Air Gap	= Cold Environment (Offline)
Hot Node	= Synchronized Node (Online)

Cardano-cli:~$ Study sheets

Part 1: keys, address and delegation

This document aims to explain in detail how to interpret the cardano-cli commands and their options in order to be able to assemble them by yourself if necessary. To do this, you must have a computer and install a Node of the Cardano blockchain and the Cardano command line interphase (cardano-cli) on it. You will start with simple commands first and increase in complexity as the tutorial progresses.

First exercise: Creation of the payment and the stake keys — Air Gap

1 First, locate the branch that you are going to use for your payment keys.
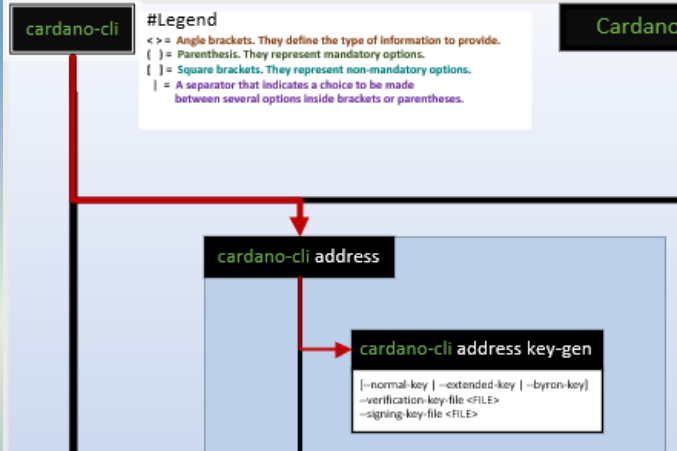
2 You have 5 options in total.

The first 3 options have square brackets with 2 separators which indicates that choosing between these 3 options is not mandatory since a normal key will be used by default if nothing is specified. For this example, you will not use them.

cardano-cli address key-gen
```
[--normal-key | --extended-key | --byron-key]
--verification-key-file <FILE>
--signing-key-file <FILE>
```

3 The following 2 options must be used.

The angle brackets indicating the type of information <File>. You must provide the name that you will give to your 2 respective key files.

cardano-cli address key-gen
```
[--normal-key | --extended-key | --byron-key]
--verification-key-file payment.vkey
--signing-key-file payment.skey
```

4 This is the final result of this simple command on your terminal.
```
user@computer:~$ cardano-cli address key-gen \
> --verification-key-file payment.vkey \
> --signing-key-file payment.skey
```

5 Now that your payment keys are ready, you have to create your stake keys. This one is even easier because there is only one type of stake key.

cardano-cli stake-address key-gen
```
--verification-key-file <FILE>
--signing-key-file <FILE>
```

6 Same as **3**. You have to indicate the type of information. <File>

cardano-cli stake-address key-gen
```
--verification-key-file stake.vkey
--signing-key-file stake.skey
```

**7** And for the final result on the terminal..
```
user@computer:~$ cardano-cli stake-address key-gen \
> --verification-key-file stake.vkey \
> --signing-key-file stake.skey
```

⚠ **Warning.** It is recommended to generate and use your payment and stake keys to sign transactions in an "Air Gap" environment for security reasons. *https://developers.cardano.org/docs/get-started/air-gap*

Now that your 2 key pairs are created, you will be able to create a stake address which will allow you to inquire about the amount of your rewards and allow you to withdraw them when used in a transaction with your stake.skey.

Second exercise: Creation of a stake address — Air Gap

1 Locate the branch that you are going to use to create your stake address.
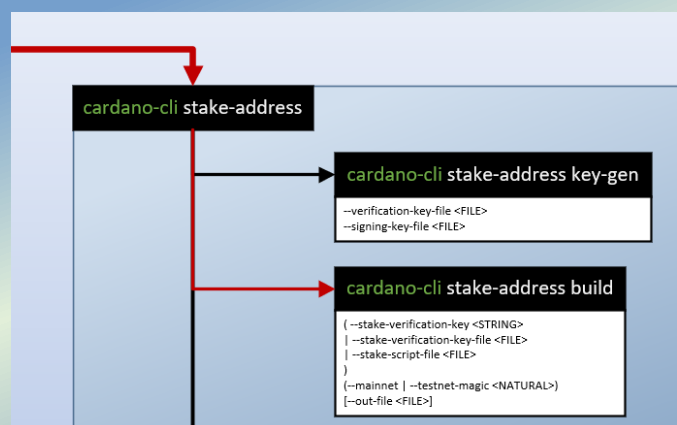
2 You have 6 options in total.

The first 3 options are joined by parentheses and have 2 separators which indicates a mandatory choice to be made between these three options.

cardano-cli stake-address build
```
( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
)
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 You will use the stake-verification-key-file option.

The angle brackets indicating the type of information <File>. This time, you must provide the path that leads to your stake.vkey

cardano-cli stake-address build
```
( --stake-verification-key <STRING>
| --stake-verification-key-file stake.vkey
| --stake-script-file <FILE>
)
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

4 Now you have 2 options in parenthesis.

You must specify the network used and if it is the testnet, mention the network magic number. Lets use the mainnet.

cardano-cli stake-address build
```
--stake-verification-key-file stake.vkey
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

5 And for the last option --out-file<FILE>

This one is not mandatory because it is inside square brackets. But if you don't use this option, the output(stake-address) of the command will be displayed in your terminal instead of a file, and since you will need to use this stake address later, lets give it a name.

cardano-cli stake-address build
```
--stake-verification-key-file stake.vkey
--mainnet
--out-file stake.addr
```

6 This is how this command will look on your terminal.
```
user@computer:~$ cardano-cli stake-address build \
> --stake-verification-key-file stake.vkey \
> --mainnet \
> --out-file stake.addr
```

7 Here is what you should have so far.
```
user@computer:~$ ls
payment.vkey   payment.skey   stake.vkey
stake.skey   stake.addr
```

*Now that your 2 key pairs and your stake address are created, you will be able to create an address combining your payment key with the stake key so that the money in the address generated will be included in the staking protocol with your rewards.*

**1** Locate the branch that you are going to use for your payment with stake address file.
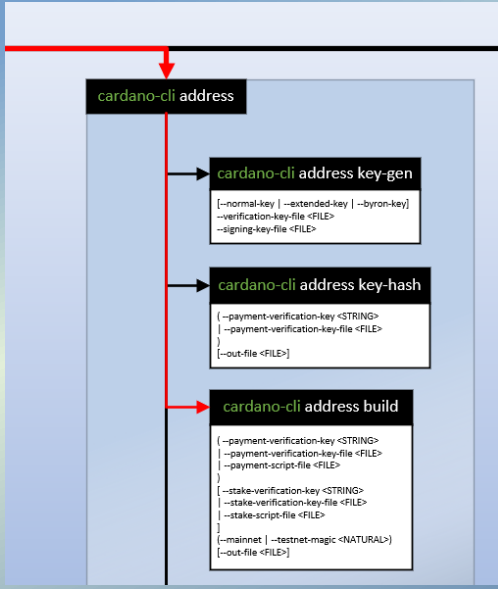


**2** You have 9 options in total.

It is possible for you to create a payment address using only your payment key without linking it to your stake key, which explains why the 1st group of options is mandatory and not the 2nd group.

