Air Gap = Студена Среда (Offline)

Hot Node = Синхронизиран Node (Online)

Cardano-cli:~$ Study sheets (Bulgarian)

Parte 1: Ключове, Адреси и Делегиране

Превод - Васил Раков

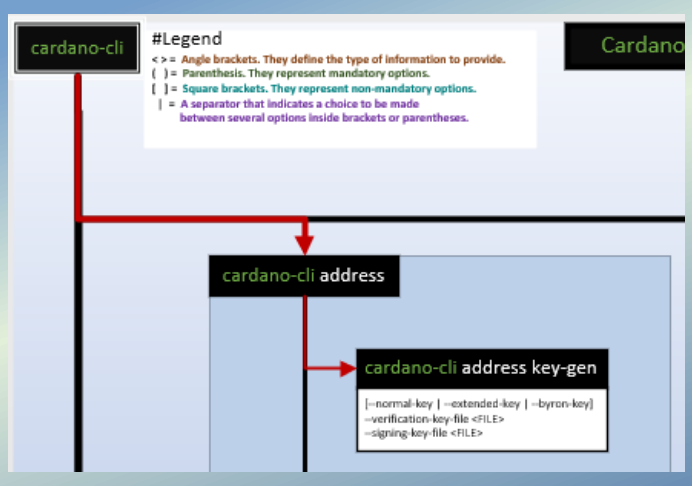
Clover Nodes
Blockchain Telecommunication Services

Това ръководство е създадено за да се използва с Печатната версия на Cardano-CLI V8.0.0. синтезираният вариант

Този документ има за цел да обясни подробно как да интерпретирате командите cardano-cli и le свързани опции, за да можете да ги сглобите сами, ако е необходимо. За да направите това, трябва да имате компютър и да инсталирате Cardano blockchain Node и онлайн интерфейс Команда Cardano (cardano-cli). Ще започнете с прости команди и постепенно ще усложнявате докато урокът напредва.

Първо упражнение: Създаване на ключове за плащане и залог — Air Gap

1 **Първо намерете командата, която ще използвате при вашите ключове за плащане.**



2 **Имате общо 5 опции.**

Първите 3 опции имат, са включени в квадратни скоби с 2 разделителя, които показва, че избор между тези 3 опции не се изисква, тъй като ще се използва един нормален ключ за настройка по подразбиране (default) ако не се посочи нищо. За този пример, няма да ги използваш

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

3 **Трябва да се използват следните 2 опции**

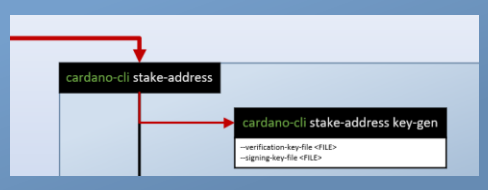
Ъгловите скоби показват вида на Информация <FILE>. Трябва да посочите име което ще дадете на своите 2 ключови файла

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

4 **Това е крайният резултат от тази проста команда на вашият терминал.**

```
user@computer:~$ cardano-cli address key-gen \
> --verification-key-file payment.vkey \
> --signing-key-file payment.skey
```

5 **След като вашите ключове за плащане са готови, трябва да създадете вашите ключове за stake/залог. Тази процедура дори по-лесна, защото има само един тип ключ за stake/участие в създаване на Блок.**




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

6 **Същото като 3 . Трябва да посочите вида на информацията. <FILE>**

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

7 **И като краен резултат на терминала.**

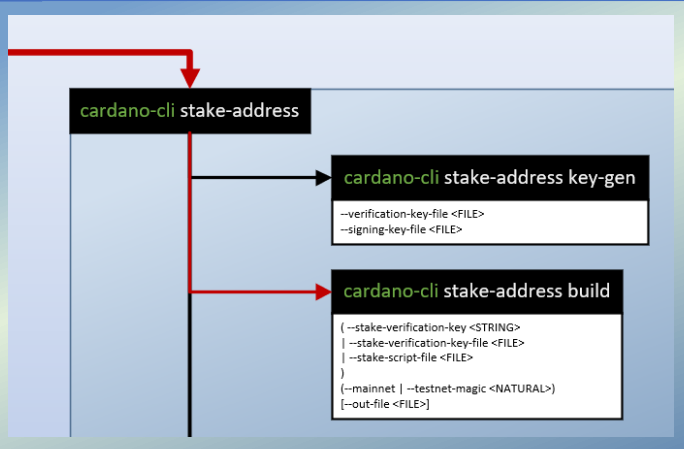
```
user@computer:~$ cardano-cli stake-address key-gen \
> --verification-key-file stake.vkey \
> --signing-key-file stake.skey
```

Внимание. Препоръчително е Да генерирате *и използвате вашите ключове за плащане и делегиране при подписване на транзакции в среда "Air Gap" от съображения за безопасност. https://developers.cardano.org/docs/get-started/air-gap*

Сега, след като вашите 2 двойки ключове са създадени, вие ще можете да създадете адрес на стейка, който ще изследва, и ще ви позволи да проверите размера на вашите награди и ще ги изтеглите, когато се използват в транзакция с вашият stake.skey.

