Transactions

Transaction Tutorial

Creating transactions is something most Bitcoin applications do. This section describes how to use Bitcoin Core's RPC interface to create transactions with various attributes.

Your applications may use something besides Bitcoin Core to create transactions, but in any system, you will need to provide the same kinds of data to create transactions with the same attributes as those described below.

In order to use this tutorial, you will need to setup Bitcoin Core and create a regression test mode environment with 50 BTC in your test wallet.

Simple Spending

Bitcoin Core provides several RPCs which handle all the details of spending, including creating change outputs and paying appropriate fees. Even advanced users should use these RPCs whenever possible to decrease the chance that satoshis will be lost by mistake.

```
> bitcoin-cli -regtest getnewaddress
mvbnrCX3bg1cDRUu8pkecrvP6vQkSLDSou

> NEW_ADDRESS=mvbnrCX3bg1cDRUu8pkecrvP6vQkSLDSou
```

Get a new Bitcoin address and save it in the shell variable `$NEW_ADDRESS`.

```
> bitcoin-cli -regtest sendtoaddress $NEW_ADDRESS 10.00
263c018582731ff54dc72c7d67e858c002ae298835501d80200f05753de0edf0
```

Send 10 bitcoins to the address using the "sendtoaddress" RPC. The returned hex string is the transaction identifier (txid).

The "sendtoaddress" RPC automatically selects an unspent transaction output (UTXO) from which to spend the satoshis. In this case, it withdrew the satoshis from our only available UTXO, the coinbase transaction for block #1 which matured with the creation of block #101. To spend a specific UTXO, you could use the `sendfrom` RPC instead.

```
> bitcoin-cli -regtest listunspent
[
]
```

Use the "listunspent" RPC to display the UTXOs belonging to this wallet. The list is empty because it defaults to only showing confirmed UTXOs and we just spent our only confirmed UTXO.

```
> bitcoin-cli -regtest listunspent 0
```

```
[
    {
        "txid" : "263c018582731ff54dc72c7d67e858c002ae298835501d\
                  80200f05753de0edf0",
        "vout" : 0,
        "address" : "muhtvdmsnbQEPFuEmxcChX58fGvXaaUoVt",
        "scriptPubKey" : "76a9149ba386253ea698158b6d34802bb9b550\
                          f5ce36dd88ac",
        "amount" : 40.00000000,
        "confirmations" : 0,
        "spendable" : true,
        "solvable" : true
    },
    {
        "txid" : "263c018582731ff54dc72c7d67e858c002ae298835501d\
                  80200f05753de0edf0",
        "vout" : 1,
        "address" : "mvbnrCX3bg1cDRUu8pkecrvP6vQkSLDSou",
        "account" : "",
        "scriptPubKey" : "76a914a57414e5ffae9ef5074bacbe10a320bb\
                          2614e1f388ac",
        "amount" : 10.00000000,
        "confirmations" : 0,
        "spendable" : true,
        "solvable" : true
    }
]
```


Re-running the "listunspent" RPC with the argument "0" to also display unconfirmed transactions shows that we have two UTXOs, both with the same txid. The first UTXO shown is a change output that "sendtoaddress" created using a new address from the key pool. The second UTXO shown is the spend to the address we provided. If we had spent those satoshis to someone else, that second transaction would not be displayed in our list of UTXOs.

```
> bitcoin-cli -regtest -generate 1

> unset NEW_ADDRESS
```

Create a new block to confirm the transaction above (takes less than a second) and clear the shell variable.

Simple Raw Transaction

The raw transaction RPCs allow users to create custom transactions and delay broadcasting those transactions. However, mistakes made in raw transactions may not be detected by Bitcoin Core, and a number of raw transaction users have permanently lost large numbers of satoshis, so please be careful using raw transactions on mainnet.

This subsection covers one of the simplest possible raw transactions.

```
> bitcoin-cli -regtest listunspent
```

```
[
    {
        "txid" : "263c018582731ff54dc72c7d67e858c002ae298835501d\
                  80200f05753de0edf0",
        "vout" : 0,
        "address" : "muhtvdmsnbQEPFuEmxcChX58fGvXaaUoVt",
        "scriptPubKey" : "76a9149ba386253ea698158b6d34802bb9b550\
                          f5ce36dd88ac",
        "amount" : 40.00000000,
        "confirmations" : 1,
        "spendable" : true,
        "solvable" : true
    },
    {
        "txid" : "263c018582731ff54dc72c7d67e858c002ae298835501d\
                  80200f05753de0edf0",
        "vout" : 1,
        "address" : "mvbnrCX3bg1cDRUu8pkecrvP6vQkSLDSou",
        "account" : "",
        "scriptPubKey" : "76a914a57414e5ffae9ef5074bacbe10a320bb\
                          2614e1f388ac",
        "amount" : 10.00000000,
        "confirmations" : 1,
        "spendable" : true,
        "solvable" : true
    },
    {
        "txid" : "3f4fa19803dec4d6a84fae3821da7ac7577080ef754512\
                  94e71f9b20e0ab1e7b",
        "vout" : 0,
        "address" : "mwJTL1dZG8BAP6X7Be3CNNcuVKi7Qqt7Gk",
        "scriptPubKey" : "210260a275cccf0f4b106220725be516adba27\
                          52db1bec8c5b7174c89c4c07891f88ac",
        "amount" : 50.00000000,
        "confirmations" : 101,
        "spendable" : true,
        "solvable" : true
    }
]
```


```
> UTXO_TXID=3f4fa19803dec4d6a84fae3821da7ac7577080ef75451294e71f[...]
> UTXO_VOUT=0
```

Re-run "listunspent". We now have three UTXOs: the two transactions we created before plus the coinbase transaction from block #2. We save the txid and output index number (vout) of that coinbase UTXO to shell variables.

```
> bitcoin-cli -regtest getnewaddress
mz6KvC4aoUeo6wSxtiVQTo7FDwPnkp6URG

> NEW_ADDRESS=mz6KvC4aoUeo6wSxtiVQTo7FDwPnkp6URG
```