```
cardano-cli address build

( | --payment-verification-key <STRING>
  | --payment-verification-key-file <FILE>
  | --payment-script-file <FILE>
)
[ | --stake-verification-key <STRING>
  | --stake-verification-key-file <FILE>
  | --stake-script-file <FILE>
]
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 Lets use both payment and stake verification keys

Again, according to the angle brackets <FILE>, you have to define the path to both your payment.vkey and your stake.vkey.

```
cardano-cli address build

( --payment-verification-key <STRING>
  | --payment-verification-key-file payment.vkey
  | --payment-script-file <FILE>
)
( --stake-verification-key <STRING>
  | --stake-verification-key-file stake.vkey
  | --stake-script-file <FILE>
)
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

4 Now you will mention the network to use and the file name for your address.

```
cardano-cli address build

--payment-verification-key-file payment.vkey
--stake-verification-key-file stake.vkey
[--mainnet | --testnet-magic <NATURAL>)
--out-file paymentwithstake.addr
```

5 This is the final result.

```
user@computer:~$ cardano-cli address build \
> --payment-verification-key-file payment.vkey \
> --stake-verification-key-file stake.vkey \
> --mainnet \
> --out-file paymentwithstake.addr
```

You can copy the content of paymentwithstake.addr in a text editor and paste it in a transaction from the Cardano wallet that you usually use and send some ada to it. (10 ada should be enough to start)

1 Locate the branch that you are going to use for your stake certificate.
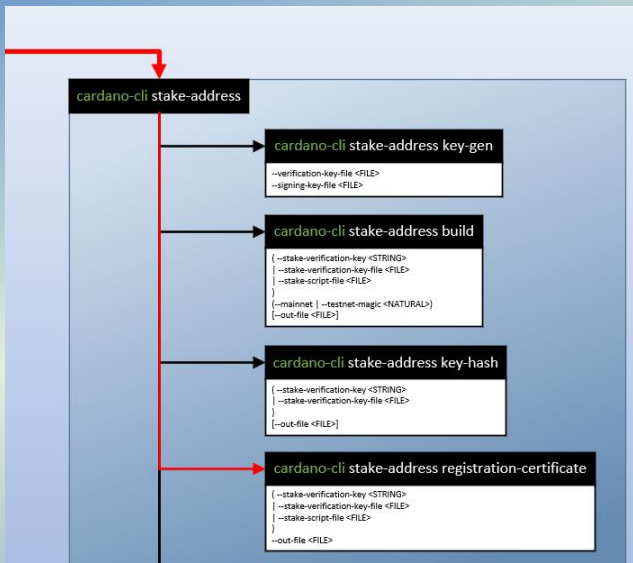
2 You have 4 options

To be able to participate in the protocol and stake your ada, you need to link your stake verification key to a certificate that you will submit to the blockchain in the next few exercises. The command to create your certificate is quite simple. You just need to provide one of these 3 mandatory options and specify the name of the file that will serve as your certificate.

```
cardano-cli stake-address registration-certificate

( --stake-verification-key <STRING>
| --stake-verification-key-file <FILE>
| --stake-script-file <FILE>
)
--out-file <FILE>
```

```
cardano-cli stake-address registration-certificate

( --stake-verification-key <STRING>
| --stake-verification-key-file stake.vrf
| --stake-script-file <FILE>
)
--out-file stake.cert
```

3 This is the final result of how this command should look like on your terminal.

```
user@computer:~$ cardano-cli stake-address registration-certificate \
> --stake-verification-key-file stake.vkey \
> --out-file stake.cert
```

4 Here's what you should have so far

```
user@computer:~$ ls
payment.vkey      payment.skey            stake.vkey      stake.skey
stake.addr        paymentwithstake.addr   stake.cert
```

You will now get the protocol parameters and the blockchain tip so that you can start building your very first transaction.

1 First, for your transaction, you will need protocol parameters for fee calculation.
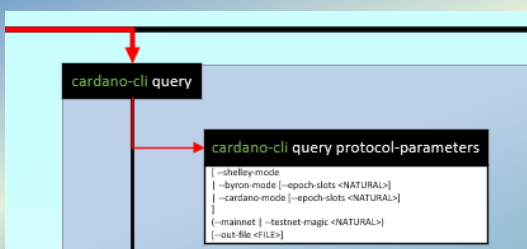
2 You have 6 options and 2 sub options in total.

```
cardano-cli query protocol-parameters

[ --shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--mainnet | --testnet-magic <NATURAL>)
[--out-file <FILE>]
```

3 Skip the mode options. Mention the desired network and the name of the file to be created.

```
cardano-cli query protocol-parameters

[ --shelley-mode
| --byron-mode [--epoch-slots <NATURAL>]
| --cardano-mode [--epoch-slots <NATURAL>]
]
(--mainnet | --testnet-magic <NATURAL>)
--out-file protocol.json
```

4 This is the final result of how this command will look on your terminal.