Второ упражнение: Създаване на стейк адрес — Air Gap

1 **Намерете командата, която ще използвате за изграждане на вашият stake адрес.**



2 **Имате общо 6 опции**

Първите 3 опции са включени в скоби и имат 2 разделителя, които показват, че е задължително да се направи избор между тези три варианта.

```
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 **Ще използвате опцията stake-verification-key-file.**

Ъгловите скоби показват вида на информацията <FILE>. Този път трябва да осигурите пътя, който сочи към вашият

```
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 **Сега имате 2 опции в скоби.**

Ще трябва да уточните използваната мрежа и дали тя е в тестови режим, споменавайки магическият номер на същата. Нека използваме основната мрежа.

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

5 **И за последният вариант –out-file <FILE>**

Тази опция не е задължителна, защото е оградена в квадратни скоби. Ако обаче не ползвате тази опция, изходът (стейк адрес) на командата ще бъде показан на вашия терминал, вместо да бъде записан във файл. И докато ще ви трябва да използвате този адрес на делегиране по-късно, нека му дадем име и нека го запазим във файл.

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

6 **Ето как командата се появява на вашият терминал.**

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

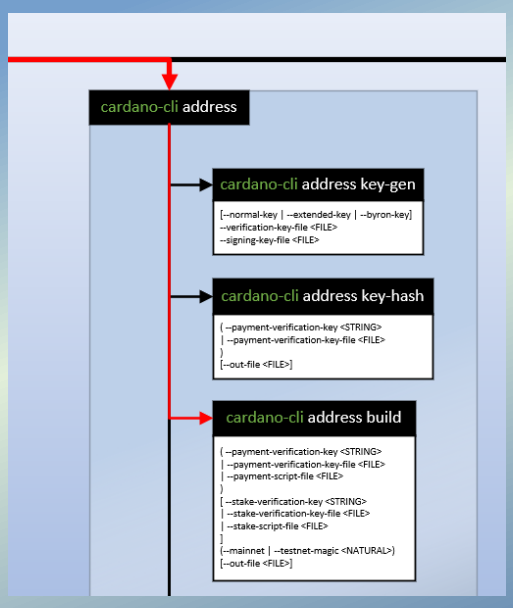
7 **Ето какво трябва да имате досега**

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

Сега, след като двете двойки ключове и стейк адресът са създадени, вие ще можете да произведете адрес, като комбинирате своят ключ за плащане със стейк ключа, така че парите в генерираният адрес са включени в делегирания протокол с вашите награди.

Трето упражнение: Създаване на плащане със стейк адрес файла — Air Gap

1 **Намерете командата, която ще използвате за вашето плащане със стейк адрес файла.**



2 **Имате общо 9 опции.**

Можете да създадете адрес за плащане, като използвате само вашият ключ за плащане, без да го свързвате с вашия стейк ключ, което обяснява защо първата група от опции е задължителна, а не втората

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 **Нека ползваме и двата ключа: за плащането и потвърденият ключ за делегирането.**

Отново според ъгловите скоби <FILE>, вие трябва да дефинирате пътя до файловете за плащане vkey и вашият 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 **Сега ще трябва да посочите мрежата, която да ползвате, и името на файл а на вашият адрес.**

cardano-cli address build

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

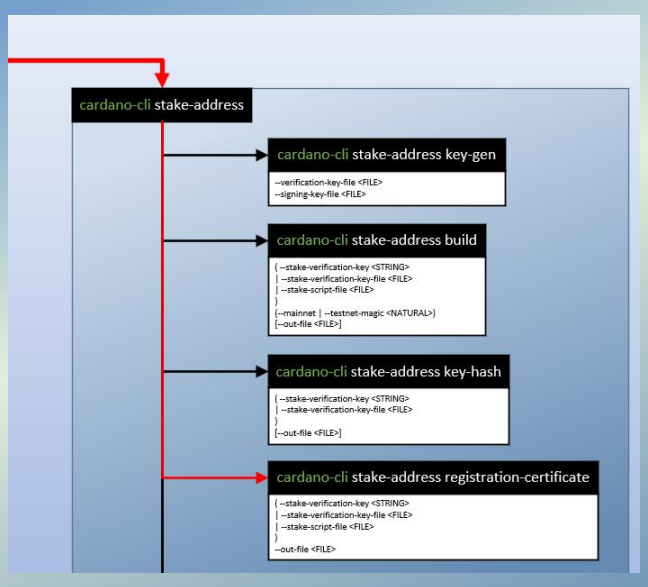
5 **Това е крайният резултат.**

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

Можете да копирате съдържанието на paymentwithstake.addr в текстов файл и го поставете за транзакция в портфейл за Cardano, който обикновено ползвате, и изпратете малко ADA към него. (10 ADA трябва да са достатъчни за да започнете)

Четвърто упражнение: Създаване на стейк сертификат — Air Gap

1 **Намерете командата, която ще ползвате за вашият стейк сертификат.**



2 **Имате 4 възможности**

За да участвате в протокола и да заложите своя ADA, трябва да свържете своя ключ за потвърждение от стейка към сертификата, който ще изпратите на блокчейна в следващите няколко упражнения. Командата за създаване на вашият сертификат е доста опростена. Просто трябва да предоставите една от тези 3 задължителни опции и да посочите името на файла, който ще действа като сертификат.