Get a new address to use in the raw transaction.

```
## Outputs - inputs = transaction fee, so always double-check your math!
> bitcoin-cli -regtest createrawtransaction '''
    [
      {
        "txid": "'$UTXO_TXID'",
        "vout": '$UTXO_VOUT'
      }
    ]
    ''' '''
    {
      "'$NEW_ADDRESS'": 49.9999
    }'''
01000000017b1eabe0209b1fe794124575ef807057c77ada2138ae4fa8d6c4de\
0398a14f3f0000000000ffffffff01f0ca052a010000001976a914cbc20a7664\
f2f69e5355aa427045bc15e7c6c77288ac00000000

> RAW_TX=01000000017b1eabe0209b1fe794124575ef807057c77ada2138ae4[...]
```

Using two arguments to the "createrawtransaction" RPC, we create a new raw format transaction. The first argument (a JSON array) references the txid of the coinbase transaction from block #2 and the index number (0) of the output from that transaction we want to spend. The second argument (a JSON object) creates the output with the address (public key hash) and number of bitcoins we want to transfer. We save the resulting raw format transaction to a shell variable.

⚠ **Warning:** "createrawtransaction" does not automatically create change outputs, so you can easily accidentally pay a large transaction fee. In this example, our input had 50.0000 bitcoins and our output (`$NEW_ADDRESS`) is being paid 49.9999 bitcoins, so the transaction will include a fee of 0.0001 bitcoins. If we had paid `$NEW_ADDRESS` only 10 bitcoins with no other changes to this transaction, the transaction fee would be a whopping 40 bitcoins. See the Complex Raw Transaction subsection below for how to create a transaction with multiple outputs so you can send the change back to yourself.

```
> bitcoin-cli -regtest decoderawtransaction $RAW_TX
```


```
{
    "txid" : "c80b343d2ce2b5d829c2de9854c7c8d423c0e33bda264c4013\
              8d834aab4c0638",
    "hash" : "c80b343d2ce2b5d829c2de9854c7c8d423c0e33bda264c40138d834aab4c0638",
    "size" : 85,
    "vsize" : 85,
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "txid" : "3f4fa19803dec4d6a84fae3821da7ac7577080ef75\
                      451294e71f9b20e0ab1e7b",
            "vout" : 0,
            "scriptSig" : {
                "asm" : "",
                "hex" : ""
            },
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 49.99990000,
            "n" : 0,
            "scriptPubKey" : {
                "asm" : "OP_DUP OP_HASH160 cbc20a7664f2f69e5355a\
                         a427045bc15e7c6c772 OP_EQUALVERIFY OP_CHECKSIG",
                "hex" : "76a914cbc20a7664f2f69e5355aa427045bc15e\
                         7c6c77288ac",
                "reqSigs" : 1,
                "type" : "pubkeyhash",
                "addresses" : [
                    "mz6KvC4aoUeo6wSxtiVQTo7FDwPnkp6URG"
                ]
            }
        }
    ]
}
```


Use the "decoderawtransaction" RPC to see exactly what the transaction we just created does.

```
> bitcoin-cli -regtest signrawtransaction $RAW_TX
```


```
{
    "hex" : "01000000017b1eabe0209b1fe794124575ef807057c77ada213\
             8ae4fa8d6c4de0398a14f3f0000000049483045022100 8949f0\
             cb400094ad2b5eb399d59d01c14d73d8fe6e96df1a7150deb38\
             8ab8935022079656090d7f6bac4c9a94e0aad311a4268e082a7\
             25f8aeae0573fb12ff866a5f01ffffffff01f0ca052a0100000\
             01976a914cbc20a7664f2f69e5355aa427045bc15e7c6c77288\
             ac00000000",
    "complete" : true
}
```


```
> SIGNED_RAW_TX=01000000017b1eabe0209b1fe794124575ef807057c77ada[...]
```

Use the `signrawtransaction` RPC to sign the transaction created by "createrawtransaction" and save the returned "hex" raw format signed transaction to a shell variable.

Even though the transaction is now complete, the Bitcoin Core node we're connected to doesn't know anything about the transaction, nor does any other part of the network. We've created a spend, but we haven't actually spent anything because we could simply unset the `$SIGNED_RAW_TX` variable to eliminate the transaction.

```
> bitcoin-cli -regtest sendrawtransaction $SIGNED_RAW_TX
c7736a0a0046d5a8cc61c8c3c2821d4d7517f5de2bc66a966011aaa79965ffba
```

Send the signed transaction to the connected node using the "sendrawtransaction" RPC. After accepting the transaction, the node would usually then broadcast it to other peers, but we're not currently connected to other peers because we started in regtest mode.

```
> bitcoin-cli -regtest -generate 1

> unset UTXO_TXID UTXO_VOUT NEW_ADDRESS RAW_TX SIGNED_RAW_TX
```

Generate a block to confirm the transaction and clear our shell variables.