```
user@computer:~$ cardano-cli query protocol-parameters \
> --mainnet \
> --out-file protocol.json
```

5 So now go ahead and lets see the content of that file:

```
user@computer:~$ cat protocol.json
```

In the protocol.json file you will look for the deposit to be made in the blockchain to register your stake address and participate in the staking protocol. This deposit can be recovered at any time if you deregister your address.

6 Take note of the deposit amount, as you will need it later. The amount is in Lovelace. (1 ada = 1,000,000 Lovelace)

```
    "poolRetireMaxEpoch": 18,
    "protocolVersion": {
        "major": 8,
        "minor": 0
    },
    "stakeAddressDeposit": 2000000,
    "stakePoolDeposit": 500000000,
    "stakePoolTargetNum": 500,
    "treasuryCut": 0.2,
    "txFeeFixed": 155381,
    "txFeePerByte": 44,
    "utxoCostPerByte": 4310,
    "utxoCostPerWord": null
```

7 Now you need to take your protocol.json file and transfer it to your "Air Gap" environment to be able to calculate the fees when you will build your transactions.

⚠️ With the CIP-1694 and the Voltaire era which is at our doorstep, it will be possible for the ada holders of the community with the help of the constitutional committee and the Dreps to modify the protocol parameters in a well-developed voting system. This is why it is important for you to be sure to have the most recent modifications of these protocols in your "Air Gap" environment as this could have a direct impact on the various parameters surrounding your transactions.

Sixth exercise: Gets the node's current tip   **Hot Node**

1 Next exercise you will need to know the node current tip to calculate our TTL. (description in the next exercise)

2 Just like the previous excercise, 6 options 2 sub options.

Now that you are beginning to fully understand the principle, you can skip a few steps. No need to create a file, you just need the slot number.

```
cardano-cli query tip

    shelley-mode
    | byron-mode [--epoch-slots <NATURAL>]
    | cardano-mode [--epoch-slots <NATURAL>]
    ]
    (--mainnet | --testnet-magic <NATURAL>)
    [--out-file <FILE>]
```

3 This is the final result on your terminal.

```
user@computer:~$ cardano-cli query tip \
> --mainnet
```

4 Note the slot number

```
{
    "block": 8749125,
    "epoch": 410,
    "era": "Babbage",
    "hash": "503e4af96abc18e1d4b5de08e0d35cb508e364...",
    "slot": 92027764,
    "syncProgress": "100.00"
}
```

Seventh exercise: Query the UTXO   **Hot Node**

1 You will now query the UTXOs of your paymentwithstake.addr. (If you sent ada to it)

2 This command has 9 options 2 sub options.

You need to consume at least one UTXO as input to your transaction. A transaction can contain several input and several output but in this case, you should have only one UTXO associated with your paymentwithstake.addr because you made only one deposit of 10 ada in this one. So lets only use what is mandatory. In short, your paymentwithstake.addr, the network to use and you will then create a file to carry this list of UTXO to your "air gap" environment.

```
cardano-cli query utxo

    [ shelley-mode
    | byron-mode [--epoch-slots <NATURAL>]
    | cardano-mode [--epoch-slots <NATURAL>]
    ]
    (--whole-utxo | --address <ADDRESS> | (--tx-in <TX-IN>))
    --mainnet | --testnet-magic <NATURAL>
    --out-file <FILE>
```

3 This is what it should look like on your terminal

```
cardano-cli query utxo

    shelley-mode
    byron-mode [--epoch-slots <NATURAL>]
    cardano-mode [--epoch-slots <NATURAL>]
    ]
    whole-utxo | --address paymentwithstake.addr | (--tx-in <TX-IN>))
    --mainnet   --testnet-magic <NATURAL>
    --out-file UTXO.addrs
```

```
user@computer:~$ cardano-cli query utxo \
> --address paymentwithstake.addr
> --mainnet
> --out-file utxo.addrs
```

4 The contents of the utxo.addr file should look like this. The utxo of your deposit of 10 ada = 10,000,000 Lovelace. You can now take this file and put it in your "Air Gap" environment. You will need it soon.

```
                            TxHash                                    TxIx        Amount
--------------------------------------------------------------------------------------------------------
1234a4d18e9dkhb34234kjbvdec3ad81e299c1a523443453561e61ce9bf8608e8c802df3b7f8c   0    10000000 lovelace + TxOutDatumNone
```

It is now time to build your first transaction which will be used to submit your stake certificate. Before you start, what is to come may seem daunting but as you go step by step you should be able to understand why and how you will reduce the next options down to 6 options in total for your transaction process. For security purposes, in this tutorial, you will use methods involving "cardano-cli transaction build-raw" command instead of the "cardano-cli transaction build" command because it can be built in an offline environment.

Eighth exercise: Creation of your first transaction draft | Air Gap

1 — Locate the "cardano-cli transaction build-raw" branch

2 — let's gradually start from the top down.

```
cardano-cli transaction build-raw

[ --byron-era
| --shelley-era
| --allegra-era
| --mary-era
| --alonzo-era
| --babbage-era
```

3 — The first 5 options are optional [square brackets] By not mentioning anything, it will be Mary era by default.

```
cardano-cli transaction build-raw

[ --byron-era
| --shelley-era
| --allegra-era
| --mary-era
| --alonzo-era
| --babbage-era
]
[--script-valid | --script-invalid]
```

4 — Your transaction does not involve a script, so you can skip the next 2 options.

```
cardano-cli transaction build-raw

[ --byron-era
| --shelley-era
| --allegra-era
| --mary-era
| --alonzo-era
| --babbage-era
]
[--script-valid | --script-invalid]
```

5 — Then for the next options and its 20 sub-options, an explanation is required.

Inside an option bracket there may be sub-options defined by columns. This is why there is a notion of priority and a particular order to be respected when constructing a transaction. In this case you know that there are 3 distinct columns defining the order with which the options must be entered if we want the transaction body to be produced correctly.

```
1  [--script-valid | --script-invalid]
   (--tx-in <TX-IN>
2    [ --spending-tx-in-reference <TX-IN>
     --spending-plutus-script-v2
3    ( --spending-reference-tx-in-datum-cbor-file
     | --spending-reference-tx-in-datum-file
```

6 — Take the time to carefully analyze the priority order of the "tx-in" option and its brackets.

```
[--script-valid | --script-invalid]
(--tx-in <TX-IN>
  [ --spending-tx-in-reference <TX-IN>
  --spending-plutus-script-v2
  ( --spending-reference-tx-in-datum-cbor-file <CBOR FILE>
  | --spending-reference-tx-in-datum-file <JSON FILE>
  | --spending-reference-tx-in-datum-value <JSON VALUE>
  | --spending-reference-tx-in-inline-datum-present
  )
  ( --spending-reference-tx-in-redeemer-cbor-file <CBOR FILE>
  | --spending-reference-tx-in-redeemer-file <JSON FILE>
  | --spending-reference-tx-in-redeemer-value <JSON VALUE>
  )
  --spending-reference-tx-in-execution-units (<INT, INT>)
  | --simple-script-tx-in-reference <TX-IN>
  | --tx-in-script-file <FILE>
  [
    ( --tx-in-datum-cbor-file <CBOR FILE>
    | --tx-in-datum-file <JSON FILE>
    | --tx-in-datum-value <JSON VALUE>
    | --tx-in-inline-datum-present
    )
    ( --tx-in-redeemer-cbor-file <CBOR FILE>
    | --tx-in-redeemer-file <JSON FILE>
    | --tx-in-redeemer-value <JSON VALUE>
    )
    --tx-in-execution-units (<INT, INT>)]]
])
[--read-only-tx-in-reference <TX-IN>]
[--tx-in-collateral <TX-IN>]
```

7 — "--tx-in" is mandatory but not its sub-options.

The following options are not mandatory and are used for plutus scripts. So don't need to use them for our simple transaction.