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 **Това е крайният резултат как командата трябва да изглежда на вашият терминал.**

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

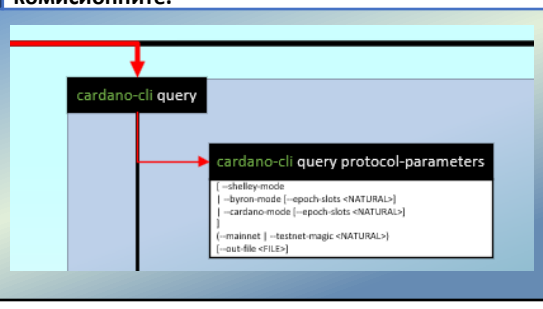
4 **Ето какво трябва да имате досега.**

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

Сега ще получите параметри на протокола и индикатор за прогрес на блокчейна (като съвет), за да можете да започнете изграждането на първата си транзакция.

Пето упражнение: Получаване параметрите на протокола — Hot Node

1 **На първо място, за вашата транзакция ще ви трябват параметрите на протокола за изчисляване на комисионните.**



2 **Имате общо 6 опции и 2 подварианти**

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 **Пропуснете опциите „режим“. Определете желаната мрежа и името на файла за създаване.**

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 Това е крайният резултат от това как командата ще се появи във вашият терминал.

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

5 Сега продължете и нека видим съдържанието на този файл:

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

Във файла protocol.json ще потърсите депозита, който да бъде изпратен на блокчейна, за да регистрирате вашият стейк адрес и участието ви в стейкинг протокола/пула. Този депозит може да бъде възстановен по всяко време, ако дерегистрирате своя адрес.

6 Обърнете внимание на сумата на депозита, тъй като ще ви трябва по-късно. Сумата е изразена в Lovelace. (1 ADA = 1 000 000 Lovelace)

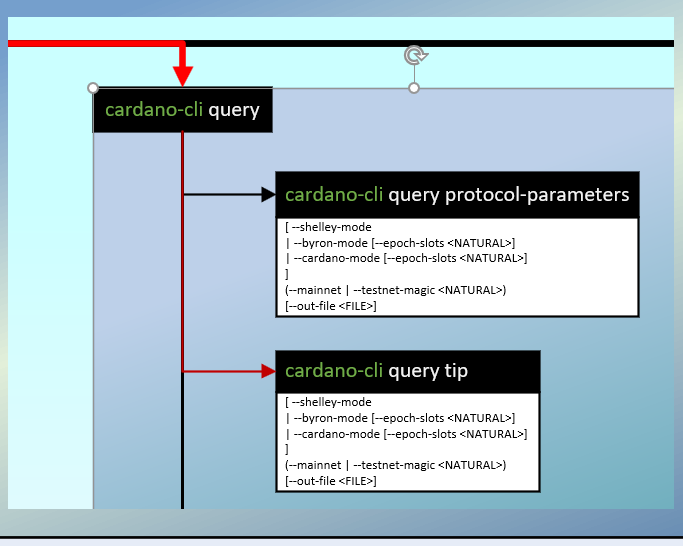
```
"poolRetireMaxEpoch": 18,
"protocolVersion": {
    "major": 8,
    "minor": 0
},
"stakeAddressDeposit": 2000000,
"stakePoolDeposit": 500000000,
"stakePoolTargetNum": 500,
"treasuryCut": 0.2,
"txFeeFixed": 155381,
"txFeePerByte": 44,
"utxoCostPerByte": 4310,
"utxoCostPerWord": null
```

7 Сега трябва да вземете вашият файл protocol.json и да го прехвърлите във вашата "Air Gap" среда, командата ще се появи във вашият терминал. за да можете да изчислите комисионните, при изграждането на транзакциите си.

⚠ **С прилагането на CIP-1694 и наближаващата епоха на Волтер това ще бъде възможно за притежателите на ADA общността, с помощта на конституционната комисия и на Dreps, да се променят параметрите на протокола чрез структуриране на добра система за гласуване. Заради това е важно да сте сигурни, че имате най-актуалните промени на тези протоколи във вашата среда "Air Gap", тъй като това може да има пряко въздействие върху различните параметри оросредстващи вашите транзакции.**

Шесто упражнение: Вземете текущият полезен съвет за нода — Hot Node

1 В следващия урок ще трябва да се запознаете с полезен съвет за нода, с цел да изчислим нашия TTL (Time-to-Live). (Ще бъде предоставено подробно описание в следващото упражнение)



2 Както в предишното упражнение, ще имате 6 налични опции и 2 подварианти.

