Using Bitcoin-CLI

with Ordinals and Inscriptions

Table of Contents

Synopsis

The Ordinals project (https://github.com/casey/ord) is a command line tool for creating inscriptions on the Bitcoin blockchain using taproot transactions. It also creates, maintains, and depends on an index of pertinent information for inscriptions associated to ordinals. It uses Bitcoin Core as the backend for wallet operations taking special care to structure transactions so as not to inadvertently lose references to inscriptions in the UTXOs produced.

Sometimes there comes a need to perform more complex transactions that the Ord tool does not currently support. In these cases, you can build the transaction using the bitcoin-cli tool, but then its up to you to ensure that the UTXOs for inscriptions retain their ordinal references.

This document is intended to cover some common scenarios for transaction building and by the end you should have an increased knowledge of how to use the tools for your wallet.

Change History

2023-02-13	Initial Document
2023-02-18	Removed header as its noisy. Cleaned up document.

Prerequisites

In general this guide assumes that you have already setup a full Bitcoin node, and the Ord client tool.

You should also be familiar with logging into your node via SSH if you are working from Windows or Mac and accessing a Bitcoin Node running on Linux (e.g. Raspberry Pi, other local servers or remote in Cloud providers). If you haven't yet done that, do so now and then return to this guide.

Launching BitcoinD or Bitcoin-QT

Whether you are running Bitcoin as a daemon service or with the GUI, what you will need to ensure is that Bitcoin is listening and accepting to RPC calls. This is a very common setup with Bitcoind, but not so common with Bitcoin-QT.

Bitcoin-QT

If you are using Bitcoin-QT, open the properties for the shortcut you use to launch Bitcoin-QT. The shortcut from the Start Menu can be found in

`%APPDATA%\Microsoft\Windows\Start Menu\Programs\Bitcoin Core`

If you open that path in Windows Explorer, right click Bitcoin Core (64-bit), and choose Properties. In the Target field that appears, make sure you have the command line option "-server" specified. If you made any changes, press OK to close the window, and then stop the Bitcoin-QT application before launching it again.

BitcoinD

For BitcoinD, you should have the following settings somewhere in your bitcoin.conf

```
server=1
listen=1
txindex=1
```

If you had to make any changes, restart the service daemon.

Setting up to use Bitcoin-CLI

You should also have access to bitcoin-cli tool on your path, your data directory and cookie and be familiar with your wallet names.

On Linux

- Your bitcoin-cli should be on your path, and is commonly located in `/usr/local/bin`
- Your .cookie file is created in the Bitcoin Data Directory unless otherwise specified via command line arguments or bitcoin.conf config files. By default, the data directory will be at `~/.bitcoin`

On Windows

- A typical install will place bitcoin-cli in the `C:\Program Files\Bitcoin\daemon` folder.
- Bitcoin-cli won't be on your path unless you add the above to your path. You can do this in windows by either of the following
  - If you want to always have the path set when you open a command prompt window. Press WINKEY+X, select System. On the right hand side of About screen, choose Advanced system settings. On the System Properties dialog that opens activate the Advanced tab, and click the Environment Variables… button near the bottom. Select the row for the variable named "Path" under your user. Click Edit… Click the New button on the dialog and provide the path: `C:\Program Files\Bitcoin\daemon`. Press OK to close out of the dialogs and close the System window.
  - If you want to manually set the path each time you open a command prompt window, then use this command: `PATH="C:\Program Files\Bitcoin\daemon";%PATH%`
- If you kept your data directory as the default on the C: drive, then your .cookie file should be located in either
  `%APPDATA%\Bitcoin\.cookie`
  or
  `C:\Program Files\Bitcoin\.cookie`
  If you set it to a secondary drive, such as an attached USB drive mounted on E: then look for the .cookie file in the same folder as the data directory.

On Mac / OSX

Note: I don't have a Mac or access to one to test and verify. If you find this information create, please let me know so I can remove this notice. If it's incorrect, let me know that too so we can fix it!

- From a typical installation, the bitcoin-cli binary should be located in
- I believe that the .cookie file would be found within the data directory at `~/Library/Application Support/Bitcoin/.cookie`

To use the Bitcoin-CLI tool, you need to have a terminal window open

Linux:

- If you're aren't at the command line already…
- If using Gnome as your desktop, press CTRL+ALT+T
- Otherwise search through Applications and Utilities menus to find the Terminal app.

Windows:

- Press WINKEY+R, type "cmd", and press <enter>

Mac/OSX:

- Press COMMAND+SPACEBAR, Type "Terminal" and open the app

See that you can report the version and blockchain information with the bitcoin-cli tool

```
$ bitcoin-cli -version
Bitcoin Core RPC client version v24.0.1
Copyright (C) 2009-2022 The Bitcoin Core developers

Please contribute if you find Bitcoin Core useful. Visit
<https://bitcoincore.org/> for further information about the software.
The source code is available from <https://github.com/bitcoin/bitcoin>.

This is experimental software.
Distributed under the MIT software license, see the accompanying file COPYING
or <https://opensource.org/licenses/MIT>

$ bitcoin-cli getblockchaininfo
{
  "chain": "main",
  "blocks": 776290,
  "headers": 776290,
  "bestblockhash":
"000000000000000000016ad7fa34a22065617f4017b9b0086bf332f0fd0763d7",
  "difficulty": 39156400059293.19,
  "time": 1676265366,
  "mediantime": 1676260636,
```

```
  "verificationprogress": 0.9999969185304435,
  "initialblockdownload": false,
  "chainwork":
"0000000000000000000000000000000000000000004008221e022d5c09712a428a",
  "size_on_disk": 517144909482,
  "pruned": false,
  "warnings": ""
}
```

If either of these give an error like the following

error: Could not locate RPC credentials. No authentication cookie could be found, and RPC password is not set. See -rpcpassword and -stdinrpcpass.

Then you can try to provide your cookie file location as a command line option as follows

```
C:\>bitcoin-cli -rpccookiefile=”D:\Bitcoin1\.cookie” getblockchaininfo
{
  "chain": "main",
  "blocks": 776291,
  "headers": 776291,
  "bestblockhash":
"00000000000000000000105274feae2520ca59b9a813ab2430ab1f95589611a1f",
  "difficulty": 39156400059293.19,
  "time": 1676266308,
  "mediantime": 1676261533,
  "verificationprogress": 0.9999986297382245,
  "initialblockdownload": false,
  "chainwork":
"0000000000000000000000000000000000000000400845baf51b429486bc1898",
  "size_on_disk": 517111567434,
  "pruned": false,
  "warnings": ""
}
```

List Your Wallets with Bitcoin-CLI

You want to make sure you can see your Ordinal wallet. This guide will assume that you created a default wallet with ord tool named ord. That tool lets you optionally specify a different wallet name to work with by using the `--wallet` option. You should see your expected wallets with the following commands.

```
$ bitcoin-cli listwallets
[
  "ord"
]
```

At this point, your prerequisites are all met and you are setup to use bitcoin-cli in the following sections.

Compare Ord to Bitcoin-CLI for UTXOs

In Ord, you can list your unspent transaction outputs (UTXOs) as follows. Here I'm explicitly specifying the wallet to use with the `--wallet` command line option.

```
$ ord --wallet ord wallet outputs
[
  {
    "output": "93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415:0",
    "amount": 10000
  },
  {
    "output": "51721487e92ecc3a1a0ead8614e8cdee7a221b30e65af6aa2a58d3a097a1134d:0",
    "amount": 10000
  },
  {
    "output": "8a378655928a2041056bdd442d7b70c2622a94b5d78296c5a28c15eb0dea3657:1",
    "amount": 184434
  }
]
```

Note that this includes not only your inscriptions, but also any cardinal transaction outputs such as change or initial funding that can be used for inscribing.