```
--tx-in <TX-IN>
[ --spending-tx-in-reference <TX-IN>
--spending-plutus-script-v2
( --spending-reference-tx-in-datum-cbor-file <CBOR FILE>
| --spending-reference-tx-in-datum-file <JSON FILE>
| --spending-reference-tx-in-datum-value <JSON VALUE>
| --spending-reference-tx-in-inline-datum-present
)
( --spending-reference-tx-in-redeemer-cbor-file <CBOR FILE>
| --spending-reference-tx-in-redeemer-file <JSON FILE>
| --spending-reference-tx-in-redeemer-value <JSON VALUE>
)
--spending-reference-tx-in-execution-units (<INT, INT>)
| --simple-script-tx-in-reference <TX-IN>
| --tx-in-script-file <FILE>
[
( --tx-in-datum-cbor-file <CBOR FILE>
| --tx-in-datum-file <JSON FILE>
| --tx-in-datum-value <JSON VALUE>
| --tx-in-inline-datum-present
)
( --tx-in-redeemer-cbor-file <CBOR FILE>
| --tx-in-redeemer-file <JSON FILE>
| --tx-in-redeemer-value <JSON VALUE>
)
--tx-in-execution-units (<INT, INT>)
]]
```

8 — For what follows:

You will also not need "read only reference input" or anything concerning the collateral since it is a simple transaction which will not include a plutus script.

```
[--read-only-tx-in-reference <TX-IN>]
[--tx-in-collateral <TX-IN>]
[--tx-out-return-collateral <ADDRESS VALUE>]
[--tx-total-collateral <INTEGER>]
[--required-signer <FILE> | --required-signer-hash <HASH>]
[--tx-out <ADDRESS VALUE>
  [ --tx-out-datum-hash <HASH>
  | --tx-out-datum-hash-cbor-file <CBOR FILE>
```

9 — About "--required-signer-hash <HASH>"

This option will not be useful at this time for your transaction which will be used to submit your stake.cert onchain but it should be noted that it will be very useful to you in the excercise about the governance vote.

```
[--read-only-tx-in-reference <TX-IN>]
[--tx-in-collateral <TX-IN>]
[--tx-out-return-collateral <ADDRESS VALUE>]
[--tx-total-collateral <INTEGER>]
[--required-signer <FILE> | --required-signer-hash <HASH>]
[--tx-out <ADDRESS VALUE>
```

10 — Finally! An option that you will need, "--tx-out"

You will need this option to specify the address that will receive the balance of your consumed UTXO minus the fees. So let's copy this option and add it to your transaction draft.

```
[--read-only-tx-in-reference <TX-IN>]
[--tx-in-collateral <TX-IN>]
[--tx-out-return-collateral <ADDRESS VALUE>]
[--tx-total-collateral <INTEGER>]
[--required-signer <FILE> | --required-signer-hash <HASH>]
--tx-out <ADDRESS VALUE>
```

11 — The sub-options of "tx-out" about plutus script can be skipped. You don't need it for now.

```
--tx-out <ADDRESS VALUE>
[ --tx-out-datum-hash <HASH>
| --tx-out-datum-hash-cbor-file <CBOR FILE>
| --tx-out-datum-hash-file <JSON FILE>
| --tx-out-datum-hash-value <JSON VALUE>
| --tx-out-datum-embed-cbor-file <CBOR FILE>
| --tx-out-datum-embed-file <JSON FILE>
| --tx-out-datum-embed-value <JSON VALUE>
| --tx-out-inline-datum-cbor-file <CBOR FILE>
| --tx-out-inline-datum-file <JSON FILE>
| --tx-out-inline-datum-value <JSON VALUE>
]
| --tx-out-reference-script-file <FILE>]]
[--mint <VALUE>
```

12 — You are gradually begining to understand. No multi-asset, no NFT, no plutus script for your transactions.

```
[--mint <VALUE>
( --mint-script-file <FILE>
  [
  ( --mint-redeemer-cbor-file <CBOR FILE>
  | --mint-redeemer-file <JSON FILE>
  | --mint-redeemer-value <JSON VALUE>
  )
  --mint-execution-units (<INT, INT>)]
| --simple-minting-script-tx-in-reference <TX-IN> --policy-id <HASH>
| --mint-tx-in-reference <TX-IN>
--mint-plutus-script-v2
( --mint-reference-tx-in-redeemer-cbor-file <CBOR FILE>
| --mint-reference-tx-in-redeemer-file <JSON FILE>
| --mint-reference-tx-in-redeemer-value <JSON VALUE>
)
--mint-reference-tx-in-execution-units (<INT, INT>)
--policy-id <HASH>
)]
```

13 — You will then use 3 of the next 4 options.

- "--invalid-before" determines from which Slot the transaction will be valid to be processed.
- while "--invalid-herafter" determines from which Slot the transaction will become invalid. (just like an expiration date)

```
[--invalid-before <SLOT>]
[--invalid-hereafter <SLOT>]
[--fee <LOVELACE>]
[--certificate-file <CERTIFICATEFILE>
```

14 — You get the TTL, the fee and certificate-file.

That's why you did a "cardano-cli query tip" some exercises earlier. By knowing the slot number of your synchronized node, you can determine a "time to live" or "TTL" for your transaction while its in the memory pool. So you can now add to your draft these three options that will be detailed later.

```
[--invalid-before <SLOT>]
--invalid-hereafter <SLOT>]
--fee <LOVELACE>]
--certificate-file <CERTIFICATEFILE>
```

15 — Again, no certificate options relative to the plutus script will be used.