Сега, когато започвате да разбирате напълно принципа, можете да пропуснете някои стъпки. Не е необходимо да създавате файл, имате нужда само от номера на слота.

cardano-cli query tip

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

3 Това е крайният резултат на вашия терминал.

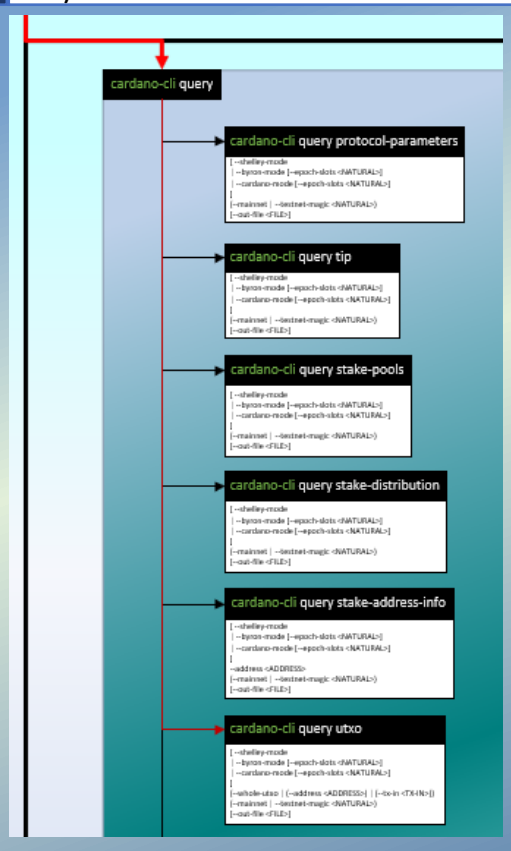
```
user@computer:~$ cardano-cli query tip \
> --mainnet
```

4 Обърнете внимание на слот номера

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

Седмо упражнение: Разпитайте за UTXO — Hot Node

1 Сега ще направите заявка за UTXO на адреса paymentwithstake.addr. (Ако сте изпратили там ADA)



2 Тази команда има 9 опции и 2 подопции.

Ще трябва да консумирате поне един UTXO като вход към вашата транзакция. Една транзакция може да съдържа няколко входове и няколко изхода, но в този случай трябва да имате само един UTXO, свързан с вашия адрес: paymentwithstake.addr, защото си направил само един 10 ADA депозит на този адрес. И така 10 ADA депозит на този адрес. Така че, ще използваме само това, което е задължително. Накратко, вашият paymentwithstake.addr, адрес, ще го ползваме в мрежата и тогава ще създадем файл за да прехвърлете този UTXO списък във вашата "air gap" среда/ директория.

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 Ето как трябва да изглежда на вашият терминал

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 Съдържанието на файла utxo.addr трябва да изглежда така. UTXO на вашия 10 ADA депозит е равен на 10 000 000 Lovelace. Вече можете да вземете този файл и да го поставите във вашата среда "Air Gap". Ще ви потрябва много скоро.

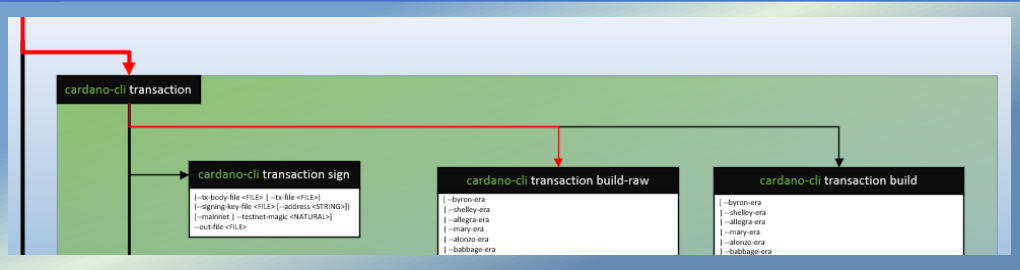
```
                           TxHash                                 TxIx        Amount
--------------------------------------------------------------------------------------
1234a4d18e9dkhb34234kjbvdec3ad81e299c1a523443453561e61ce9bf8608e8c802df3b7f8c     0        10000000 lovelace + TxOutDatumNone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

Сега е време да създадете първата си транзакция, която ще се използва за изпращане на вашият стейк сертификат. Преди да започнете, това което ще видите, може да ви изглежда смущаващо, но като следвате стъпка по стъпка, трябва да напредвате и да сте в състояние да разбере защо и как да намалите следващите опции до общо 6 такива за процеса на транзакцията. За целите на сигурността в този урок ще използвате методи, включващи „Cardano-cli транзакцията "build-raw", вместо командата "Cardano-cli Transaction Build", тъй като може да бъде изградена в офлайн среда.

Осмо упражнение: Създаване на собствена чернова, първа транзакция

Air Gap

1 **Намерете командата „cardano-cli transaction build-raw“**



2 **Постепенно започваме отгоре надолу.**