Compare it to the output from listing unspent transactions for the wallet using bitcoin-cli. I use the `-rpcwallet` command line option to constrain to a specific wallet

```
$ bitcoin-cli -rpcwallet=ord listunspent
[
  {
    "txid": "93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415",
    "vout": 0,
    "address": "bc1pqq7hkfxdaqfq82qfwrazakrj9y5jlqv0jfw4llw3vylatt00asyqd3namu",
    "amount": 0.00010000,
    "confirmations": 501,
    "spendable": true,
    ...
  },
  {
    "txid": "8a378655928a2041056bdd442d7b70c2622a94b5d78296c5a28c15eb0dea3657",
    "vout": 1,
    "address": "bc1pmllm6y4jfgjnt93s229gf0fp6yyr9ls6cpnw83zadrl6gqssm56shkq69y",
    "amount": 0.00184434,
    "confirmations": 501,
    "spendable": true,
    ...
  },
  {
    "txid": "51721487e92ecc3a1a0ead8614e8cdee7a221b30e65af6aa2a58d3a097a1134d",
    "vout": 0,
    "address": "bc1p4782cw43rq3dnaqcqkut9neny9jae46x8rfspn92a8h90z4te37qwxtal7",
    "amount": 0.00010000,
    "confirmations": 546,
    "spendable": true,
    ...
  }
]
```

I've removed some extra fields for brevity, but you can see here that the output field from ord matches the txid and vout values from bitcoin-cli, and the amount in ord reflected as sats matches the floating point value of bitcoin from bitcoin-cli.

From this I know I'm working with the same data.

List Inscriptions in Ordinals Wallet

In Ord, you can list your inscriptions. Here I'm explicitly specifying the wallet to use with the --wallet command line option. These equate to UTXOs that you need to be careful with when working with bitcoin-cli directly

```
$ ord --wallet ord wallet inscriptions
[
  {
    "inscription": "93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415i0",
    "location": "93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415:0:0",
    "explorer":
"https://ordinals.com/inscription/93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415i0"
  },
  {
    "inscription": "51721487e92ecc3a1a0ead8614e8cdee7a221b30e65af6aa2a58d3a097a1134di0",
    "location": "51721487e92ecc3a1a0ead8614e8cdee7a221b30e65af6aa2a58d3a097a1134d:0:0",
    "explorer":
"https://ordinals.com/inscription/51721487e92ecc3a1a0ead8614e8cdee7a221b30e65af6aa2a58d3a097a1134di0"
  }
]
```

Bitcoin isn't ordinal aware, so there's no direct way to do a comparative command. However, if you assume that an inscription is always on outpoint 0, and always has the amount 10000, then you can find matches by piping the output of the listunpent command through the json query (JQ) command line parsing tool if its installed. The following is an example on Linux

```
$ bitcoin-cli -rpcwallet=ord listunspent | jq '.[]|select((amount == 0.00010000)
and (.vout ==0))|[.txid,.vout,.address,.amount]'
[
  "93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415",
  0,
  "bc1pqq7hkfxdaqfq82qfwrazakrj9y5jlqv0jfw4llw3vylatt00asyqd3namu",
  0.0001
]
[
  "51721487e92ecc3a1a0ead8614e8cdee7a221b30e65af6aa2a58d3a097a1134d",
  0,
  "bc1p4782cw43rq3dnaqcqkut9neny9jae46x8rfspn92a8h90z4te37qwxtal7",
  0.0001
]
```

Note that this isn't foolproof, particular after inscriptions are sent from one person to another as often fees for the transaction will be paid from the amount of sats remaining on the inscription.

Guidance for Sending Inscriptions

If you are sending a single inscription to a single address, it is easiest and least risky to simply use the ord tool. Details for this feature are available via ord wallet send –help

When using bitcoin-cli to prepare transactions that involve inscriptions, you should follow these rules

1. Preferably transfer one inscription at a time as its easier to avoid messing up.
2. Whether you are transferring one inscription, or multiple inscriptions, its best to list the inscription inputs before any other inputs used to cover fees
3. The destination for inscriptions should be listed in the outputs in the same order as their input before any other outputs, including for overall change.
4. You can add change addresses between inscriptions in the output if changing the amount of the inscription
5. A UTXO in a transaction is only listed once on the input side
6. An address in a transaction is only referenced once on the output side
7. Inputs are spent in order to outputs. All sats of an inscription input should be accounted for in outputs before any subsequent inscription inputs.
8. Double and Triple check the structure of the transaction before signing and broadcasting
    1. With bitcoin-cli you'll use the `decoderawtransaction` option
    2. If you want a visual viewer, you can load the transaction in Sparrow Wallet

Generating Addresses

For some transactions, we need to have created addresses. For this guide, I've used the getnewaddress command and created 3 addresses that will be referenced in the examples, and labeled them. In all cases, the address type is "bech32m" indicating the new taproot address format.

Here's a basic example of creating an address in the wallet that I can send Bitcoin to to pay mining fees

```
$ bitcoin-cli -rpcwallet=ord getnewaddress "receiving funds" "bech32m"
bc1p255h3gw7s98spjO7y8h9v96tj2s4t6wmOaj5utwqa455y35wmpgslnle7q
```

This address created for receiving an inscription. In examples, I'm creating to send to myself

```
$ bitcoin-cli -rpcwallet=ord getnewaddress "receiving inscription" "bech32m"
bc1pjm7Oexly5h9vpadlcOkuuejlw3ervOyOlxpcyf36w523tarmkndq9z68eq
```

And this address I'll use for my change address in transactions I build

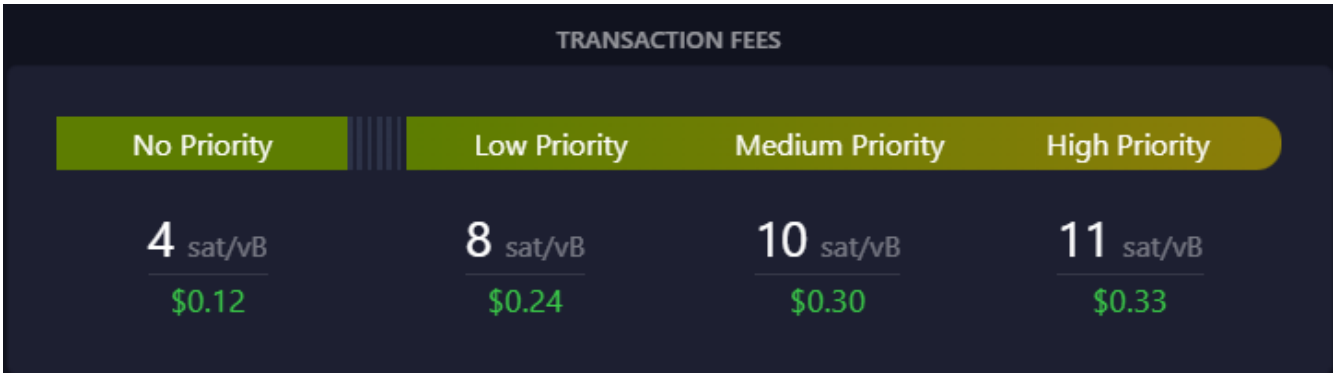
```
$ bitcoin-cli -rpcwallet=ord getnewaddress "change from sending" "bech32m"
bc1p4rentm5fajxj6fu7md8ssxzcz9syh23OOdvqky735szyqtzkqpes8n9ylw
```

Estimating Fees to Pay for Transactions

Long gone are the days of 0 fee transactions. And with inscriptions being made and other activity on the network, the fee pressure is a bid on the precious blockspace. Thankfully over time, improved algorithms have made estimating fees easier then ever, without grossly overpaying.