```
[--certificate-file <CERTIFICATEFILE>
| --certificate-script-file <FILE>
[
( --certificate-redeemer-cbor-file <CBOR FILE>
| --certificate-redeemer-file <JSON FILE>
| --certificate-redeemer-value <JSON VALUE>
)
--certificate-execution-units (<INT, INT>)]
| --certificate-tx-in-reference <TX-IN>
--certificate-plutus-script-v2
( --certificate-reference-tx-in-redeemer-cbor-file <CBOR FILE>
| --certificate-reference-tx-in-redeemer-file <JSON FILE>
| --certificate-reference-tx-in-redeemer-value <JSON VALUE>
)
--certificate-reference-tx-in-execution-units (<INT, INT>)
]]
```

16 The "--withdrawal" option is an input that allows you to withdraw your rewards in your stake.addr.

```
--withdrawal <WITHDRAWAL>
[ --withdrawal-script-file <FILE>
  [
    ( --withdrawal-redeemer-cbor-file <CBOR FILE>
    | --withdrawal-redeemer-file <JSON FILE>
    | --withdrawal-redeemer-value <JSON VALUE>
    )
    --withdrawal-execution-units (<INT, INT>)]]
| --withdrawal-tx-in-reference <TX-IN>
  --withdrawal-plutus-script-v2
  ( --withdrawal-reference-tx-in-redeemer-cbor-file <CBOR FILE>
  | --withdrawal-reference-tx-in-redeemer-file <JSON FILE>
  | --withdrawal-reference-tx-in-redeemer-value <JSON VALUE>
  )
  --withdrawal-reference-tx-in-execution-units (<INT, INT>)
]]
[--json-metadata-no-schema | --json-metadata-detailed-schema]
[--auxiliary-script-file <FILE>]
[--metadata-json-file <FILE> | --metadata-cbor-file <FILE>]
[--genesis <FILE> | --protocol-params-file <FILE>]
[--update-proposal-file <FILE>]
--out-file <FILE>
```

17 You can skip it and those related to the plutus script for now.

```
[ --withdrawal <WITHDRAWAL>
[ --withdrawal-script-file <FILE>
  [
    ( --withdrawal-redeemer-cbor-file <CBOR FILE>
    | --withdrawal-redeemer-file <JSON FILE>
    | --withdrawal-redeemer-value <JSON VALUE>
    )
    --withdrawal-execution-units (<INT, INT>)]
| --withdrawal-tx-in-reference <TX-IN>
  --withdrawal-plutus-script-v2
  ( --withdrawal-reference-tx-in-redeemer-cbor-file <CBOR FILE>
  | --withdrawal-reference-tx-in-redeemer-file <JSON FILE>
  | --withdrawal-reference-tx-in-redeemer-value <JSON VALUE>
  )
  --withdrawal-reference-tx-in-execution-units (<INT, INT>)
]]
[--json-metadata-no-schema | --json-metadata-detailed-schema]
[--auxiliary-script-file <FILE>]
[--metadata-json-file <FILE> | --metadata-cbor-file <FILE>]
[--genesis <FILE> | --protocol-params-file <FILE>]
[--update-proposal-file <FILE>]
--out-file <FILE>
```

18 Only a few remaining options.

You don't have any metadata to submit at this time, nor an auxiliary script file, nor need to specify a genesis or protocol parameters file. And you do not submit an update proposal for the catalyst fund. What you are left with is simply naming the file for your draft transaction. (--out-file <FILE>)

```
[--json-metadata-no-schema | --json-metadata-detailed-schema]
[--auxiliary-script-file <FILE>]
[--metadata-json-file <FILE> | --metadata-cbor-file <FILE>]
[--genesis <FILE> | --protocol-params-file <FILE>]
[--update-proposal-file <FILE>]
--out-file <FILE>
```

19 By grouping the options that you copied during this exercise, you will obtain something like this:

```
cardano-cli transaction build-raw

--tx-in <TX-IN>
--tx-out <ADDRESS VALUE>
[--invalid-hereafter <SLOT>]
[--fee <LOVELACE>]
[--certificate-file <CERTIFICATEFILE>
--out-file <FILE>
```

20 Now to complete your draft:

You add the UTXO input (tx-in), the change address (tx-out) and the certificate file. For now you will give a value of 0 to tx-out, invalid-hereafter and fee.

```
cardano-cli transaction build-raw

--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+0
--invalid-hereafter 0
--fee 0
--certificate-file stake.cert
--out-file tx.raw
```
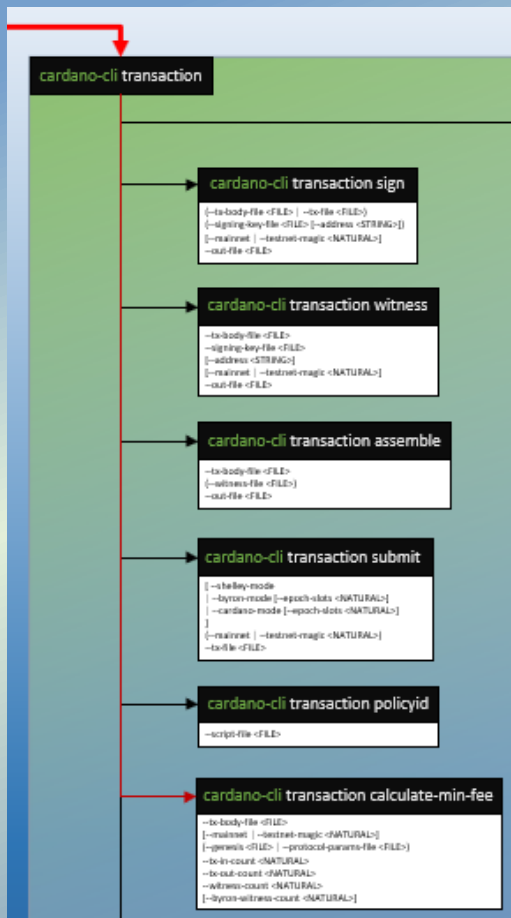
21 This is how it will look on your terminal:

```
user@computer:~$ cardano-cli transaction build-raw \
> --tx-in 1234a4d18e9dkhb34234kjbvd81e299c#0 \
> --tx-out $(cat paymentwithstake.addr)+0 \
> --invalid-hereafter 0 \
> --fee 0 \
> --certificate-file stake.cert \
> --out-file tx.raw
```

Congratulation! you got there. Save the command and options from **21** *in a text editor file, you will need them after the next exercise. Now you are going to calculate the fees that your transaction will cost you. Then you can subtract it from the amount of your UTXO (tx-in) and don't forget to include the deposit for the registration of the stake address.*

Ninth exercise: Calculation of the fees | Air Gap

1 Locate the branch that you are going to use for your fee calculation.

