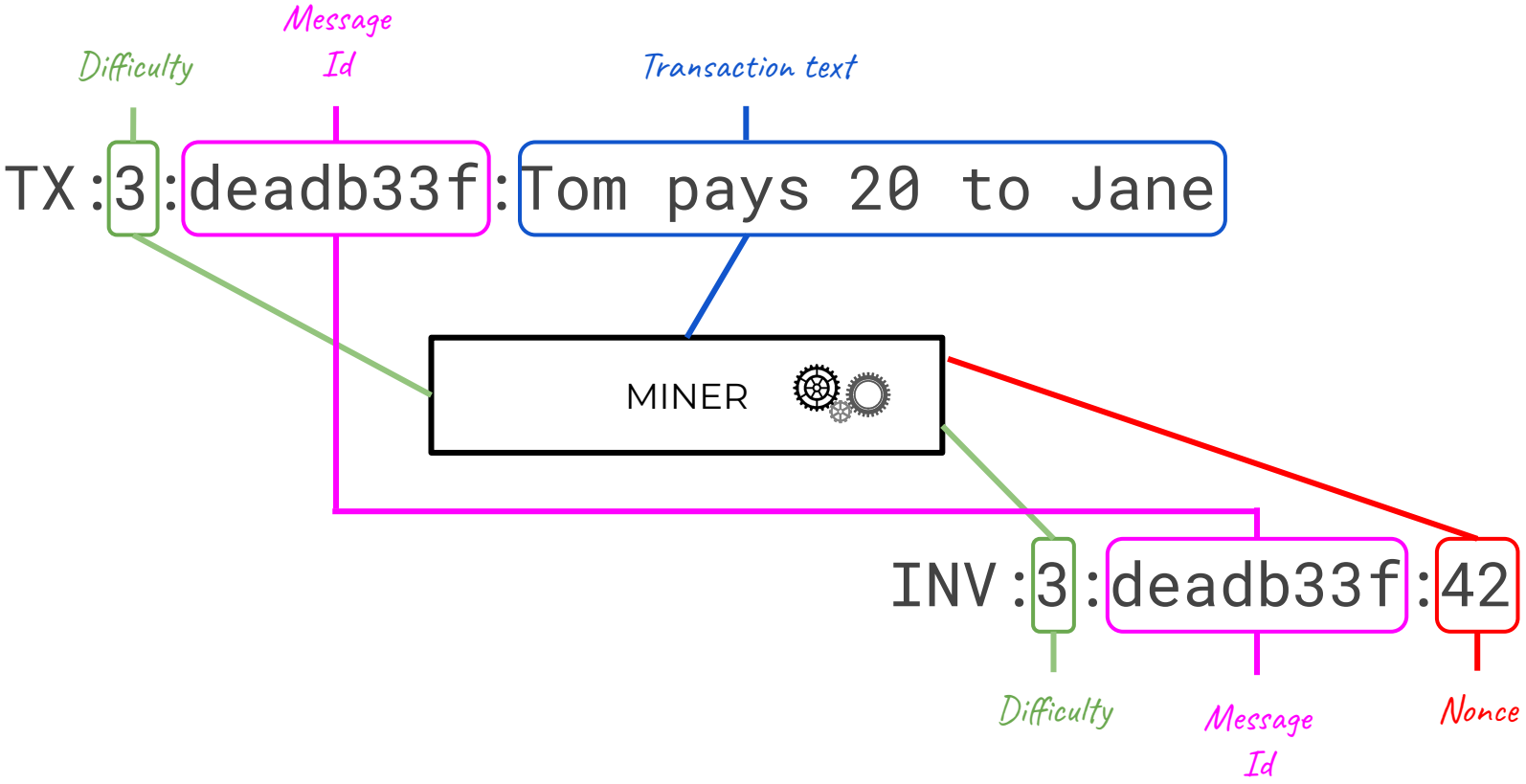
An IRC client that *mines* transaction messages.

Once mined, clients send a message in this format:



Important:
Only hash the *stripped* transaction text!

Recap


Difficulty
Message Id
Transaction text
TX:3:deadb33f:Tom pays 20 to Jane
MINER
INV:3:deadb33f:42
Difficulty
Message Id
Nonce

Let’s get to work