Some background: If you are new to Bitcoin, but have experience working with other systems like Ethereum, you'll already be familiar with the notion of "gas". In Bitcoin, fees for transactions are paid from the same Bitcoin you hold from the inputs referenced in the transaction, and the rate of fees are often referred to as sats/vB or sats/vByte. A sat is the smallest unit of a Bitcoin (There are 100,000,000 sats per 1 whole BTC). A vB is a virtual byte within a block. Blocks are limited to a virtual size of 1 MB, and can offload the witness data for transactions to increase the potential overall size of a block to up to about 4MB. We'll look at how to estimate the fees for sat/vB first, and then determine the size, or the number of vB we need to pay for when creating the transactions in later sections.

If you are using Mempool.Space to estimate fees, you can open up that site in a browser. Look for the Transaction fees block on the page. An example at the time of this writing looks like this



For the next block (about 10 minutes), it recommends 11 sat/vB. Within the next few blocks (about 20 minutes), it recommends 10 sat/vB. As your time preference is lowered, the rate can be lower, but also less certain. This is because the estimates are based upon what is known about the mempool now, and doesn't take into account the potential for new transactions to be added to the pool with higher bidding.

You can also get fee estimates using the Bitcoin-CLI tool with the estimatesmartfee command

```
$ bitcoin-cli estimatesmartfee 1
{
  "feerate": 0.00018111,
  "blocks": 2
}
```

The feerate provided is in Bitcoin. The above indicates a fee rate of 18111 sats per vKB (1000 vBytes) should be sufficient to get a transaction confirmed in about 2 blocks. In other words, it's suggesting 18 sat/vB.

How do I Know How Big a Transaction Will Be?

Your transaction will be created based on inputs and outputs. The inputs are signed based on what it takes to satisfy conditions for being allowed to spend them. Different address types as outputs take up different amounts of space. Transactions making use of segregated witness get the benefit of lower virtual size in the block, resulting in lower fees. If you want to optimize the fees while following this guide, you may need to first create and sign the transaction to see its overall size. After that you can recreate the transaction with adjusted change amounts accordingly to target the fee rate desired for getting into a block.

Creating Transactions for Sending Inscriptions

We will use the bitcoin-cli tool to assemble a transaction using the `createrawtransaction` option. This will be demonstrated in the subsections that follow.

1 Input – 1 Output / Simple Send

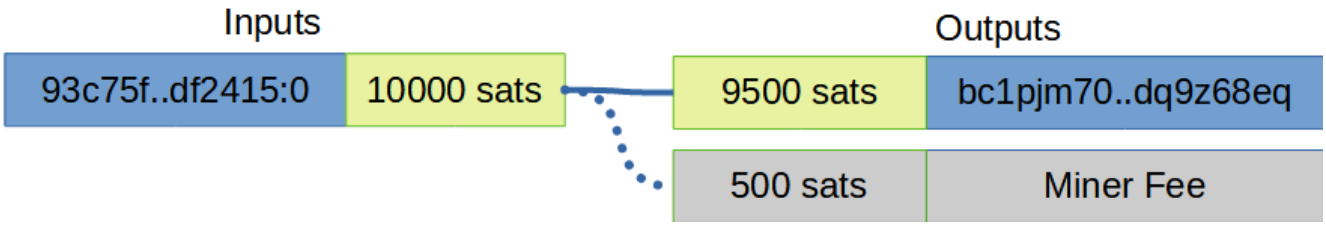
The simplest method of transferring an inscription to another address is if there is only one input, and one output. In this case, fees for transaction are paid from the remaining value on the inscription. Inscriptions that are created using Ord will start with 10000 sats output value for the specific purpose of facilitating transfers like this.

To prepare you transaction you need to know the following

- The txid of the inscription you want to send and the vout point of that inscription.
- The address you want to send to.
- How much of the amount you want to send to the address.
  - You should deduct the fee being paid from the amount of the input inscription. A suitable fee can be determined by looking at the rates on mempool.space, or using the estimatesmartfee command of bitcoin-cli
  - Note that the unallocated amount in a transaction results in the fees used to pay for the transaction. If you want change from this transaction, look at the 1 Input – Many Output type.

As an example from my wallet, I will choose the following

- Inscription (with an amount balance of 10000)
  - txid: 93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415
  - vout: 0
- Send To
  - address: bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq
  - amount: 9500 sats or 0.00009500 BTC



This will result in 10000 sats being included as input, and 9500 sats being assigned in outputs. The remaining 500 sats will be implicitly given as fees to the miner that includes it in a block. To create the transaction, I call the following command, referencing the inscription information as input in the first argument, and the address and amount to assign as output in the second argument:

```
$ bitcoin-cli createrawtransaction
"[{\"txid\":\"93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415\",
\"vout\":0}]"
"[{\"bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq\":
0.00009500}]"

02000000011524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c
5fc7930000000000fdffffff011c2500000000000022512096fcfc9be4a5cac0f5bf
c3edce665f7472363c8ff98382263a751515f47bb4da00000000
```

The output is a hexadecimal string of the raw transaction. The length of this string (divided by 2) is the size of the transaction. In this case, its currently 94 bytes long. From the 500 sats we didn't allocate, this yields a rate of 500 / 94 or 5.31 sats/vB. It will be longer after a signature is provided.

Decoding a Transaction

From any transaction in hexadecimal format, we can also decode the transaction to see details. From the hexadecimal output, I provide it as input as follows:

```
$ bitcoin-cli decoderawtransaction \
02000000011524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c
5fc7930000000000fdffffff011c2500000000000022512096fcfc9be4a5cac0f5bf
c3edce665f7472363c8ff98382263a751515f47bb4da00000000

{
  "txid": "e4e3ec06ebb987c3c317e037f75ea50052fd3655bd6d9725db3a92b923f83d23",
  "hash": "e4e3ec06ebb987c3c317e037f75ea50052fd3655bd6d9725db3a92b923f83d23",
  "version": 2,
  "size": 94,
  "vsize": 94,
  "weight": 376,
  "locktime": 0,
  "vin": [
    {
      "txid": "93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415",
      "vout": 0,
      "scriptSig": {
        "asm": "",
        "hex": ""
      },
      "sequence": 4294967293
    }
  ],
  "vout": [
    {
      "value": 0.00009500,
      "n": 0,
```

```
      "scriptPubKey": {
        "asm": "1 96fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da",
        "desc": "rawtr(96fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da)#ds5ns5ll",
        "hex": "512096fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da",
        "address": "bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq",
        "type": "witness_v1_taproot"
      }
    }
  ]
}
```

There is a lot of additional information here. For example, the output denotes a generated transaction id, the version, size and vsize, weight etc. For this transaction, the vsize and weight are the same right now before the transaction is signed.

Signing a Transaction

Prior to broadcasting a transaction to the mempool, we must sign the transaction. Here I specify the wallet I want to use and the command to sign the transaction

```
$ bitcoin-cli -rpcwallet=ord signrawtransactionwithwallet \
020000000011524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c5fc7930000000
000fdffffff011c2500000000000022512096fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382
263a751515f47bb4da00000000

{
  "hex":
"020000000001011524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c5fc79300
00000000fdffffff011c2500000000000022512096fcfc9be4a5cac0f5bfc3edce665f7472363c8ff
98382263a751515f47bb4da01403373770ede09b0ccec70d09e020685a0e34b29ac5dba225904b993
f72956040d2203a09031fdb93a826c3921a03f9968f154ebd1175b763732e5b109369fa6ba0000000
0",
  "complete": true
}
```