```
cardano-cli transaction

  cardano-cli transaction sign
    [--tx-body-file <FILE> | --tx-file <FILE>]
    [--signing-key-file <FILE>] [--address <STRING>]
    [--mainnet | --testnet-magic <NATURAL>]
    --out-file <FILE>

  cardano-cli transaction witness
    --tx-body-file <FILE>
    --signing-key-file <FILE>
    [--address <STRING>]
    [--mainnet | --testnet-magic <NATURAL>]
    --out-file <FILE>

  cardano-cli transaction assemble
    --tx-body-file <FILE>
    [--witness-file <FILE>]
    --out-file <FILE>

  cardano-cli transaction submit
    [--stellay-mode
    | --byron-mode [--epoch-slots <NATURAL>]
    | --cardano-mode [--epoch-slots <NATURAL>]]
    [--mainnet | --testnet-magic <NATURAL>]
    --tx-file <FILE>

  cardano-cli transaction policyid
    --script-file <FILE>

  cardano-cli transaction calculate-min-fee
    --tx-body-file <FILE>
    [--mainnet | --testnet-magic <NATURAL>]
    (--genesis <FILE> | --protocol-params-file <FILE>)
    --tx-in-count <NATURAL>
    --tx-out-count <NATURAL>
    --witness-count <NATURAL>
    [--byron-witness-count <NATURAL>]
```

2 You have 9 options in total.

This command will give you exactly the amount of fees you will have to pay depending on the number of tx-in, tx-out and the number of signatures required.

```
cardano-cli transaction calculate-min-fee

--tx-body-file <FILE>
[--mainnet | --testnet-magic <NATURAL>]
(--genesis <FILE> | --protocol-params-file <FILE>)
--tx-in-count <NATURAL>
--tx-out-count <NATURAL>
--witness-count <NATURAL>
[--byron-witness-count <NATURAL>]
```

3 Only 3 of these options will not be used.

- testnet-magic (obviously we use the mainnet on this tutorial)
- genesis (you will be using protocol params that you got earlier)
- byron-witness-count (because you don't use byron key pairs)

```
cardano-cli transaction calculate-min-fee

--tx-body-file <FILE>
[--mainnet | --testnet-magic <NATURAL>]
(--genesis <FILE> | --protocol-params-file <FILE>)
--tx-in-count <NATURAL>
--tx-out-count <NATURAL>
--witness-count <NATURAL>
[--byron-witness-count <NATURAL>]
```

4 3 options to be defined.

Let's specify the number of input and output addresses as well as the number of keys that you will use to sign your transaction.

```
cardano-cli transaction calculate-min-fee

--tx-body-file <FILE>
[--mainnet | --testnet-magic <NATURAL>]
(--genesis <FILE> | --protocol-params-file <FILE>)
--tx-in-count 1
--tx-out-count 1
--witness-count 2
[--byron-witness-count <NATURAL>]
```

5 Then you just have to indicate the PATH to your protocol.json file and your transaction draft tx.raw

```
cardano-cli transaction calculate-min-fee

--tx-body-file tx.raw
--mainnet
--protocol-params-file protocol.json
--tx-in-count 1
--tx-out-count 1
--witness-count 2
```

6 This is the result in your terminal. (The amount of the fees is not always the same.)

```
user@computer:~$ cardano-cli transaction calculate-min-fee \
> --tx-body-file tx.raw \
> --mainnet \
> --protocol-params-file protocol.json \
> --tx-in-count 1 \
> --tx-out-count 1 \
> --witness-count 2
```

If the command works as intended, the fees will appear at the bottom of it.

```
user@computer:~$ cardano-cli transaction calculate-min-fee \
> --tx-body-file tx.raw \
> --mainnet \
> --protocol-params-file protocol.json \
> --tx-in-count 1 \
> --tx-out-count 1 \
> --witness-count 2
178525 Lovelace
```

For the next exercise you will have to open the file of your text editor that you saved earlier with the "cardano-cli transaction build-raw" command from exercise eight. You are going to modify its content to build your final transaction.

Tenth exercise: Building the final transaction

1 This is your transaction draft from exercise eight.

You will modify it to enter the amount of the fees (which you know) and then you will calculate the amount of Lovelace to send back to your address.

cardano-cli transaction build-raw

```
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+0
--invalid-hereafter 0
--fee 178525
--certificate-file stake.cert
--out-file tx.raw
```

2 By using the command ''expr'' you can perform your calculation.

Amount of the UTXO   Stake address deposit

```
user@computer:~$ expr 10000000 - 178525 – 2000000
```
Fee

```
user@computer:~$ expr 10000000 - 178525 – 2000000
7821475
user@computer:~$
```

3 You can enter the result in your transaction.

Note that there must be no space between your address, the operator ''+'' and the amount in Lovelace. Otherwise, there will be an error when executing your command.

cardano-cli transaction build-raw

```
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+7821475
--invalid-hereafter 0
--fee 178525
--certificate-file stake.cert
--out-file tx.raw
```

4 Now, let's determine your ''TTL'' (time-to-live)

To choose from which Slot the transaction will become invalid, you need to know the Slot number you are in either by repeating exercise #6 or by checking your logs. Here is an example of what you could get:

```
{
    "block": 8749178,
    "epoch": 410,
    "era": "Babbage",
    "hash": "367e4af96abc18e1d4b5de08af535cb508e691...",
    "slot": 92029934,
    "syncProgress": "100.00"
}
```

5 Add a few minutes to it. (1 slot = 1 second)

To allow yourself to have time to sign your transaction and submit it on your ''hot node'', let's add 15 minutes to the option value.(92029934 + 900 = 92030834)

cardano-cli transaction build-raw

```
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+7821475
--invalid-hereafter 92030834
--fee 178525
--certificate-file stake.cert
--out-file tx.raw
```

6 This is the result in your terminal:

```
user@computer:~$ cardano-cli transaction build-raw \
> --tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0 \
> --tx-out $(cat paymentwithstake.addr)+7821475 \
> --invalid-hereafter 92030834 \
> --fee 178525 \
> --certificate-file stake.cert \
> --out-file tx.raw
```

Eleventh exercise: Signing your transaction

1 You are now ready to sign your transaction with your 2 private keys (payment.skey and stake.skey)

cardano-cli transaction

cardano-cli transaction sign

```
(--tx-body-file <FILE> | --tx-file <FILE>)
(--signing-key-file <FILE> [--address <STRING>])
[--mainnet | --testnet-magic <NATURAL>]
--out-file <FILE>
```

2 You have a total of 7 options.

For this one you will need to mention your ''transaction body file'', your 2 private keys PATH, the network to be used and the name of the file that you will submit to the blockchain.

cardano-cli transaction sign