Complex Raw Transaction

In this example, we'll create a transaction with two inputs and two outputs. We'll sign each of the inputs separately, as might happen if the two inputs belonged to different people who agreed to create a transaction together (such as a CoinJoin transaction).

```
> bitcoin-cli -regtest listunspent
```

```
[
    {
        "txid" : "263c018582731ff54dc72c7d67e858c002ae298835501d\
                  80200f05753de0edf0",
        "vout" : 0,
        "address" : "muhtvdmsnbQEPFuEmxcChX58fGvXaaUoVt",
        "scriptPubKey" : "76a9149ba386253ea698158b6d34802bb9b550\
                          f5ce36dd88ac",
        "amount" : 40.00000000,
        "confirmations" : 2,
        "spendable" : true,
        "solvable" : true
    },
    {
        "txid" : "263c018582731ff54dc72c7d67e858c002ae298835501d\
                  80200f05753de0edf0",
        "vout" : 1,
        "address" : "mvbnrCX3bg1cDRUu8pkecrvP6vQkSLDSou",
        "account" : "",
        "scriptPubKey" : "76a914a57414e5ffae9ef5074bacbe10a320bb\
                          2614e1f388ac",
        "amount" : 10.00000000,
        "confirmations" : 2,
        "spendable" : true,
        "solvable" : true
    },
    {
        "txid" : "78203a8f6b529693759e1917a1b9f05670d036fbb12911\
                  0ed26be6a36de827f3",
        "vout" : 0,
        "address" : "n2KprMQm4z2vmZnPMENfbp2P1LLdAEFRjS",
        "scriptPubKey" : "210229688a74abd0d5ad3b06ddff36fa9cd8ed\
                          d181d97b9489a6adc40431fb56e1d8ac",
        "amount" : 50.00000000,
        "confirmations" : 101,
        "spendable" : true,
        "solvable" : true
    },
    {
        "txid" : "c7736a0a0046d5a8cc61c8c3c2821d4d7517f5de2bc66a\
                  966011aaa79965ffba",
        "vout" : 0,
        "address" : "mz6KvC4aoUeo6wSxtiVQTo7FDwPnkp6URG",
        "account" : "",
        "scriptPubKey" : "76a914cbc20a7664f2f69e5355aa427045bc15\
                          e7c6c77288ac",
        "amount" : 49.99990000,
        "confirmations" : 1,
        "spendable" : true,
        "solvable" : true
    }
]
```


```
> UTXO1_TXID=78203a8f6b529693759e1917a1b9f05670d036fbb129110ed26[...]
> UTXO1_VOUT=0
> UTXO1_ADDRESS=n2KprMQm4z2vmZnPMENfbp2P1LLdAEFRjS

> UTXO2_TXID=263c018582731ff54dc72c7d67e858c002ae298835501d80200[...]
> UTXO2_VOUT=0
> UTXO2_ADDRESS=muhtvdmsnbQEPFuEmxcChX58fGvXaaUoVt
```

For our two inputs, we select two UTXOs by placing the txid and output index numbers (vouts) in shell variables. We also save the addresses corresponding to the public keys (hashed or unhashed) used in those transactions. We need the addresses so we can get the corresponding private keys from our wallet.

```
> bitcoin-cli -regtest dumpprivkey $UTXO1_ADDRESS
cSp57iWuu5APuzrPGyGc4PGUeCg23PjenZPBPoUs24HtJawccHPm

> bitcoin-cli -regtest dumpprivkey $UTXO2_ADDRESS
cT26DX6Ctco7pxaUptJujRfbMS2PJvdqiSMaGaoSktHyon8kQUSg

> UTXO1_PRIVATE_KEY=cSp57iWuu5APuzrPGyGc4PGUeCg23PjenZPBPoUs24Ht[...]

> UTXO2_PRIVATE_KEY=cT26DX6Ctco7pxaUptJujRfbMS2PJvdqiSMaGaoSktHy[...]
```

Use the "dumpprivkey" RPC to get the private keys corresponding to the public keys used in the two UTXOs we will be spending. We need the private keys so we can sign each of the inputs separately.

⚠️ **Warning:** Users should never manually manage private keys on mainnet. As dangerous as raw transactions are (see warnings above), making a mistake with a private key can be much worse—as in the case of a HD wallet cross-generational key compromise. These examples are to help you learn, not for you to emulate on mainnet.

```
> bitcoin-cli -regtest getnewaddress
n4puhBEeEWD2VvjdRC9kQuX2abKxSCMNqN
> bitcoin-cli -regtest getnewaddress
n4LWXU59yM5MzQev7Jx7VNeq1BqZ85ZbLj

> NEW_ADDRESS1=n4puhBEeEWD2VvjdRC9kQuX2abKxSCMNqN
> NEW_ADDRESS2=n4LWXU59yM5MzQev7Jx7VNeq1BqZ85ZbLj
```

For our two outputs, get two new addresses.

```
## Outputs - inputs = transaction fee, so always double-check your math!
> bitcoin-cli -regtest createrawtransaction '''
    [
      {
        "txid": "'$UTXO1_TXID'",
        "vout": '$UTXO1_VOUT'
      },
      {
        "txid": "'$UTXO2_TXID'",
        "vout": '$UTXO2_VOUT'
      }
    ]
    ''' '''
    {
      "'$NEW_ADDRESS1'": 79.9999,
      "'$NEW_ADDRESS2'": 10
    }'''
0100000002f327e86da3e66bd20e1129b1fb36d07056f0b9a117199e75939652\
6b8f3a20780000000000ffffffff0ede03d75050f20801d50358829ae02c058\
e8677d2cc74df51f738285013c260000000000ffffffff02f028d6dc01000000\
1976a914ffb035781c3c69e076d48b60c3d38592e7ce06a788ac00ca9a3b0000\
00001976a914fa5139067622fd7e1e722a05c17c2bb7d5fd6df088ac00000000

> RAW_TX=0100000002f327e86da3e66bd20e1129b1fb36d07056f0b9a117199[...]
```

Create the raw transaction using "createrawtransaction" much the same as before, except now we have two inputs and two outputs.

```
> bitcoin-cli -regtest signrawtransaction $RAW_TX '[]' '''
    [
      "'$UTXO1_PRIVATE_KEY'"
    ]'''
```

```
{
    "hex" : "0100000002f327e86da3e66bd20e1129b1fb36d07056f0b9a11\
             7199e759396526b8f3a207800000000494830450221 00fce442\
             ec52aa2792efc27fd3ad0eaf7fa69f097fdcefab017ea56d179\
             9b10b2102207a6ae3eb61e11ffaba0453f173d1792f1b7bb8e7\
             422ea945101d68535c4b474801fffffffff0ede03d75050f208\
             01d50358829ae02c058e8677d2cc74df51f738285013c260000\
             000000ffffffff02f028d6dc010000001976a914ffb035781c3\
             c69e076d48b60c3d38592e7ce06a788ac00ca9a3b0000000019\
             76a914fa5139067622fd7e1e722a05c17c2bb7d5fd6df088ac0\
             0000000",
    "complete" : false
    "errors": [
    {
      "txid": "c53f8f5ac0b6b10cdc77f543718eb3880fee6cf9b5e0cbf4edb2a59c0fae09a4",
      "vout": 0,
      "scriptSig": "",
      "sequence": 4294967295,
      "error": "Operation not valid with the current stack size"
    }
  ]
}
```

```
> PARTLY_SIGNED_RAW_TX=0100000002f327e86da3e66bd20e1129b1fb36d07[...]
```

Signing the raw transaction with `signrawtransaction` gets more complicated as we now have three arguments:

1. The unsigned raw transaction.
2. An empty array. We don't do anything with this argument in this operation, but some valid JSON must be provided to get access to the later positional arguments.
3. The private key we want to use to sign one of the inputs.