The output is a revised hexadecimal representation of the transaction that includes the signature, and the “complete” field indicates that the transaction has all necessary signatures and ready for broadcasting. Let’s decode the transaction again to see what changed

```
$ bitcoin-cli decoderawtransaction \
020000000001011524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c5fc793000
0000000fdffffff011c2500000000000022512096fcfc9be4a5cac0f5bfc3edce665f7472363c8ff9
8382263a751515f47bb4da01403373770ede09b0ccec70d09e020685a0e34b29ac5dba225904b993f
72956040d2203a09031fdb93a826c3921a03f9968f154ebd1175b763732e5b109369fa6ba00000000

{
  "txid": "e4e3ec06ebb987c3c317e037f75ea50052fd3655bd6d9725db3a92b923f83d23",
  "hash": "9a3fff2fa9b92ed11a27b7f0051a9874a38e9304802b6ddc3af6876a7e47446e",
  "version": 2,
  "size": 162,
  "vsize": 111,
  "weight": 444,
  "locktime": 0,
  "vin": [
    {
      "txid": "93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415",
      "vout": 0,
      "scriptSig": {
        "asm": "",
        "hex": ""
      },
```

```
      "txinwitness": [

"3373770ede09b0ccec70d09e020685a0e34b29ac5dba225904b993f72956040d2203a09031fdb93a826c3921a03f9968f154e
bd1175b763732e5b109369fa6ba"
      ],
      "sequence": 4294967293
    }
  ],
  "vout": [
    {
      "value": 0.00009500,
      "n": 0,
      "scriptPubKey": {
        "asm": "1 96fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da",
        "desc": "rawtr(96fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da)#ds5ns5ll",
        "hex": "512096fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da",
        "address": "bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq",
        "type": "witness_v1_taproot"
      }
    }
  ]
}
```

The txid has stayed the same, but the hash has changed. The size has increased, as well as the vsize, but now they are different. Relatedly, the weight also increased and reflects the vsize x 4. The txinwitness field has been populated in our input. This represents the signature and is why the transaction size increased.

With a vsize of 111, our 500 sats that are going to fees now result in a fee rate of 4.50 sats/vB.

Broadcasting a Transaction

One a transaction is completed and fully signed, it's ready to be broadcast to the network. You can do that with the sendrawtransaction command. Anyone can broadcast an already signed transaction which pushes in the local mempool for peer to peer propagation.

```
$ bitcoin-cli sendrawtransaction \
020000000001031524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c5fc7930000000000fdffffff4d13a1
97a0d3582aaaf65ae6301b227aeecde81486ad0e1a3acc2ee98714725100000000000fdffffff5b17d6453976bcc79d99074100
0aebf9b2fe4192172e384efe0084b4038624460000000000fdffffff04e80300000000000022512096fcfc9be4a5cac0f5bfc3
edce665f7472363c8ff98382263a751515f47bb4da2823000000000000022512079770 6b6dc3a99f954f7005f4ae48d40e96070
e7fcce9383cbb75e441fb78d67d00700000000000022512 0fb0ba5e52fc03f8bc1ec7b4e5453a5dd7889defc79d9e494f4ec22
9322993db3c023000000000000022512 0a8f335ee89ec8d2d279edb4f08185811604baa2f7b580b13d1a404402c560073014 01f
c714e308edfe42aea94ad6829c57c15250dddaffec714d5db38d07183d2da3fc47f9332b0de828e8132a204248b5d1925f6f36
b7933a8f2662baacc1396cdd0140ba6e7cda7468a935dffc98c58d06f9975bc2c153c66a70cf2c3d9ed27a28e7218359eb7c1a
a464dd9fab0507654182698 0bf9ad10d9042cdfe9fb4f0b7828efb014029ff5dcba258c1d2377e7020390bbb65bbcc9afce2ed5
63043b52de591d7fb0710bc01b653ab482db566b01f73c263e6e524e1abd9325e6688d41646d24c6d2bb00000000

1c3037a4fdddd445ade899ff3172831599e1258de5dcf2853bc04dac8862b788
```

The response from the broadcast is the transaction id. You can look this transaction up in a block explorer.

You may optionally broadcast a transaction through a service provider online. Some websites that accept transactions for broadcast include the following. Some offer onion endpoints.

- https://mempool.space/tx/push

- https://blockchair.com/broadcast
- https://blockstream.info/tx/push
- https://live.blockcypher.com/btc/pushtx/
- https://www.blockchain.com/explorer/assets/btc/broadcast-transaction
- https://app.mycrypto.com/broadcast-transaction

Many Inputs – 1 Output / Increasing amount of sats on Inscription

This is a common scenario where you either want to add additional value to an inscription, or pay fees for the inscription from additional UTXOs so as not to reduce the value assigned the inscription.

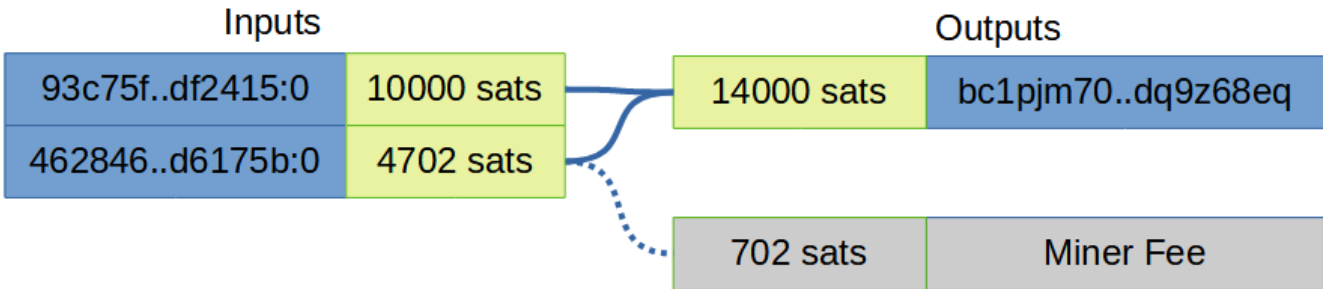
Only one inscription should be included in this type of transaction as we only have 1 output. If you want to send many inscriptions at the same time to different recipients, look to the Many Inputs – Many Output transaction type.

To prepare you transaction you need to know the following

- The txid of the inscription you want to send and the vout point of that inscription.
- Another non-inscription UTXO you can use to pay for fees
- The address you want to send to.
- How much of the amount you want to send to the address.

As an example from my wallet, I will choose the following

- Inputs
  - Inscription. Its amount value is 10000
    - txid: 93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415
    - vout: 0
  - UTXO with additional sats to cover fees and increase amount assigned to inscriptions. It's amount value is 4702
    - txid: 46248603b48400fe4e382e179241feb2f9eb0a0d4107999dc7bc763945d6175b
    - vout: 0
- Outputs
  - address: bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq
    - amount: 14000 sats or 0.00014000 BTC



This will result in 14702 sats being included as input, and 14000 sats being assigned in outputs. The remaining 702 sats will be implicitly given as fees to the miner that includes it in a block.

To create the transaction, I call the following command, referencing the inscription information as the first input in the first argument, and the address and amount to assign as output in the second argument:

```
$ bitcoin-cli createrawtransaction
"[{\"txid\":\"93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415\",
\"vout\":0},
{\"txid\":\"46248603b48400fe4e382e179241feb2f9eb0a0d4107999dc7bc763945d6175b\",\"
vout\":0}]"
"[{\"bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq\":
0.00014000}]"

02000000021524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c
5fc7930000000000fdffffff5b17d6453976bcc79d9907410d0aebf9b2fe4192172e
384efe0084b4038624460000000000fdffffff01b0360000000000000022512096fcfc
9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da00000000
```

We can decode the transaction to see its details and verify the order of inputs is as we provided

```
$ bitcoin-cli decoderawtransaction \
02000000021524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c
5fc7930000000000fdffffff5b17d6453976bcc79d9907410d0aebf9b2fe4192172e
384efe0084b4038624460000000000fdffffff01b0360000000000000022512096fcfc
9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da00000000

{
  "txid": "c4d4d4f493cc46f21db9d8b3fa8ccc7e03c6fd7cb46c8478da178623a3a3962f",
  "hash": "c4d4d4f493cc46f21db9d8b3fa8ccc7e03c6fd7cb46c8478da178623a3a3962f",
  "version": 2,
  "size": 135,
  "vsize": 135,
  "weight": 540,
  "locktime": 0,
  "vin": [
    {
      "txid": "93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415",
      "vout": 0,
      "scriptSig": {
        "asm": "",
        "hex": ""
      },
      "sequence": 4294967293
    },
    {
      "txid": "46248603b48400fe4e382e179241feb2f9eb0a0d4107999dc7bc763945d6175b",
      "vout": 0,
      "scriptSig": {
        "asm": "",
        "hex": ""
```

```
      },
      "sequence": 4294967293
    }
  ],
  "vout": [
    {
      "value": 0.00014000,
      "n": 0,
      "scriptPubKey": {
        "asm": "1 96fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da",
        "desc": "rawtr(96fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da)#ds5ns5ll",
        "hex": "512096fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da",
        "address": "bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq",
        "type": "witness_v1_taproot"
      }
    }
  ]
}
```