```
--tx-body-file <FILE> | --tx-file <FILE>)
--signing-key-file <FILE> [--address <STRING>])
--mainnet | --testnet-magic <NATURAL>)
--out-file <FILE>
```

3 This is the result in your terminal:

```
user@computer:~$ cardano-cli transaction sign\
> --tx-body-file tx.raw \
> --signing-key-file payment.skey \
> --signing-key-file stake.skey \
> --mainnet \
> --out-file tx.signed
```

⚠️ **Note that in many situations, some options may be used multiple times.**

You can now transfer the ''tx.signed'' file to your ''Hot Node'' to submit it to the blockchain but first, make sure that the permissions of this one are set to ''Read only''.

Twelfth exercise: Submitting your transaction

1 You are now ready to submit your transaction!

cardano-cli transaction

cardano-cli transaction sign

```
(--tx-body-file <FILE> | --tx-file <FILE>)
(--signing-key-file <FILE> [--address <STRING>])
[--mainnet | --testnet-magic <NATURAL>]
--out-file <FILE>
```

cardano-cli transaction witness

```
--tx-body-file <FILE>
--signing-key-file <FILE>
[--address <STRING>]
[--mainnet | --testnet-magic <NATURAL>]
--out-file <FILE>
```

cardano-cli transaction assemble

```
--tx-body-file <FILE>
[--witness-file <FILE>]
--out-file <FILE>
```

cardano-cli transaction submit

```
--socket-path <SOCKET_PATH>
--shelley-mode
[--byron-mode [--epoch-slots <NATURAL>]
--cardano-mode [--epoch-slots <NATURAL>]
[--mainnet | --testnet-magic <NATURAL>]
--tx-file <FILE>
```

2 You have a total of 9 options.

The ''--socket-path'' option is not required if the PATH to your node socket file is already in your environment. You will only use what is required. In other words, the network and the name of the file to submit.

cardano-cli transaction submit

```
[--socket-path <SOCKET_PATH>]
[--shelley-mode
|--byron-mode [--epoch-slots <NATURAL>]
|--cardano-mode [--epoch-slots <NATURAL>]
]
--mainnet | --testnet-magic <NATURAL>)
--tx-file <FILE>
```

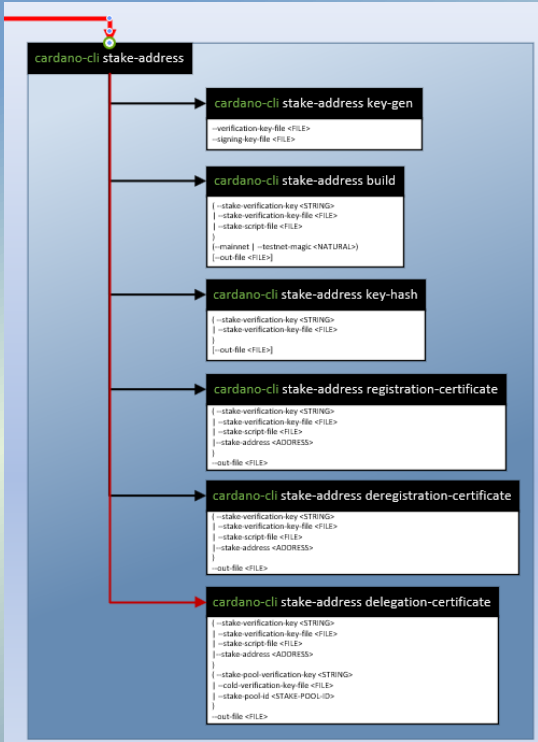**3** Here is this command in the terminal:

```
user@computer:~$ cardano-cli transaction submit \
> --mainnet \
> --tx-file tx.signed
```

```
user@computer:~$ cardano-cli transaction submit \
> --mainnet \
> --tx-file tx.signed
transaction successfully submitted
```

Congratulations, your stake address is now registered on the blockchain. You can now create a delegation certificate to choose a pool and participate in Cardano's ''Proof of Stake'' protocol. However, before moving on to the other exercise, be sure to delete your tx.signed file from your hot node.
(You won't need it anymore)

1 First, locate the branch that you are going to use for your certificate.

```
cardano-cli stake-address

  cardano-cli stake-address key-gen
    --verification-key-file <FILE>
    --signing-key-file <FILE>

  cardano-cli stake-address build
    | --stake-verification-key <STRING>
    | --stake-verification-key-file <FILE>
    |
    | --mainnet | --testnet-magic <NATURAL>)
    --out-file <FILE>

  cardano-cli stake-address key-hash
    | --stake-verification-key <STRING>
    | --stake-verification-key-file <FILE>
    |
    | --out-file <FILE>

  cardano-cli stake-address registration-certificate
    | --stake-verification-key <STRING>
    | --stake-verification-key-file <FILE>
    | --stake-script-file <FILE>
    | --stake-address <ADDRESS>
    --out-file <FILE>

  cardano-cli stake-address deregistration-certificate
    | --stake-verification-key <STRING>
    | --stake-verification-key-file <FILE>
    | --stake-script-file <FILE>
    | --stake-address <ADDRESS>
    --out-file <FILE>

  cardano-cli stake-address delegation-certificate
    | --stake-verification-key <STRING>
    | --stake-verification-key-file <FILE>
    |
    | --stake-pool-verification-key <STRING>
    | --cold-verification-key-file <FILE>
    | --stake-pool-id <STAKE-POOL-ID>
    |
    --out-file <FILE>
```

2 You have 8 options in total.

Once submitted onchain, this certificate will be used to indicate with which pool you will stake your ada. There are 2 groups of required options. These options will be chosen differently depending on your needs (ex: using cold.vrf for your own stake pool) In this excercise, we assume that you want to choose between several pool choices.

```
cardano-cli stake-address delegation-certificate

  | --stake-verification-key <STRING>
  | --stake-verification-key-file <FILE>
  | --stake-script-file <FILE>
  | --stake-address <ADDRESS>

  | --stake-pool-verification-key <STRING>
  | --cold-verification-key-file <FILE>
  | --stake-pool-id <STAKE-POOL-ID>

  --out-file <FILE>
```

3 So let's use --stake-address and --stake-pool-id

- Mention the PATH to your stake.addr.
- The stake pool ID to which you want to delegate your wallet. (can be Bech32-encoded or Hex-encoded)
- The name of your certificate file.

