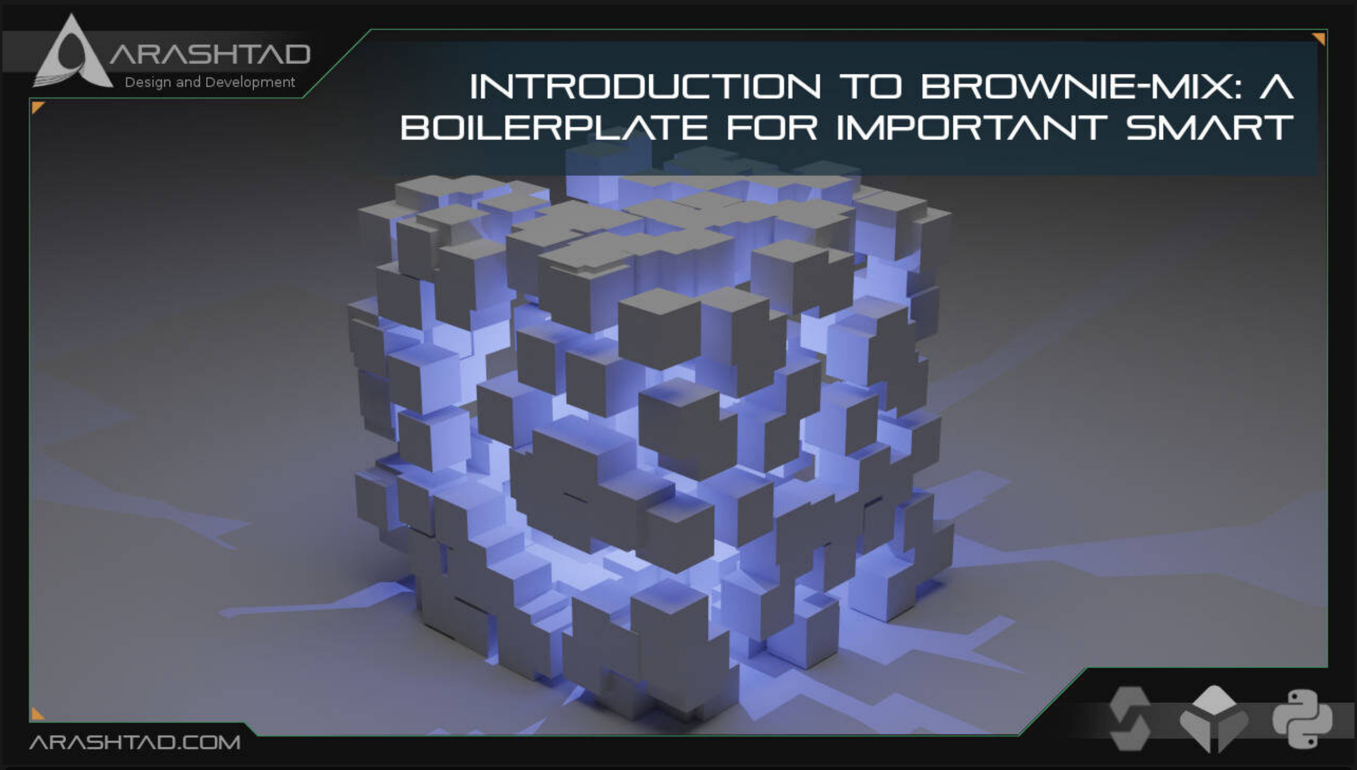
Introduction to Brownie-mix: A Boilerplate for Important Smart Contract Projects

No comments


ARASHTAD
Design and Development
INTRODUCTION TO BROWNIE-MIX: A BOILERPLATE FOR IMPORTANT SMART
ARASHTAD.COM

In this article, we are going to get familiar with Brownie-mix as a boilerplate for blockchain projects in python. Using this boilerplate with the Brownie bake command will help you have so many of the dependency contracts provided for that specific project. There is also a complete brownie_config.yaml file provided with a complete list of networks. Some ready python files such as helpful_scripts.py and deploy_mocks.py are provided for you.