The size of this transaction before signing is 135 bytes. As compared to the 1 to 1 transaction, this is 41 bytes longer. The more inputs and outputs on a transaction, the more efficient it will be. After signed and decoded following steps outlined previously, each transaction input has the txinwitness data added, and the overall size of the transaction increased to 269 bytes, with a vsize of 169. The 702 sats set aside for fees results in an effective fee rate of 702 / 169 = 4.15 sats/vB

Follow the same steps to sign and broadcast a transaction of this type.

1 Input – Many Outputs / Reducing sats on Inscription

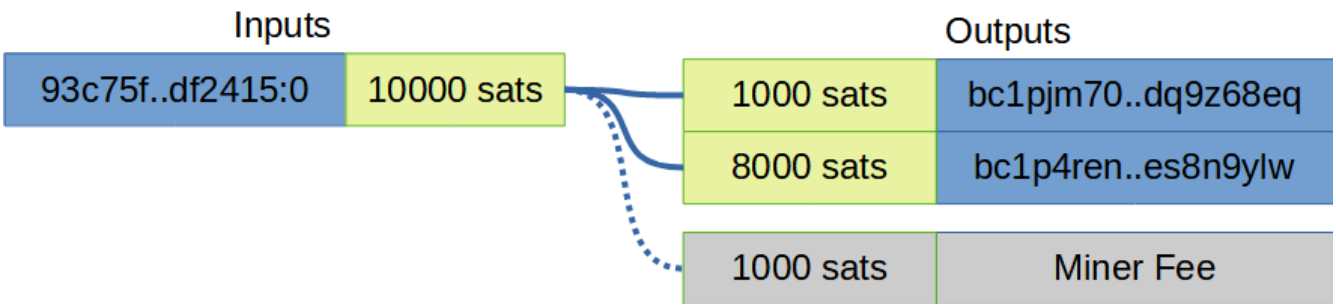
In this scenario you may want to retain value from an inscription instead of sending all of the sats. While less common then others, its important to know how this is done. As inscriptions are by default created with 10000 sats, you may decide that some of that value could be used for other purposes, particularly if you intend to hold the inscription for quite some time.

To prepare you transaction you need to know the following

- The txid of the inscription you want to send and the vout point of that inscription.
- The address you want to send the inscription to (either another person, or in your own wallet)
- The new amount of sats to allocate to the inscription
- An address to use for the change, and the amount to send there

As an example from my wallet, I will choose the following

- Inputs
  - Inscription. Its amount value is 10000
    - txid: 93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415
    - vout: 0
- Outputs
  - Send Inscription to
    - address: bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq
    - amount: 1000 sats or 0.00001000 BTC
  - Send Change to
    - address: bc1p4rentm5fajxj6fu7md8ssxzcz9syh2300dvqky735szyqtzkqpes8n9ylw
    - amount: 8000 sats or 0.00008000 BTC



This will result in 10000 sats being included as input, and 9000 as outputs. The remaining 1000 sats will be implicitly given as fees to the miner that includes it in a block.

To create the transaction, I call the following command, referencing the inscription information as the first input in the first argument, and the address and amount to assign as the first output in the second argument, followed by my intended change address and its amount:

```
$ bitcoin-cli createrawtransaction
"[{\"txid\":\"93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415\",
\"vout\":0}]"
"[{\"bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq\":
0.00001000},{\"bc1p4rentm5fajxj6fu7md8ssxzcz9syh2300dvqky735szyqtzkqpes8n9ylw\":
0.00008000}]"
020000000011524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c5fc7930000000
000fdffffff02e8030000000000002512096fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382
263a751515f47bb4da401f0000000000000225120a8f335ee89ec8d2d279edb4f08185811604baa2f7
b580b13d1a404402c56007300000000
```

Decoded, we see the current size is 137 before signing and the order of outputs is as desired

```
$ bitcoin-cli decoderawtransaction \
020000000011524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c5fc7930000000
000fdffffff02e8030000000000002512096fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382
263a751515f47bb4da401f0000000000000225120a8f335ee89ec8d2d279edb4f08185811604baa2f7
b580b13d1a404402c56007300000000
{
  "txid": "d7d0345e73304dfa85d4023289dc67daca4c6151448fcce786cf3ad271db9c35",
  "hash": "d7d0345e73304dfa85d4023289dc67daca4c6151448fcce786cf3ad271db9c35",
  "version": 2,
  "size": 137,
  "vsize": 137,
  "weight": 548,
  "locktime": 0,
  "vin": [
    {
      "txid": "93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415",
      "vout": 0,
      "scriptSig": {
        "asm": "",
        "hex": ""
      },
      "sequence": 4294967293
    }
  ],
  "vout": [
    {
      "value": 0.00001000,
      "n": 0,
      "scriptPubKey": {
```

```
        "asm": "1 96fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da",
        "desc": "rawtr(96fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da)#ds5ns5ll",
        "hex": "512096fcfc9be4a5cac0f5bfc3edce665f7472363c8ff98382263a751515f47bb4da",
        "address": "bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq",
        "type": "witness_v1_taproot"
      }
    },
    {
      "value": 0.00008000,
      "n": 1,
      "scriptPubKey": {
        "asm": "1 a8f335ee89ec8d2d279edb4f08185811604baa2f7b580b13d1a404402c560073",
        "desc": "rawtr(a8f335ee89ec8d2d279edb4f08185811604baa2f7b580b13d1a404402c560073)#0spwmm0c",
        "hex": "5120a8f335ee89ec8d2d279edb4f08185811604baa2f7b580b13d1a404402c560073",
        "address": "bc1p4rentm5fajxj6fu7md8ssxzcz9syh2300dvqky735szyqtzkqpes8n9ylw",
        "type": "witness_v1_taproot"
      }
    }
  ]
}
```


After signed and decoded following steps outlined previously, each transaction input has the txinwitness data added, and the overall size of the transaction increased to 205 with a vsize of 154. From the 1000 sats set aside to cover fees, this results in 1000 / 154 = 6.49 sats/vB to be paid.

Many Inputs – Many Outputs / Bulk distribution of Inscriptions

One of the benefits of Bitcoin is doing multiple activities together as a batch to save on transaction fees. In this example, we have multiple inscriptions that we need to send to different recipient addresses. I want to keep the same amount of sats assigned to each inscription as this makes it easier to prepare the transaction. As such, will include a non-inscription based UTXO to cover the miner fees, as well as a change address for the balance that I don't want to give to miners. Once again, the order of inputs and outputs matters.

To prepare you transaction you need to know the following

- For each inscription being sent
  - The txid and the vout point of that inscription
  - The current amount of sats on that inscription.
  - The recipient address to send to
- Another non-inscription UTXO that will be used to pay mining fees to simplify the transaction
  - The txid and the vout of the UTXO
  - The current amount of sats on that UTXO
- An address to use for change and the amount that should be sent there.