The result is a raw transaction with only one input signed; the fact that the transaction isn't fully signed is indicated by value of the `complete` JSON field. We save the incomplete, partly-signed raw transaction hex to a shell variable.

```
> bitcoin-cli -regtest signrawtransaction $PARTLY_SIGNED_RAW_TX '[]' '''
    [
      "'$UTXO2_PRIVATE_KEY'"
    ]'''
```

```
{
    "hex" : "0100000002f327e86da3e66bd20e1129b1fb36d07056f0b9a11\
             7199e759396526b8f3a207800000000494830450221 00fce442\
             ec52aa2792efc27fd3ad0eaf7fa69f097fdcefab017ea56d179\
             9b10b2102207a6ae3eb61e11ffaba0453f173d1792f1b7bb8e7\
             422ea945101d68535c4b474801fffffffff0ede03d75050f208\
             01d50358829ae02c058e8677d2cc74df51f738285013c260000\
             00006b483045022100b77f935ff366a6f3c2fdeb83589c79026\
             5d43b3d2cf5e5f0047da56c36de75f40220707ceda75d8dcf2c\
             caebc506f7293c3dcb910554560763d7659fb202f8ec324b012\
             102240d7d3c7aad57b68aa0178f4c56f997d1bfab2ded3c2f94\
             27686017c603a6d6fffffffff02f028d6dc010000001976a914f\
             fb035781c3c69e076d48b60c3d38592e7ce06a788ac00ca9a3b\
             000000001976a914fa5139067622fd7e1e722a05c17c2bb7d5f\
             d6df088ac00000000",
    "complete" : true
}
```

To sign the second input, we repeat the process we used to sign the first input using the second private key. Now that both inputs are signed, the `complete` result is *true*.

```
> unset PARTLY_SIGNED_RAW_TX RAW_TX NEW_ADDRESS1 [...]
```

Clean up the shell variables used. Unlike previous subsections, we're not going to send this transaction to the connected node with "sendrawtransaction". This will allow us to illustrate in the Offline Signing subsection below how to spend a transaction which is not yet in the block chain or memory pool.

Offline Signing

We will now spend the transaction created in the Complex Raw Transaction subsection above without sending it to the local node first. This is the same basic process used by wallet programs for offline signing—which generally means signing a transaction without access to the current UTXO set.

Offline signing is safe. However, in this example we will also be spending an output which is not part of the block chain because the transaction containing it has never been broadcast. That can be unsafe:

⚠️ **Warning:** Transactions which spend outputs from unconfirmed transactions are vulnerable to transaction malleability. Be sure to read about transaction malleability and adopt good practices before spending unconfirmed transactions on mainnet.

```
> OLD_SIGNED_RAW_TX=0100000002f327e86da3e66bd20e1129b1fb36d07056\
      f0b9a117199e759396526b8f3a207800000000494830450221 00fce442\
      ec52aa2792efc27fd3ad0eaf7fa69f097fdcefab017ea56d1799b10b21\
      02207a6ae3eb61e11ffaba0453f173d1792f1b7bb8e7422ea945101d68\
      535c4b474801fffffffff0ede03d75050f20801d50358829ae02c058e8\
      677d2cc74df51f738285013c26000000006b483045022100b77f935ff3\
      66a6f3c2fdeb83589c790265d43b3d2cf5e5f0047da56c36de75f40220\
      707ceda75d8dcf2ccaebc506f7293c3dcb910554560763d7659fb202f8\
      ec324b012102240d7d3c7aad57b68aa0178f4c56f997d1bfab2ded3c2f\
      9427686017c603a6d6ffffffff02f028d6dc010000001976a914ffb035\
      781c3c69e076d48b60c3d38592e7ce06a788ac00ca9a3b000000001976\
      a914fa5139067622fd7e1e722a05c17c2bb7d5fd6df088ac00000000
```