```
cardano-cli transaction build-raw
[ --byron-era
| --shelley-era
| --allegra-era
| --mary-era
| --alonzo-era
| --babbage-era
]
```

3 **Първите 5 опции не са задължителни и са оградени в [квадратни скоби]. Ако нищо не е посочено, ще се използва по подразбиране в ерата - Mary.**

```
cardano-cli transaction build-raw
[ --byron-era
| --shelley-era
| --allegra-era
| --mary-era
| --alonzo-era
| --babbage-era
]
[--script-valid | --script-invalid]
```

4 **Вашата транзакция не включва скрипт, така че можете да я пропуснете в следващите 2 опции.**

```
cardano-cli transaction build-raw
[ --byron-era
| --shelley-era
| --allegra-era
| --mary-era
| --alonzo-era
| --babbage-era
]
[--script-valid | --script-invalid]
```

5 **За следващите опции и нейните 20 подопции е така необходимо и обяснение.**

В рамките на дадена опция могат да присъстват скоби подопции, определени от колони. Ето защо съществува понятието за приоритети и определен ред, който трябва да се спазва при изграждането на такъв род тразакция. В този случай знаем, че има 3 отделни колони, определящи реда, в който опциите трябва да бъдат вмъкнати, ако искаме тялото на транзакцията да бъде създадено правилно.

```
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 **Отделете време, за да анализирате внимателно приоритетният ред на опцията "tx-in" и нейните скоби.**

```
[--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“ е задължителен, но неговите подопции не са.**

Следните опции не са задължителни и се използват за скриптове Plutus. Така че не е необходимо да ги използвате За нашата елементарна транзакция.

```
--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 **За това което следва:**

Няма да имате нужда от „въвеждане на справки само за четене“ или нещо друго по отношение на обезпечението, тъй като е обикновена транзакция която няма да включва скрипт Plutus.

```
[--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 **Othocho “--required-signer-hash <HASH>”**

Тази опция няма да е полезна в момента за вас, като транзакция, която ще се използва за публикуване на вашия stake.cert на blockchain, но е важно да се отбележи, че ще бъде много полезна в упражнението, свързано с вота за управление.

```
[--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 **Най-накрая! Опция, от която ще имате нужда, “--tx-out”.**

Тази опция ще ви трябва, за да посочите адреса, от който ще получите баланса от консумирания от вас UTXO, минус таксите. Така че нека да копираме тази опция и да я добавим към вашата чернова за транзакциите.

```
[--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 **Подопциите "tx-out" относно скрипта Plutus може да се пропусне. За сега не ви трябват.**

```
--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 **Постепенно започвате да разбирате. Без multiasset, без NFT, без Plutus скриптове за вашите транзакции.**

```
[--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 **Ще трябва да използвате 3 от следващите 4 опции.**

- ''--invalid-before'' определя от кой слот ще бъде валидна транзакцията за да бъде обработена.
- mentre ''--invalid-after'' определя от кой слот транзакцията ще стане невалидна. (точно като срок на годност)

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

14 **Вземете файла с TTL, за такса и Файл-сертификат.**

Ето защо изпълнихте командата ''cardano-cli query tip'' от някои предишни упражнения. Знаейки номера на слота на вашия синхронизиран. нод/възел. Вече можете да определите „time to live“ или „TTL“ за транзакция, докато сте в пула за памет. Така че, сега можете да добавите

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

15 **Отново не се използват опции, свързани със сертификати за скриптове на Plutus.**

```
--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 **Опцията „--withdrawal“ е вход, който ви позволява да изтеглете наградите си от вашият 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 **Можете да пропуснете опцията "-withdrawal" и свързаните с нея Plutus скриптове за сега.**

```
[--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 **Остават само още няколко опции.**

В момента нямате метаданни за изпращане, на криптиран файл, нито необходимостта от указване на генезис файл или параметри на протокола. Освен това вие не изпращате предложение актуализация за фонда Catalyst. Това, което ви остава е просто възможността да наименувате файла за вашата чернова за транзакции

```
[--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 **Чрез групиране на опциите, които сте копирали по време на това упражнение ще получите нещо подобно:**

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 **Сега, за да завършите черновата си:**

Добавете UTXO на входа (tx-in), адреса за останалите (tx-out) и файла със сертификата. Засега присвоете стойност 0 на tx-out, недействителен за по-нататък и такса.