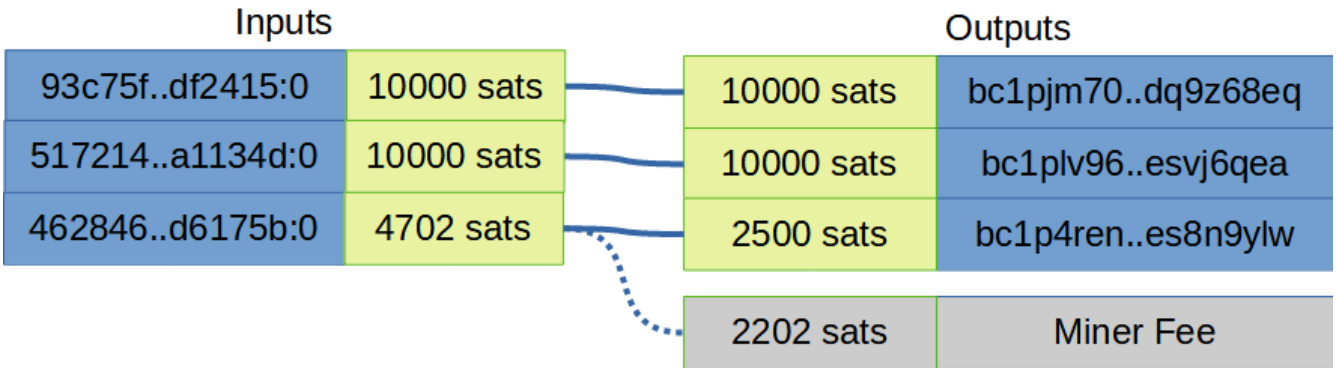
As an example from my wallet, I will choose the following

- Inscription 41492.
  - txid: 93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415
  - vout: 0
  - amount: 10000
  - Send to address: bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq
- Inscription 38220
  - txid: 51721487e92ecc3a1a0ead8614e8cdee7a221b30e65af6aa2a58d3a097a1134d
  - vout: 0
  - amount: 10000

    - send to address: bc1plv96tef0cqlchs0v0d89g5a9m4ugnhhu08v7f985as3fxg5e8kesvj6qea
- UTXO with additional sats to cover fees
    - txid: 46248603b48400fe4e382e179241feb2f9eb0a0d4107999dc7bc763945d6175b
    - vout: 0
    - amount: 4702
- Where to send Change
    - send to address: bc1p4rentm5fajxj6fu7md8ssxzcz9syh2300dvqky735szyqtzkqpes8n9ylw
    - amount: 2500

The inputs (10000 + 10000 + 4702) minus the outputs (10000 + 10000 + 2500) will yield the mining fee of 2202 sats

A visualization of the transaction I plan to create



To create the transaction, I call the following command, including all my inscriptions to send in order as the inputs, and my additional UTXO to cover fees. The second argument contains the addresses to send the inscriptions to, and an additional output address for my change:

```
$ bitcoin-cli createrawtransaction
"[{\"txid\":\"93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415\",
\"vout\":0},
{\"txid\":\"51721487e92ecc3a1a0ead8614e8cdee7a221b30e65af6aa2a58d3a097a1134d\",\"
vout\":0},
{\"txid\":\"46248603b48400fe4e382e179241feb2f9eb0a0d4107999dc7bc763945d6175b\",\"
vout\":0}]"
"[{\"bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq\":
0.00010000},{\"bc1plv96tef0cqlchs0v0d89g5a9m4ugnhhu08v7f985as3fxg5e8kesvj6qea\":
0.00010000},{\"bc1p4rentm5fajxj6fu7md8ssxzcz9syh2300dvqky735szyqtzkqpes8n9ylw\":
0.00002500}]"

020000000031524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c5fc7930000000
000fdffffff4d13a197a0d3582aaaf65ae6301b227aeecde81486ad0e1a3acc2ee987147251000000
0000fdffffff5b17d6453976bcc79d9907410d0aebf9b2fe4192172e384efe0084b40386244600000
00000fdffffff03102700000000000022512096fcfc9be4a5cac0f5bfc3edce665f7472363c8ff983
82263a751515f47bb4da102700000000000022512 0fb0ba5e52fc03f8bc1ec7b4e5453a5dd7889def
c79d9e494f4ec229322993db3c4090000000000000225120a8f335ee89ec8d2d279edb4f0818581160
4baa2f7b580b13d1a404402c56007300000000
```

Before signing, the transaction size is 262, and its vsize is the same.

After signing the transaction and decoding the output, this transaction ends up with a size of 462, and vsize of 312. As 2202 sats were allocated to miner fee, this yields 2202 / 312 = 7.05 sats/vB.

Many Inputs – Many Outputs and reduce size of Inscription Output Amounts

The common use case here is that you've created multiple inscriptions with ord, and they each have 10000 sats. You plan to send them to addresses, but retain some of the sats instead of transferring the full 10000 to the recipient. Whatever your economic needs, you can build the transaction with the bitcoin-cli tool. Please note that this type of transaction is a little more complex and risky as you must understand the ordinal theory of the order of sats in inputs and outputs as they are applied.

To prepare you transaction you need to know the following

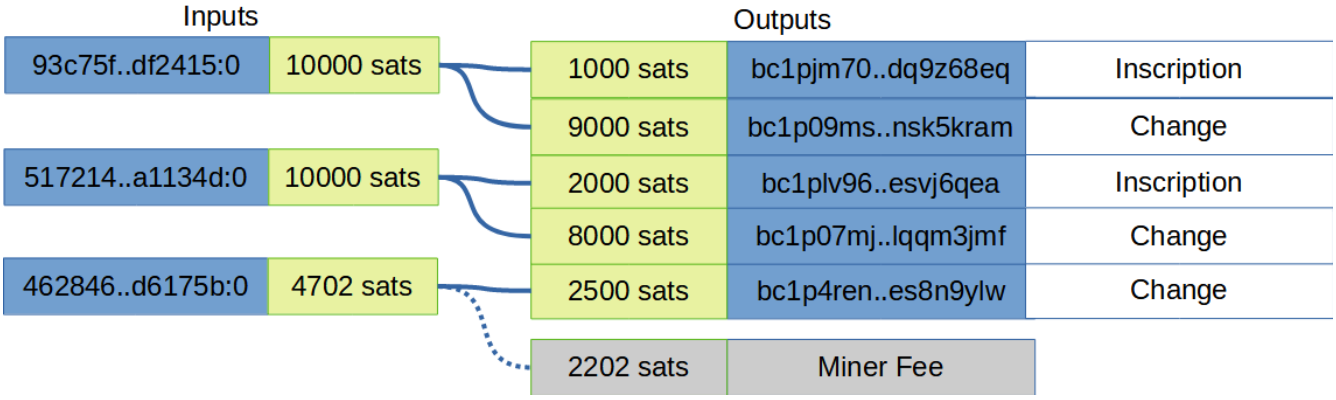
- For each inscription being sent
  - The txid and the vout point of that inscription
  - The current amount of sats on that inscription.
  - The recipient address to send to
  - The amount that should end up on the recipient address. This amount must be higher than dust limits. For ease of future transactions, strongly encourage values no less than 1000
  - An address in your own wallet for the difference between the inscription amount. A unique address must be used for each inscription being sent, and the amount must be higher than the dust limit.
- Another non-inscription UTXO that will be used to pay mining fees to simplify the transaction
  - The txid and the vout of the UTXO
  - The current amount of sats on that UTXO
- An address to use for change and the amount that should be sent there.