```
cardano-cli stake-address delegation-certificate

  | --stake-verification-key <STRING>
  | --stake-verification-key-file <FILE>
  | --stake-script-file <FILE>
  | --stake-address stake.addr

  | --stake-pool-verification-key <STRING>
  | --cold-verification-key-file <FILE>
  | --stake-pool-id pool1mt8sdg37f2h3rypyuc77k7vxrjshtvjw04zdjlae9vdzyt9uu34

  --out-file delegation.cert
```
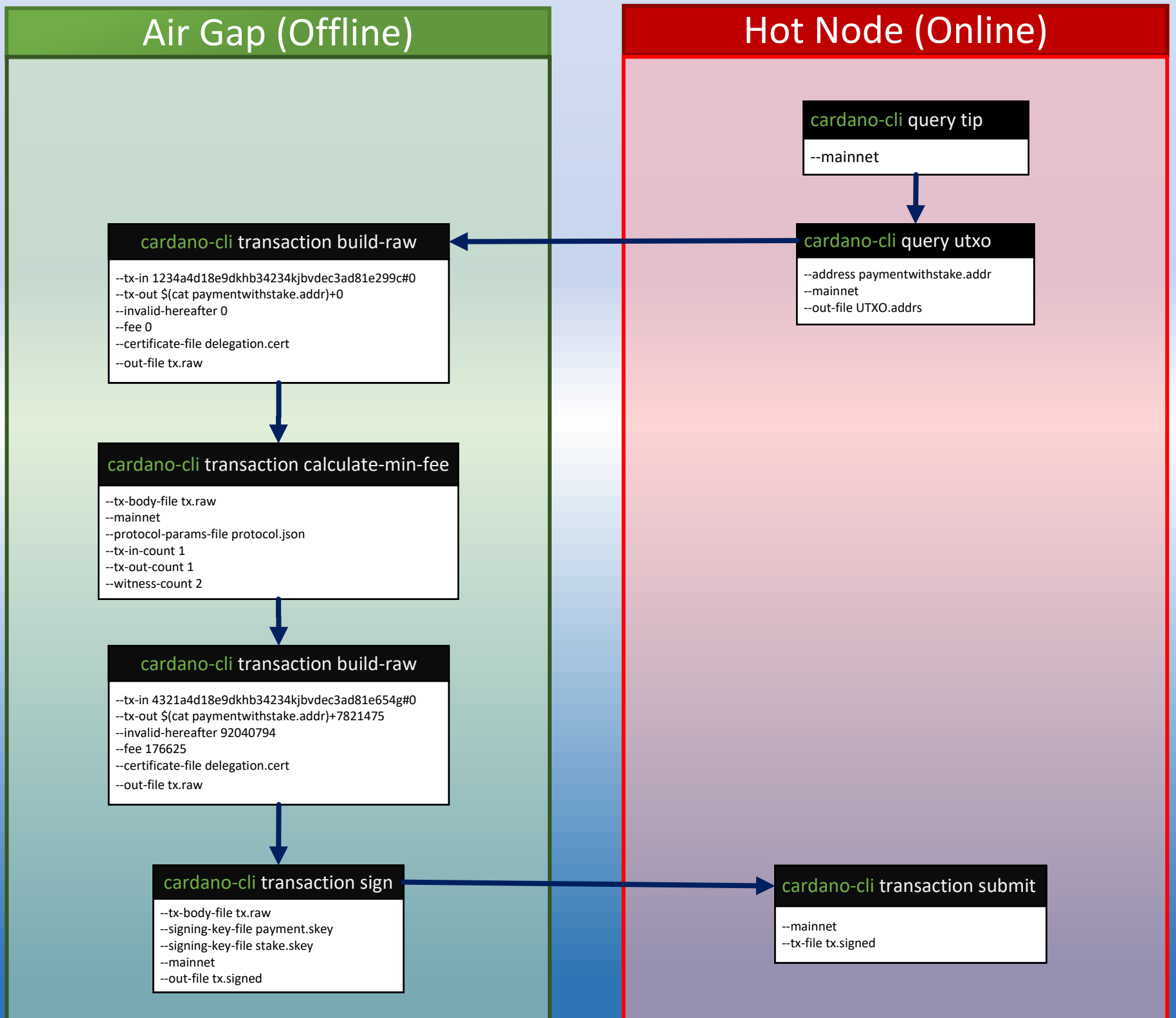
4 This is the result in your terminal:

```
user@computer:~$ cardano-cli stake-address delegation-certificate \
> --stake-address stake.addr \
> --stake-pool-id pool1mt8sdg37f2h3rypyuc77k7vxrjshtvjw04zdjlae9vdzyt9uu34 \
> --out-file delegation.cert
```

⚠ You can get the stake pool ID from cexplorer.io or if you like this document, let us know, we could add the commands: "query stake-pools", "query pool-state" and "query pool-distribution" to the 2nd chapter of this tutorial.

You can now repeat exercises 6 to 12, making sure to replace the stake.cert with the delegation.cert when you build your transaction. And don't forget that when calculating the fees you should not take into account the stake address deposit. (which has already been done.)

Air Gap (Offline)

```
cardano-cli transaction build-raw
--tx-in 1234a4d18e9dkhb34234kjbvdec3ad81e299c#0
--tx-out $(cat paymentwithstake.addr)+0
--invalid-hereafter 0
--fee 0
--certificate-file delegation.cert
--out-file tx.raw
```

```
cardano-cli transaction calculate-min-fee
--tx-body-file tx.raw
--mainnet
--protocol-params-file protocol.json
--tx-in-count 1
--tx-out-count 1
--witness-count 2
```

```
cardano-cli transaction build-raw
--tx-in 4321a4d18e9dkhb34234kjbvdec3ad81e654g#0
--tx-out $(cat paymentwithstake.addr)+7821475
--invalid-hereafter 92040794
--fee 176625
--certificate-file delegation.cert
--out-file tx.raw
```

```
cardano-cli transaction sign
--tx-body-file tx.raw
--signing-key-file payment.skey
--signing-key-file stake.skey
--mainnet
--out-file tx.signed
```

Hot Node (Online)

```
cardano-cli query tip
--mainnet
```

```
cardano-cli query utxo
--address paymentwithstake.addr
--mainnet
--out-file UTXO.addrs
```

```
cardano-cli transaction submit
--mainnet
--tx-file tx.signed
```

We will finish the part 1 of this tutorial with a quote from an SPO colleague that I greatly appreciate:
"We should encourage new SPOs, even if they have low skill. They will learn, and Cardano will decentralize."
--@StakeWithPride