Put the previously signed (but not sent) transaction into a shell variable.

```
> bitcoin-cli -regtest decoderawtransaction $OLD_SIGNED_RAW_TX
```

```
{
    "txid" : "682cad881df69cb9df8f0c996ce96ecad758357ded2da03bad\
              40cf18ffbb8e09",
    "hash" : "682cad881df69cb9df8f0c996ce96ecad758357ded2da03bad40cf18ffbb8e09",
    "size" : 340,
    "vsize" : 340,
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "txid" : "78203a8f6b529693759e1917a1b9f05670d036fbb1\
                      29110ed26be6a36de827f3",
            "vout" : 0,
            "scriptSig" : {
                "asm" : "3045022100fce442ec52aa2792efc27fd3ad0ea\
                         f7fa69f097fdcefab017ea56d1799b10b210220\
                         7a6ae3eb61e11ffaba0453f173d1792f1b7bb8e\
                         7422ea945101d68535c4b474801",
                "hex" : "483045022100FCE442ec52aa2792efc27fd3ad0\
                         eaf7fa69f097fdcefab017ea56d1799b10b2102\
                         207a6ae3eb61e11ffaba0453f173d1792f1b7bb\
                         8e7422ea945101d68535c4b474801"
            },
            "sequence" : 4294967295
        },
        {
            "txid" : "263c018582731ff54dc72c7d67e858c002ae298835\
                      501d80200f05753de0edf0",
            "vout" : 0,
            "scriptSig" : {
                "asm" : "3045022100b77f935ff366a6f3c2fdeb83589c7\
                         90265d43b3d2cf5e5f0047da56c36de75f40220\
                         707ceda75d8dcf2ccaebc506f7293c3dcb91055\
                         4560763d7659fb202f8ec324b01
                         02240d7d3c7aad57b68aa0178f4c56f997d1bfa\
                         b2ded3c2f9427686017c603a6d6",
                "hex" : "483045022100b77f935ff366a6f3c2fdeb83589\
                         c790265d43b3d2cf5e5f0047da56c36de75f402\
                         20707ceda75d8dcf2ccaebc506f7293c3dcb910\
                         554560763d7659fb202f8ec324b012102240d7d\
                         3c7aad57b68aa0178f4c56f997d1bfab2ded3c2\
                         f9427686017c603a6d6"
            },
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 79.99990000,
            "n" : 0,
            "scriptPubKey" : {
                "asm" : "OP_DUP OP_HASH160 ffb035781c3c69e076d48\
                         b60c3d38592e7ce06a7 OP_EQUALVERIFY OP_CHECKSIG",
                "hex" : "76a914ffb035781c3c69e076d48b60c3d38592e\
                         7ce06a788ac",
                "reqSigs" : 1,
                "type" : "pubkeyhash",
                "addresses" : [
                    "n4puhBEeEWD2VvjdRC9kQuX2abKxSCMNqN"
                ]
            }
        },
        {
            "value" : 10.00000000,
            "n" : 1,
            "scriptPubKey" : {
                "asm" : "OP_DUP OP_HASH160 fa5139067622fd7e1e722\
                         a05c17c2bb7d5fd6df0 OP_EQUALVERIFY OP_CHECKSIG",
                "hex" : "76a914fa5139067622fd7e1e722a05c17c2bb7d\
                         5fd6df088ac",
                "reqSigs" : 1,
```

```
                "type" : "pubkeyhash",
                "addresses" : [
                    "n4LWXU59yM5MzQev7Jx7VNeq1BqZ85ZbLj"
                ]
            }
        }
    ]
}
```

```
> UTXO_TXID=682cad881df69cb9df8f0c996ce96ecad758357ded2da03bad40[...]
> UTXO_VOUT=1
> UTXO_VALUE=10.00000000
> UTXO_OUTPUT_SCRIPT=76a914fa5139067622fd7e1e722a05c17c2bb7d5fd6[...]
```

Decode the signed raw transaction so we can get its txid. Also, choose a specific one of its UTXOs to spend and save that UTXO's output index number (vout) and hex pubkey script (scriptPubKey) into shell variables.

```
> bitcoin-cli -regtest getnewaddress
mfdCHEFL2tW9eEUpizk7XLZJcnFM4hrp78

> NEW_ADDRESS=mfdCHEFL2tW9eEUpizk7XLZJcnFM4hrp78
```

Get a new address to spend the satoshis to.

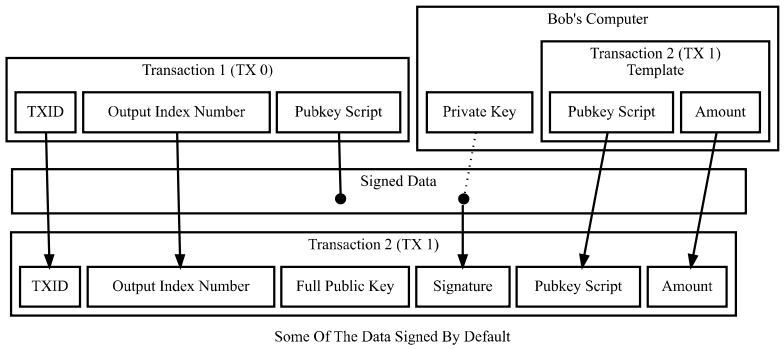
```
## Outputs - inputs = transaction fee, so always double-check your math!
> bitcoin-cli -regtest createrawtransaction '''
    [
      {
        "txid": "'$UTXO_TXID'",
        "vout": '$UTXO_VOUT'
      }
    ]
    ''' '''
    {
      "'$NEW_ADDRESS'": 9.9999
    }'''
0100000001098ebbff18cf40ad3ba02ded7d3558d7ca6ee96c990c8fdfb99cf6\
1d88ad2c680100000000ffffffff01f0a29a3b000000001976a914012e2ba6a0\
51c033b03d712ca2ea00a35eac1e7988ac00000000

> RAW_TX=0100000001098ebbff18cf40ad3ba02ded7d3558d7ca6ee96c990c8[...]
```

Create the raw transaction the same way we've done in the previous subsections.

```
> bitcoin-cli -regtest signrawtransaction $RAW_TX
```

```
{
    "hex" : "0100000001098ebbff18cf40ad3ba02ded7d3558d7ca6ee\
             96c990c8fdfb99cf61d88ad2c680100000000ffffffff01\
             f0a29a3b000000001976a914012e2ba6a051c033b03d712\
             ca2ea00a35eac1e7988ac00000000",
    "complete" : false
}
```

Attempt to sign the raw transaction without any special arguments, the way we successfully signed the the raw transaction in the Simple Raw Transaction subsection. If you've read the Transaction section of the guide, you may know why the call fails and leaves the raw transaction hex unchanged.



Some Of The Data Signed By Default

As illustrated above, the data that gets signed includes the txid and vout from the previous transaction. That information is included in the "createrawtransaction" raw transaction. But the data that gets signed also includes the pubkey script from the previous transaction, even though it doesn't appear in either the unsigned or signed transaction.

In the other raw transaction subsections above, the previous output was part of the UTXO set known to the wallet, so the wallet was able to use the txid and output index number to find the previous pubkey script and insert it automatically.