As an example from my wallet, I will choose the following

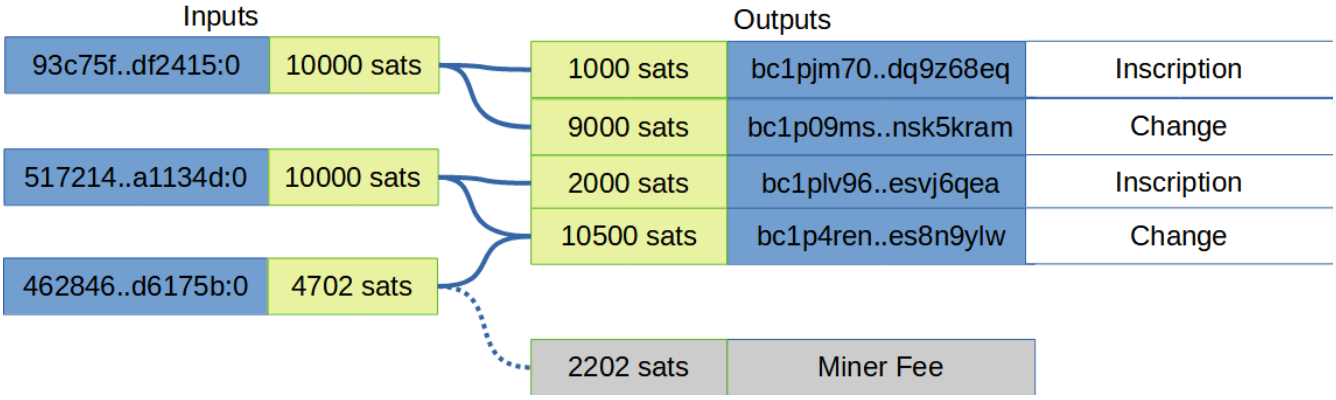
- Inscription 41492.
  - txid: 93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415
  - vout: 0
  - amount: 10000
  - Send to address: bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq
  - Amount to send: 1000
  - Change to: bc1p09msddku82vlj48hqp054eydgr5kqu88ln8f8q7tka0yg8ah34nsk5kram
  - Amount of change: 9000
- Inscription 38220
  - txid: 51721487e92ecc3a1a0ead8614e8cdee7a221b30e65af6aa2a58d3a097a1134d
  - vout: 0
  - amount: 10000
  - send to address: bc1plv96tef0cqlchs0v0d89g5a9m4ugnhhu08v7f985as3fxg5e8kesvj6qea
  - Amount to send: 2000
  - Change to: bc1p07mjp32xcvzkxw7wjll765sxm99up0q0f3mhe7nm0vkf06u9mylqqm3jmf
  - Amount of change: 8000
- UTXO with additional sats to cover fees

    - txid: 46248603b48400fe4e382e179241feb2f9eb0a0d4107999dc7bc763945d6175b
    - vout: 0
    - amount: 4702
- Where to send remaining Change
    - send to address: bc1p4rentm5fajxj6fu7md8ssxzcz9syh2300dvqky735szyqtzkqpes8n9ylw
    - amount: 2500

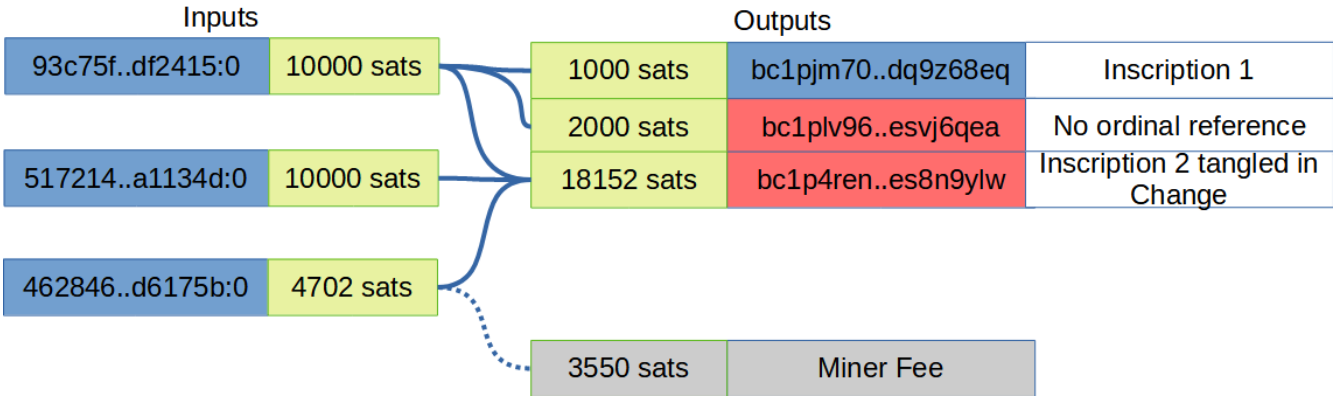
A visualization of the transaction I plan to create. There are 3 inputs, and 5 outputs



The last two outputs are both change to my wallet though. I can actually combine these, reducing the size of the overall transaction. I'll combine the 8000 + 2500 into one change output using the bc1p4ren..es8n9ylw address. Now its visualized as follows with 3 inputs, 4 outputs.



To be clear, I cannot combine the change into a single output. This is because the 1000 sats for the first inscription output leaves 9000 sats of the inscription input that needs to be handled before address outputs for the second inscription. If I attempted to do this, it would combine two inscriptions into one output and a repair transaction would be necessary to detangle. The ordinal for the second inscription would actually end up at #7001 of the 18152 sats in the change output amount.

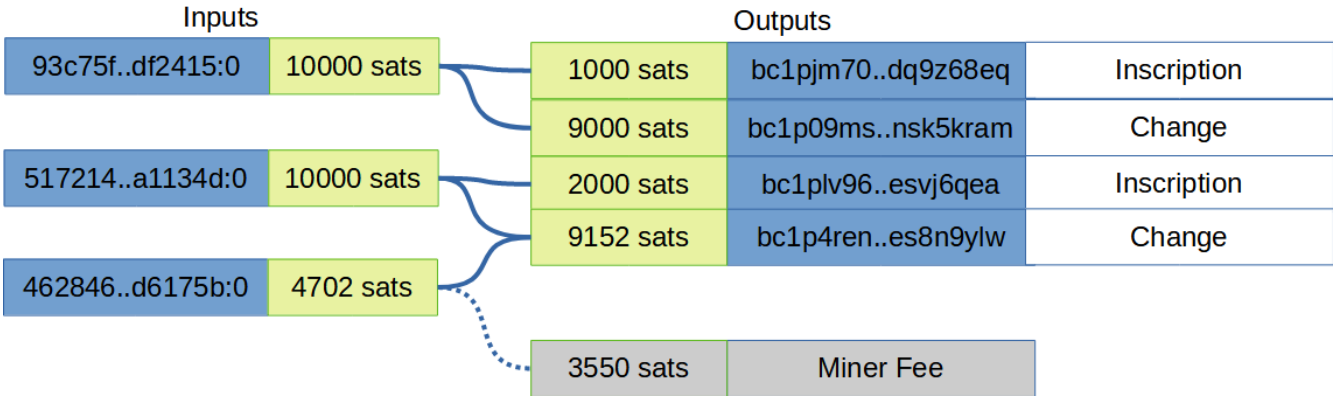


To create the correct transaction, I call the following command, including all my inscriptions to send in order as the inputs, and my additional UTXO to cover fees as before. The second argument contains the addresses to send the inscriptions to interspersed with change addresses for the remainder I want sent to me:

```
$ bitcoin-cli createrawtransaction
"[{\"txid\":\"93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415\",
\"vout\":0},
{\"txid\":\"51721487e92ecc3a1a0ead8614e8cdee7a221b30e65af6aa2a58d3a097a1134d\",\"
vout\":0},
{\"txid\":\"46248603b48400fe4e382e179241feb2f9eb0a0d4107999dc7bc763945d6175b\",\"
vout\":0}]"
"[{\"bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq\":
0.00001000},{\"bc1p09msddku82vlj48hqp054eydgr5kqu88ln8f8q7tka0yg8ah34nsk5kram\":
0.00009000},{\"bc1plv96tef0cqlchs0v0d89g5a9m4ugnhhu08v7f985as3fxg5e8kesvj6qea\":
0.00002000},{\"bc1p4rentm5fajxj6fu7md8ssxzcz9syh230odvqky735szyqtzkqpes8n9ylw\":
0.00010500}]"

020000000031524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c5fc7930000000
000fdffffff4d13a197a0d3582aaaf65ae6301b227aeecde81486ad0e1a3acc2ee987147251000000
0000fdffffff5b17d6453976bcc79d9907410d0aebf9b2fe4192172e384efe0084b40386244600000
00000fdffffff04e80300000000000022512096fcfc9be4a5cac0f5bfc3edce665f7472363c8ff983
82263a751515f47bb4da2823000000000000225120797706b6dc3a99f954f7005f4ae48d40e96070e
7fcce9383cbb75e441fb78d67d00700000000000225120fb0ba5e52fc03f8bc1ec7b4e5453a5dd78
89defc79d9e494f4ec229322993db3042900000000000225120a8f335ee89ec8d2d279edb4f08185
811604baa2f7b580b13d1a404402c56007300000000
```