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 **Ето как ще изглежда на вашия терминал:**

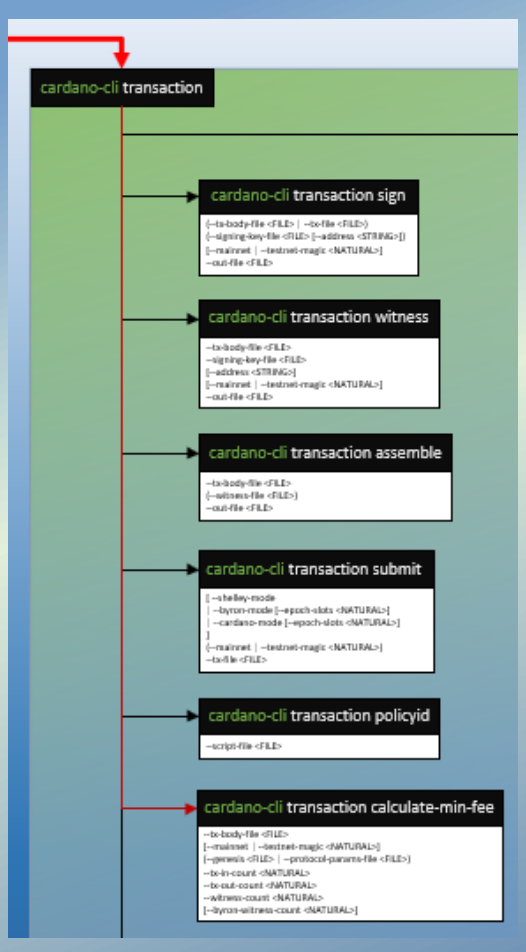
```
user@computer:~$ cardano-cli transaction build-raw \
> --tx-in 1234a4d18e9dkhb34234kjbvd81e299c#0 \
> --tx-out $(cat paymentwithstake.addr)+0 \
> --invalid-hereafter 0 \
> --fee 0 \
> --certificate-file stake.cert \
> --out-file tx.raw
```

Честито! Вие го направихте. Запазете командата и опциите от стъпка 21 в редактор на текстови файлове, за да ги имате при нужда след следващото упражнение. Сега ще изчислите комисионите (таксите), които вашата транзакция ще ви струва. Така можете да ги извадите от вашата UTXO (tx-in) сума и не забравяйте да включите регистрационния депозит от адреса на стейка.

Девето упражнение: Изчисляване на такси

Air Gap

1 **Намерете командата, която ще използвате за изчисляване на таксите (fees).**



2 **Имате общо 9 опции.**

Тази команда ще ви даде точно размера на таксите които ще трябва да платите въз основа на броя tx-in, tx-Out при необходимия брой подписи.

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 **Само 3 от тези опции няма да бъдат използвани.**

- testnet-magic (очевидно ще използваме основната мрежа в този урок)
- genesis (ще използвате параметрите на протокола, които сте въвели по-рано)
- byron-witness-count (защо не използвате двойки ключове за ерата)

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 **Ще бъдат използвани само 3 от тези опции.**

Нека определим броя на входните и изходните адреси и броя на ключовете, които ще използвате, за да подпишете своята транзакция.

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 **След това просто насочвате ПЪТЯ към вашият файл: protocol.json и вашата чернова на транзакцията: 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 **Това е резултатът във вашия терминал. (Размерът на комисионните/таксите, не са винаги същите.)**

```
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
```

Ако командата работи според очакванията, таксите ще се покаже в долната му част. Ако командата работи според оча

```
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
```

За следващото упражнение ще трябва да отворите файла с вашия текстов редактор, който преди сте запазили командата "cardano-cli transaction build-raw", в осмо упражнение. Ще редактирате съдържанието му, за да създадете своята финална транзакция за да за да създасвоята фина

Десето упражнение: Конструиране на финалната сделка

Air Gap

1 Това е вашата чернова на транзакцията от осмо упражнение.

Ще го промените, за да вмъкнете размера на таксите, вече знаете как и след това изчислете количеството Lovelace за изпращане до вашия адрес

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 С помощта на командата "expr" можете да извършите изчислението.

Сумата от UTXO

Депозит във стейк адреса

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

такси за транзакции

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

3 Можете да въведете резултата във вашата транзакция.

Имайте предвид, че не трябва да има интервал между вашия адрес, операторът "+" и сумата в Lovelace. Иначе ще отрази грешка при изпълнение на командата ви.

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 Сега да определим вашето "TTL" (time-to-live)

За да разберете от кой слот транзакцията ще бъде невалидна трябва да знаете слот №, в които сте се намирали но-рано като повторите упражнение №6 или като проверите вашите влизания Ето пример за това, което може да получите:

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

5 Добавете няколко минути към него. (1 слот = 1 секунда)

За да имате време да подпишете вашата транзакция и я изпратете на вашия „горещ нод“, добавете 15 минути към стойността на опцията. (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
```

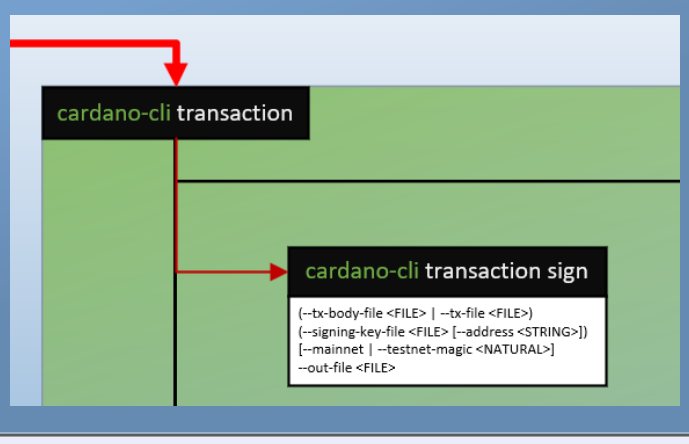
6 Това е резултатът във вашия терминал:

```
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
```

Единадесето упражнение: Подписване на вашата транзакция

Air Gap

1 Вече сте готови да подпишете вашата транзакция с вашите два лични ключа (payment.skey и stake.skey).



2 Имате общо 7 опции.

За целта ще трябва да споменете пътя до вашата транзакция, пътищата до вашите 2 лични ключа, мрежата на ползване и името на файла, който ще изпратите към блокчейна

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 Това е резултатът на вашия терминал:

```
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
```

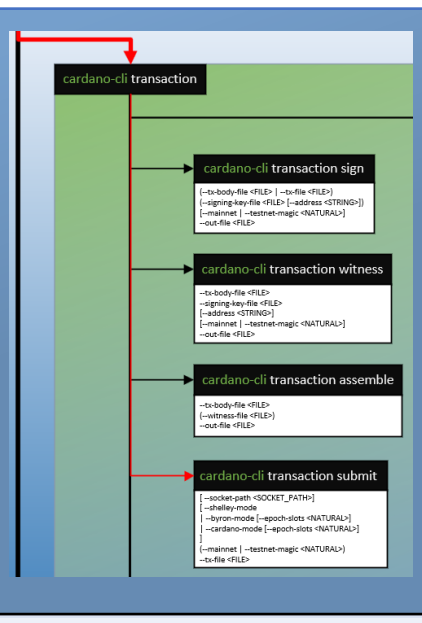
⚠ Имайте предвид, че в много ситуации можете да ползвате някои опции повече от веднъж

Вече можете да прехвърлите файла „tx.signed“ във вашия „Горещ нод“, за да го изпратите до блокчейна, но първо се уверете, че достъпът до този файл е настроен на: "само за четене".

Дванадесето упражнение: Изпратете транзакцията си

Hot Node

1 Вече сте готови да изпратите транзакцията си!



2 Имате общо 9 опции

Опцията "--socket-path" не е необходима, ако пътят към сокет Файлът на вашият нод вече съществува в средата. Ще използвате само това, което е необходимо. С други думи, мрежата и името на файла за изпращане.

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 Това е командата в терминала:

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

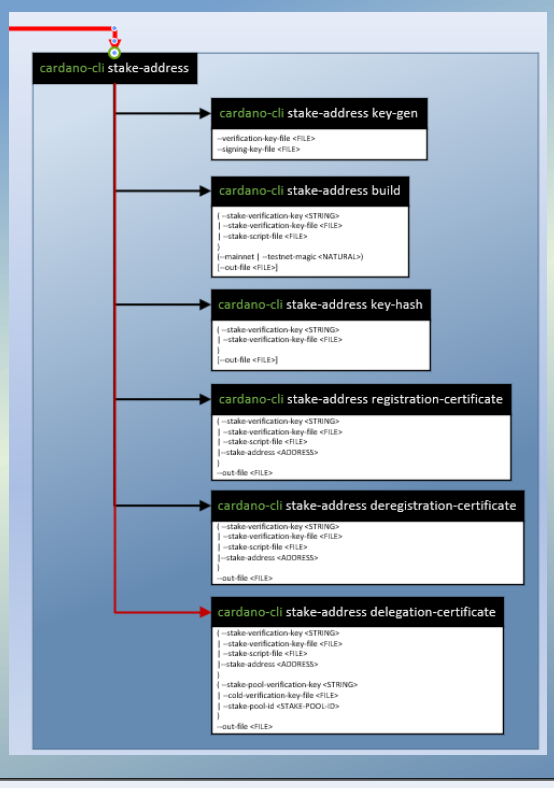
```
user@computer:~$ cardano-cli transaction submit \
> --mainnet \
> --tx-file tx.signed
transaction successfully submitted
```

Поздравления, вашият стейк адрес вече е регистриран в блокчейна. Вече можете да създадете сертификат за делегиране за да изберете стейк пул и да участвате в протокола „Proof of Stake“ на Cardano. Въпреки това, преди да преминете към следващото упражнение, не забравяйте да изтриете файла tx.signed от вашия активен нод.
(повече няма да ви трябва)

Thirteenth exercise: Creation of a delegation certificate

Air Gap

1 Първо намерете командата, която ще използвате за вашия сертификат



2 Имате общо 8 опции.

Веднъж изпратен до блокчейна, този сертификат ще се използва за да посочете с кой пул искате да заложите своята ADA. Има две групи от необходими опции. Тези опции ще бъдат избрани различно в зависимост от вашите нужди (например използване cold.vrf за вашият личен Stake Pool. В това упражнение нека приемем, че вие искате да избирате между различни опции на stake pool.