In this case, you're spending an output which is unknown to the wallet, so it can't automatically insert the previous pubkey script.

```
> bitcoin-cli -regtest signrawtransaction $RAW_TX '''
    [
      {
        "txid": "'$UTXO_TXID'",
        "vout": '$UTXO_VOUT',
        "scriptPubKey": "'$UTXO_OUTPUT_SCRIPT'",
        "value": '$UTXO_VALUE'
      }
    ]'''
```

```
{
    "hex" : "0100000001098ebbff18cf40ad3ba02ded7d3558d7ca6ee96c9\
             90c8fdfb99cf61d88ad2c68010000006b483045022100c3f92f\
             b74bfa687d76ebe75a654510bb291b8aab6f89ded4fe26777c2\
             eb233ad02207f779ce2a181cc4055cb0362aba7fd7a6f72d5db\
             b9bd863f4faaf47d8d6c4b500121028e4e62d25760709806131\
             b014e2572f7590e70be01f0ef16bfbd51ea5f389d4dffffffff\
             01f0a29a3b000000001976a914012e2ba6a051c033b03d712ca\
             2ea00a35eac1e7988ac00000000",
    "complete" : true
}
```

```
> SIGNED_RAW_TX=0100000001098ebbff18cf40ad3ba02ded7d3558d7ca6ee9[...]
```

Successfully sign the transaction by providing the previous pubkey script and other required input data.

This specific operation is typically what offline signing wallets do. The online wallet creates the raw transaction and gets the previous pubkey scripts for all the inputs. The user brings this information to the offline wallet. After displaying the transaction details to the user, the offline wallet signs the transaction as we did above. The user takes the signed transaction back to the online wallet, which broadcasts it.

```
> bitcoin-cli -regtest sendrawtransaction $SIGNED_RAW_TX
```

```
{"error": {"code":-22,"message":"TX rejected"}}
```

Attempt to broadcast the second transaction before we've broadcast the first transaction. The node rejects this attempt because the second transaction spends an output which is not a UTXO the node knows about.

```
> bitcoin-cli -regtest sendrawtransaction $OLD_SIGNED_RAW_TX
682cad881df69cb9df8f0c996ce96ecad758357ded2da03bad40cf18ffbb8e09
> bitcoin-cli -regtest sendrawtransaction $SIGNED_RAW_TX
67d53afa1a8167ca093d30be7fb9dcb8a64a5fdecacec9d93396330c47052c57
```

Broadcast the first transaction, which succeeds, and then broadcast the second transaction—which also now succeeds because the node now sees the UTXO.

```
> bitcoin-cli -regtest getrawmempool
```

```
[
    "67d53afa1a8167ca093d30be7fb9dcb8a64a5fdecacec9d93396330c47052c57",
    "682cad881df69cb9df8f0c996ce96ecad758357ded2da03bad40cf18ffbb8e09"
]
```

We have once again not generated an additional block, so the transactions above have not yet become part of the regtest block chain. However, they are part of the local node's memory pool.

```
> unset OLD_SIGNED_RAW_TX SIGNED_RAW_TX RAW_TX [...]
```

Remove old shell variables.

P2SH Multisig

In this subsection, we will create a P2SH multisig address, spend satoshis to it, and then spend those satoshis from it to another address.

Creating a multisig address is easy. Multisig outputs have two parameters, the *minimum* number of signatures required (*m*) and the *number* of public keys to use to validate those signatures. This is called m-of-n, and in this case we'll be using 2-of-3.

```
> bitcoin-cli -regtest getnewaddress
mhAXF4Eq7iRyvbYk1mpDVBiGdLP3YbY6Dm
> bitcoin-cli -regtest getnewaddress
moaCrnRfP5zzyhW8k65f6Rf2z5QpvJzSKe
> bitcoin-cli -regtest getnewaddress
mk2QpYatsKicvFVuTAQLBryyccRXMUaGHP

> NEW_ADDRESS1=mhAXF4Eq7iRyvbYk1mpDVBiGdLP3YbY6Dm
> NEW_ADDRESS2=moaCrnRfP5zzyhW8k65f6Rf2z5QpvJzSKe
> NEW_ADDRESS3=mk2QpYatsKicvFVuTAQLBryyccRXMUaGHP
```

Generate three new P2PKH addresses. P2PKH addresses cannot be used with the multisig redeem script created below. (Hashing each public key is unnecessary anyway—all the public keys are protected by a hash when the redeem script is hashed.) However, Bitcoin Core uses addresses as a way to reference the underlying full (unhashed) public keys it knows about, so we get the three new addresses above in order to use their public keys.

Recall from the Guide that the hashed public keys used in addresses obfuscate the full public key, so you cannot give an address to another person or device as part of creating a typical multisig output or P2SH multisig redeem script. You must give them a full public key.

```
> bitcoin-cli -regtest validateaddress $NEW_ADDRESS3
```

```
{
    "isvalid" : true,
    "address" : "mk2QpYatsKicvFVuTAQLBryyccRXMUaGHP",
    "scriptPubKey" : "76a9143172b5654f6683c8fb146959d347ce303cae4ca788ac",
    "ismine" : true,
    "iswatchonly" : false,
    "isscript" : false,
    "pubkey" : "029e03a901b85534ff1e92c43c74431f7ce72046060fcf7a\
                95c37e148f78c77255",
    "iscompressed" : true,
    "account" : ""
}
```

```
> NEW_ADDRESS3_PUBLIC_KEY=029e03a901b85534ff1e92c43c74431f7ce720[...]
```

Use the "validateaddress" RPC to display the full (unhashed) public key for one of the addresses. This is the information which will actually be included in the multisig redeem script. This is also the information you would give another person or device as part of creating a multisig output or P2SH multisig redeem script.