The transaction size is 305. After signing the transaction it's size is 505 with a vsize of 355. With the 2202 sats set aside for miner fees, that works out to 6.20 sats/vB. I want to target a fee rate of about 10 sats/vB. Knowing that the vsize will end up as 355, I should plan for 3550 sats being sent to miners. I adjust my transaction accordingly:



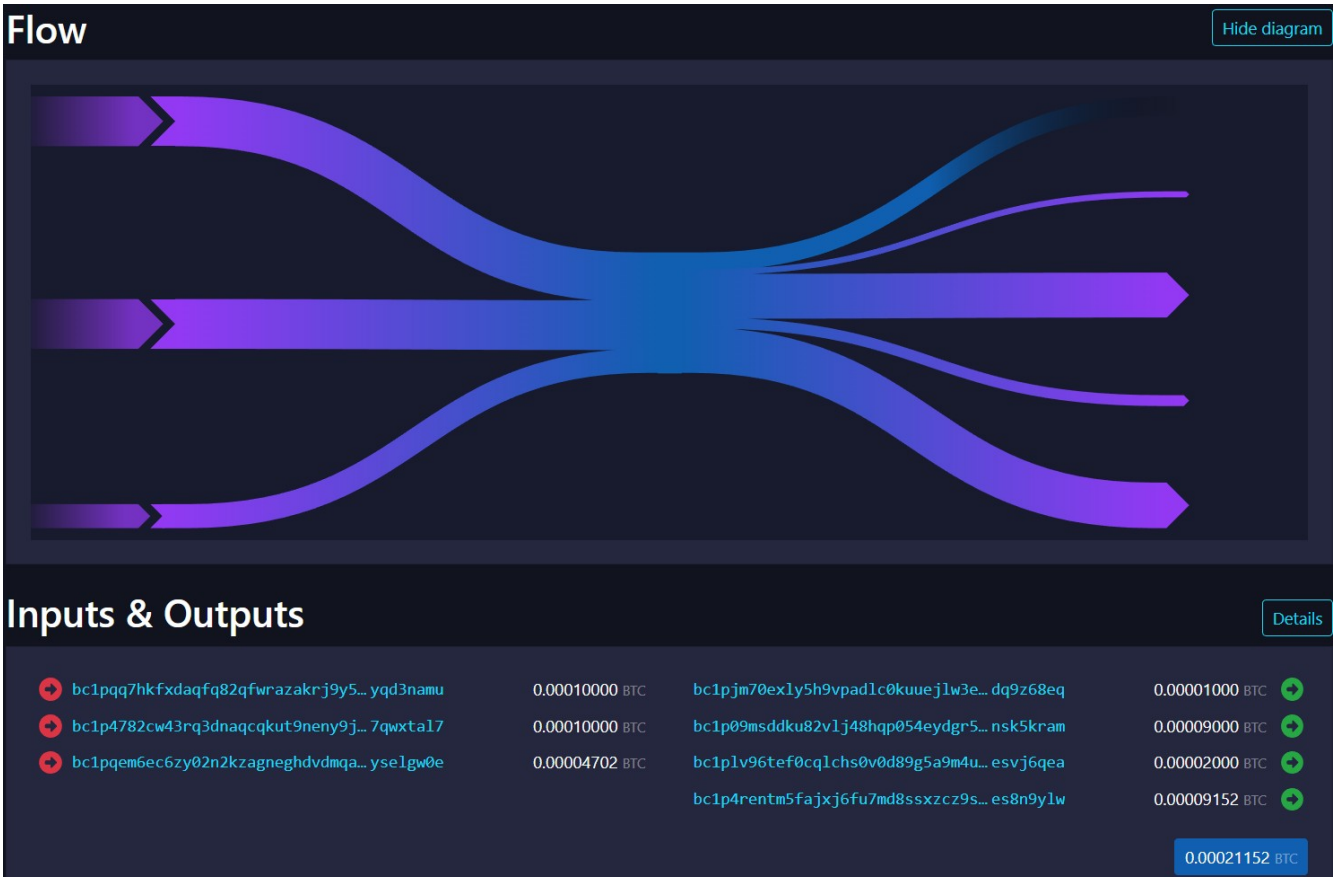
And then I create as follows

```
$ bitcoin-cli createrawtransaction
"[{\"txid\":\"93c75f1c5dadad85f20d6889ae18f43c84dc86df2ab0512eb0f09dfc46df2415\",
\"vout\":0},
{\"txid\":\"51721487e92ecc3a1a0ead8614e8cdee7a221b30e65af6aa2a58d3a097a1134d\",\"
vout\":0},
{\"txid\":\"46248603b48400fe4e382e179241feb2f9eb0a0d4107999dc7bc763945d6175b\",\"
vout\":0}]"
"[{\"bc1pjm70exly5h9vpadlc0kuuejlw3erv0y0lxpcyf36w523tarmkndq9z68eq\":
0.00001000},{\"bc1p09msddku82vlj48hqp054eydgr5kqu88ln8f8q7tka0yg8ah34nsk5kram\":
0.00009000},{\"bc1plv96tef0cqlchs0v0d89g5a9m4ugnhhu08v7f985as3fxg5e8kesvj6qea\":
0.00002000},{\"bc1p4rentm5fajxj6fu7md8ssxzcz9syh2300dvqky735szyqtzkqpes8n9ylw\":
0.00009152}]"

020000000031524df46fc9df0b02e51b02adf86dc843cf418ae89680df285adad5d1c5fc7930000000
000fdffffff4d13a197a0d3582aaaf65ae6301b227aeecde81486ad0e1a3acc2ee987147251000000
0000fdffffff5b17d6453976bcc79d9907410d0aebf9b2fe4192172e384efe0084b40386244600000
00000fdffffff04e80300000000000022512096fcfc9be4a5cac0f5bfc3edce665f7472363c8ff983
82263a751515f47bb4da2823000000000000225120797706b6dc3a99f954f7005f4ae48d40e96070e
7fcce9383cbb75e441fb78d67d00700000000000225120fb0ba5e52fc03f8bc1ec7b4e5453a5dd78
89defc79d9e494f4ec229322993db3c023000000000000225120a8f335ee89ec8d2d279edb4f08185
811604baa2f7b580b13d1a404402c56007300000000
```

Then I signed it, decoded the raw transaction to verify again.

I broadcasted the resulting transaction. You can view this transaction on the Mempool.space website at https://mempool.space/tx/1c3037a4fdddd445ade899ff3172831599e1258de5dcf2853bc04dac8862b788



Conclusion

As you can see, Bitcoin-CLI is quite useful for more advanced transaction types when sending inscriptions. It takes close some special care in building transactions to get the order of inputs and outputs correct, but you have the flexibility if you need it.

If you found this guide helpful, please consider supporting myself or others involved in Bitcoin

- To me: bc1q5smkuy7s2kuhcdnqxr3y7dqcayrsha3ks89eup
- To Casey of Ordinals: https://docs.ordinals.com/donate.html
- To Sparrow Wallet: https://sparrowwallet.com/donate/
- To Bitcoin Projects: https://opensats.org/

- @vicariousdrama