```
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 Тогава използваме --stake-address и --stake-pool-id

- Посочете пътя (PATH) на вашият stake.addr
- ID номера на stake pool на който искате да делегирате портфолиото си. (може да бъде кодиран в Bech32 или шестнадесетичен код)
- Името на вашият файл със сертификат.

```
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
```

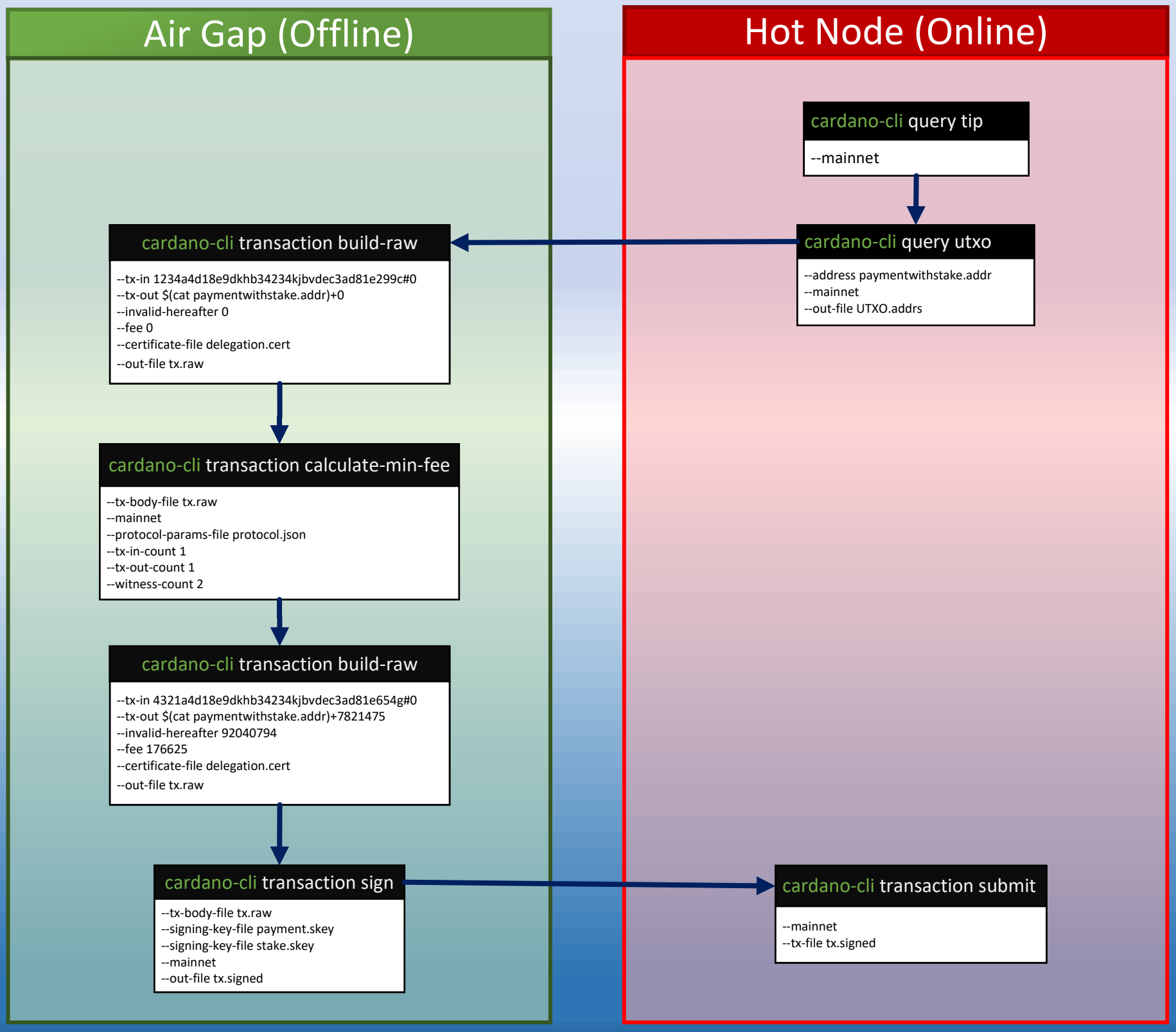
4 Това е резултатът на вашия терминал

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

Можете да получите ID на стейк пула чрез cexplorer.io или, ако харесвате този документ, моля, уведомете ни и можем да добавим команди "query stakepools", "query pool-state" е "query pool-distribution"към втора глава от този урок.

Вече можете да повторите упражнения от 6 до 12, като се уверите, че сте заменили файла stake.cert с файла delegation.cert, когато създавате своя транзакция и не забравяйте, че когато изчислявате таксите, не трябва да депозирате на стейк адреса 2 ADA (както вече беше направено преди).

Резюме на операциите: Процес на подаване на сертификат за делегиране



Ще завършим част 1 от този урок с цитат от колега SPO, който наистина ценя:

"Трябва да насърчаваме новите SPO, дори ако имат ниски умения. Те ще се научат и Cardano ще се децентрализира."

--@StakeWithPride