We save the address returned to a shell variable.

```
> bitcoin-cli -regtest createmultisig 2 '''
    [
      "'$NEW_ADDRESS1'",
      "'$NEW_ADDRESS2'",
      "'$NEW_ADDRESS3_PUBLIC_KEY'"
    ]'''
```

```
{
    "address" : "2N7NaqSKYQUeM8VNgBy8D9xQQbiA8yiJayk",
    "redeemScript" : "522103310188e911026cf18c3ce274e0ebb5f95b00\
    7f230d8cb7d09879d96dbeab1aff210243930746e6ed6552e03359db521b\
    088134652905bd2d1541fa9124303a41e95621029e03a901b85534ff1e92\
    c43c74431f7ce72046060fcf7a95c37e148f78c7725553ae"
}
```

```
> P2SH_ADDRESS=2N7NaqSKYQUeM8VNgBy8D9xQQbiA8yiJayk
> P2SH_REDEEM_SCRIPT=522103310188e911026cf18c3ce274e0ebb5f95b007[...]
```

Use the "createmultisig" RPC with two arguments, the number (*n*) of signatures required and a list of addresses or public keys. Because P2PKH addresses can't be used in the multisig redeem script created by this RPC, the only addresses which can be provided are those belonging to a public key in the wallet. In this case, we provide two addresses and one public key—all of which will be converted to public

keys in the redeem script.

The P2SH address is returned along with the redeem script which must be provided when we spend satoshis sent to the P2SH address.

⚠ **Warning:** You must not lose the redeem script, especially if you don't have a record of which public keys you used to create the P2SH multisig address. You need the redeem script to spend any bitcoins sent to the P2SH address. If you lose the redeem script, you can recreate it by running the same command above, with the public keys listed in the same order. However, if you lose both the redeem script and even one of the public keys, you will never be able to spend satoshis sent to that P2SH address.

Neither the address nor the redeem script are stored in the wallet when you use "createmultisig". To store them in the wallet, use the "addmultisigaddress" RPC instead. If you add an address to the wallet, you should also make a new backup.

```
> bitcoin-cli -regtest sendtoaddress $P2SH_ADDRESS 10.00
7278d7d030f042ebe633732b512bcb31fff14a697675a1fe1884db139876e175

> UTXO_TXID=7278d7d030f042ebe633732b512bcb31fff14a697675a1fe1884[...]
```

Paying the P2SH multisig address with Bitcoin Core is as simple as paying a more common P2PKH address. Here we use the same command (but different variable) we used in the Simple Spending subsection. As before, this command automatically selects an UTXO, creates a change output to a new one of our P2PKH addresses if necessary, and pays a transaction fee if necessary.

We save that txid to a shell variable as the txid of the UTXO we plan to spend next.

```
> bitcoin-cli -regtest getrawtransaction $UTXO_TXID 1
```

```
{
    "hex" : "0100000001f0ede03d75050f20801d50358829ae02c058e8677\
             d2cc74df51f738285013c26010000006a47304402203c375959\
             2bf608ab79c01596c4a417f3110dd6eb776270337e575cdafc6\
             99af20220317ef140d596cc255a4067df8125db7f349ad94521\
             2e9264a87fa8d777151937012102a92913b70f9fb15a7ea5c42\
             df44637f0de26e2dad97d6d54957690b94cf2cd05ffffffff01\
             00ca9a3b0000000017a9149af61346ce0aa2dffcf697352b4b7\
             04c84dcbaff8700000000",
    "txid" : "7278d7d030f042ebe633732b512bcb31fff14a697675a1fe18\
              84db139876e175",
    "hash" : "7278d7d030f042ebe633732b512bcb31fff14a697675a1fe1884db139876e175",
    "size" : 189,
    "vsize" : 189,
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "txid" : "263c018582731ff54dc72c7d67e858c002ae298835\
                      501d80200f05753de0edf0",
            "vout" : 1,
            "scriptSig" : {
                "asm" : "304402203c3759592bf608ab79c01596c4a417f\
                         3110dd6eb776270337e575cdafc699af2022031\
                         7ef140d596cc255a4067df8125db7f349ad9452\
                         12e9264a87fa8d77715193701
                         02a92913b70f9fb15a7ea5c42df44637f0de26e\
                         2dad97d6d54957690b94cf2cd05",
                "hex" : "47304402203c3759592bf608ab79c01596c4a41\
                         7f3110dd6eb776270337e575cdafc699af20220\
                         317ef140d596cc255a4067df8125db7f349ad94\
                         5212e9264a87fa8d777151937012102a92913b7\
                         0f9fb15a7ea5c42df44637f0de26e2dad97d6d5\
                         4957690b94cf2cd05"
            },
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 10.00000000,
            "n" : 0,
            "scriptPubKey" : {
                "asm" : "OP_HASH160 9af61346ce0aa2dffcf697352b4b\
                704c84dcbaff OP_EQUAL",
                "hex" : "a9149af61346ce0aa2dffcf697352b4b704c84d\
                         cbaff87",
                "reqSigs" : 1,
                "type" : "scripthash",
                "addresses" : [
                    "2N7NaqSKYQUeM8VNgBy8D9xQQbiA8yiJayk"
                ]
            }
        }
    ]
}
```


```
> UTXO_VOUT=0
> UTXO_OUTPUT_SCRIPT=a9149af61346ce0aa2dffcf697352b4b704c84dcbaff87
```

We use the "getrawtransaction" RPC with the optional second argument (*true*) to get the decoded transaction we just created with "sendtoaddress". We choose one of the outputs to be our UTXO and get its output index number (vout) and pubkey script (scriptPubKey).

```
> bitcoin-cli -regtest getnewaddress
mxCNLtKxzgjg8yyNHeuFSXvxCvagkWdfGU

> NEW_ADDRESS4=mxCNLtKxzgjg8yyNHeuFSXvxCvagkWdfGU
```

We generate a new P2PKH address to use in the output we're about to create.

```
## Outputs - inputs = transaction fee, so always double-check your math!
> bitcoin-cli -regtest createrawtransaction '''
   [
     {
       "txid": "'$UTXO_TXID'",
       "vout": '$UTXO_VOUT'
     }
   ]
   ''' '''
   {
     "'$NEW_ADDRESS4'": 9.998
   }'''

010000000175e1769813db8418fea17576694af1ff31cb2b512b7333e6eb42f0\
30d0d778720000000000ffffffff01c0bc973b000000001976a914b6f64f5bf3\
e38f25ead28817df7929c06fe847ee88ac00000000

> RAW_TX=010000000175e1769813db8418fea17576694af1ff31cb2b512b733[...]
```

We generate the raw transaction the same way we did in the Simple Raw Transaction subsection.

```
> bitcoin-cli -regtest dumpprivkey $NEW_ADDRESS1
cVinshabsALz5Wg4tGDiBuqEGq4i6WCKWXRQdM8RFxLbALvNSHw7
> bitcoin-cli -regtest dumpprivkey $NEW_ADDRESS3
cNmbnwwGzEghMMe1vBwH34DFHShEj5bcXD1QpFRPHgG9Mj1xc5hq

> NEW_ADDRESS1_PRIVATE_KEY=cVinshabsALz5Wg4tGDiBuqEGq4i6WCKWXRQd[...]
> NEW_ADDRESS3_PRIVATE_KEY=cNmbnwwGzEghMMe1vBwH34DFHShEj5bcXD1Qp[...]
```

We get the private keys for two of the public keys we used to create the transaction, the same way we got private keys in the Complex Raw Transaction subsection. Recall that we created a 2-of-3 multisig pubkey script, so signatures from two private keys are needed.

⚠️ **Reminder:** Users should never manually manage private keys on mainnet. See the warning in the complex raw transaction section.

```
> bitcoin-cli -regtest signrawtransaction $RAW_TX '''
   [
     {
       "txid": "'$UTXO_TXID'",
       "vout": '$UTXO_VOUT',
       "scriptPubKey": "'$UTXO_OUTPUT_SCRIPT'",
       "redeemScript": "'$P2SH_REDEEM_SCRIPT'"
     }
   ]
   ''' '''
   [
     "'$NEW_ADDRESS1_PRIVATE_KEY'"
   ]'''
```

```
{
    "hex" : "010000000175e1769813db8418fea17576694af1ff31cb2b512\
             b7333e6eb42f030d0d7787200000000b5004830450221008d5e\
             c57d362ff6ef6602e4e756ef1bdeee12bd5c5c72697ef1455b3\
             79c90531002202ef3ea04dfbeda043395e5bc701e4878c15baa\
             b9c6ba5808eb3d04c91f641a0c014c69522103310188e911026\
             cf18c3ce274e0ebb5f95b007f230d8cb7d09879d96dbeab1aff\
             210243930746e6ed6552e03359db521b088134652905bd2d154\
             1fa9124303a41e95621029e03a901b85534ff1e92c43c74431f\
             7ce72046060fcf7a95c37e148f78c7725553aeffffffff01c0b\
             c973b0000000001976a914b6f64f5bf3e38f25ead28817df7929\
             c06fe847ee88ac00000000",
    "complete" : false
}
```

```
> PARTLY_SIGNED_RAW_TX=010000000175e1769813db8418fea17576694af1f[...]
```

We make the first signature. The input argument (JSON object) takes the additional redeem script parameter so that it can append the redeem script to the signature script after the two signatures.

```
> bitcoin-cli -regtest signrawtransaction $PARTLY_SIGNED_RAW_TX '''
    [
      {
        "txid": "'$UTXO_TXID'",
        "vout": '$UTXO_VOUT',
        "scriptPubKey": "'$UTXO_OUTPUT_SCRIPT'",
        "redeemScript": "'$P2SH_REDEEM_SCRIPT'"
      }
    ]
    ''' '''
    [
      "'$NEW_ADDRESS3_PRIVATE_KEY'"
    ]'''
```


```
{
    "hex" : "010000000175e1769813db8418fea17576694af1ff31cb2b512\
             b7333e6eb42f030d0d7787200000000fdfd0000483045022100\
             8d5ec57d362ff6ef6602e4e756ef1bdeee12bd5c5c72697ef14\
             55b379c90531002202ef3ea04dfbeda043395e5bc701e4878c1\
             5baab9c6ba5808eb3d04c91f641a0c0147304402200bd8c62b9\
             38e02094021e481b149fd5e366a212cb823187149799a68cfa7\
             652002203b52120c5cf25ceab5f0a6b5cdb8eca0fd2f386316c\
             9721177b75ddca82a4ae8014c69522103310188e911026cf18c\
             3ce274e0ebb5f95b007f230d8cb7d09879d96dbeab1aff21024\
             3930746e6ed6552e03359db521b088134652905bd2d1541fa91\
             24303a41e95621029e03a901b85534ff1e92c43c74431f7ce72\
             046060fcf7a95c37e148f78c7725553aefffffff01c0bc973b\
             000000001976a914b6f64f5bf3e38f25ead28817df7929c06fe\
             847ee88ac00000000",
    "complete" : true
}
```


```
> SIGNED_RAW_TX=010000000175e1769813db8418fea17576694af1ff31cb2b[...]
```

The `signrawtransaction` call used here is nearly identical to the one used above. The only difference is the private key used. Now that the two required signatures have been provided, the transaction is marked as complete.

```
> bitcoin-cli -regtest sendrawtransaction $SIGNED_RAW_TX
430a4cee3a55efb04cbb8718713cab18dea7f2521039aa660ffb5aae14ff3f50
```

We send the transaction spending the P2SH multisig output to the local node, which accepts it.

Support Bitcoin.org: Donate

3E8ociqZa9mZUSwGdSmAEMAoAxBK3FNDcd

Introduction:

- Individuals
- Businesses
- Developers
- Getting started
- How it works
- You need to know
- White paper

Resources:

- Resources
- Exchanges
- Community
- Vocabulary
- Events
- Bitcoin Core

Participate:

- Support Bitcoin
- Buy Bitcoin
- Running a full node
- Development

Other:

- Avoid Scams
- Legal
- Privacy Policy
- Press
- About bitcoin.org
- Blog