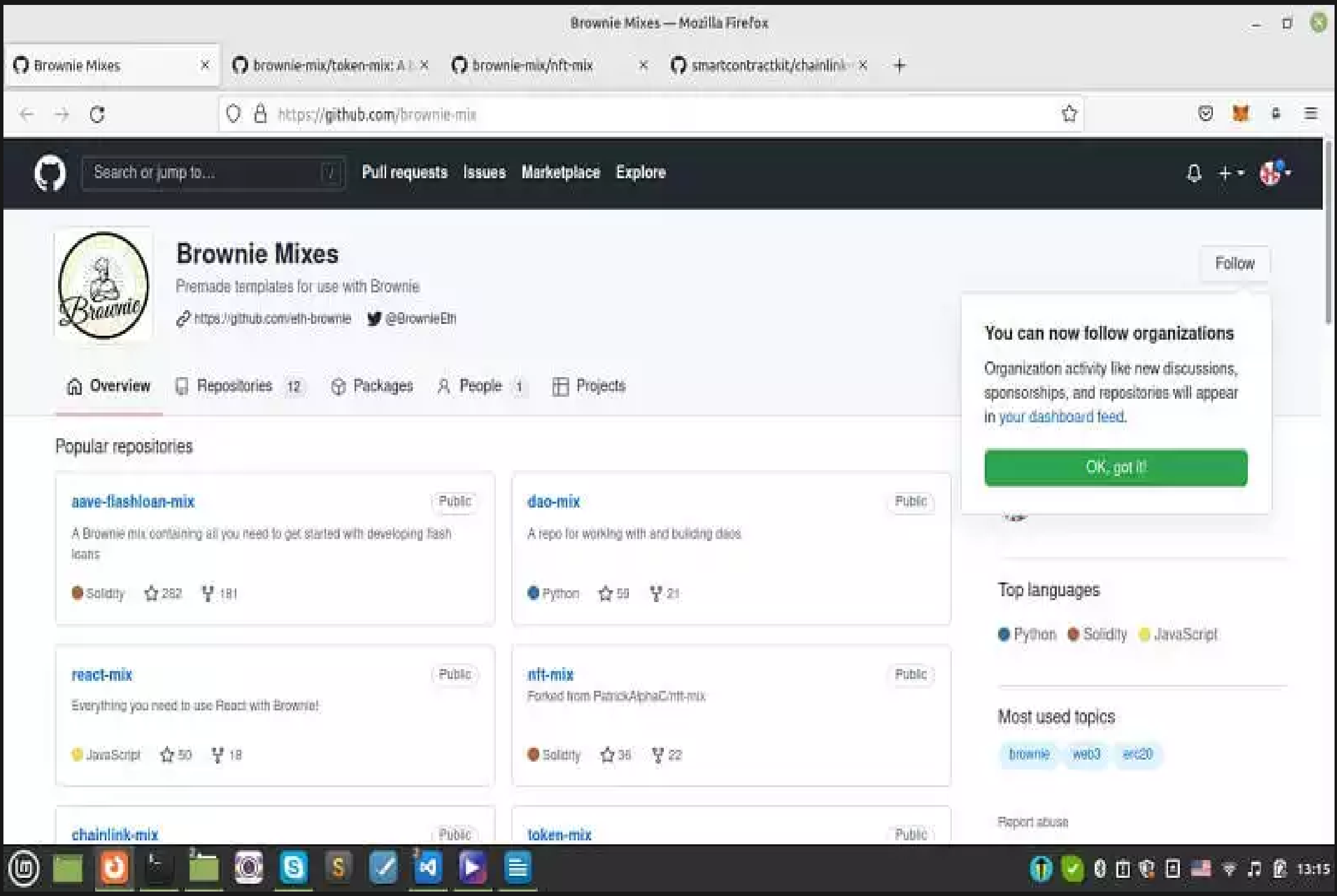
Introduction to the Brownie-mix

When we were examining how to deploy a smart contract using python web3 tools among our previous articles, we showed how to use Brownie. It is highly recommended that before you start this tutorial, be familiar with web3 python tools and also solidity language.
So far, we have coded most of our smart contract deployments using Brownie by starting like this in the terminal:

```
brownie init
```

And then some folders would have been created and then the rest of the project. But the hard part was that we needed to copy a lot of dependencies, such as the VRFConsumer.sol and other smart contracts, brownie-config.yaml file, helpful_scripts.py, deploy_mocks.py, and a lot of other useful scripts that we need to rewrite every time we created the project.

A Brownie-mix helps us cover this hard task and provides easy boilerplates for every type of project like DAO, NFTs, Token, ChainLink, and so on.



Brownie Mixes
Premade templates for use with Brownie
https://github.com/eth-brownie @BrownieEth

You can now follow organizations
Organization activity like new discussions, sponsorships, and repositories will appear in your dashboard feed.

Popular repositories

aave-flashloan-mix Public
A Brownie mix containing all you need to get started with developing flash loans
Solidity ☆ 282 181

dao-mix Public
A repo for working with and building daos
Python ☆ 59 21

react-mix Public
Everything you need to use React with Brownie!
JavaScript ☆ 50 18

nft-mix Public
Forked from PatrickAlphaC/nft-mix
Solidity ☆ 36 22

chainlink-mix Public

token-mix Public

Top languages
Python Solidity JavaScript

Most used topics
brownie web3 erc20

Report abuse

To Use Brownie-mix: Starting the Brownie Template

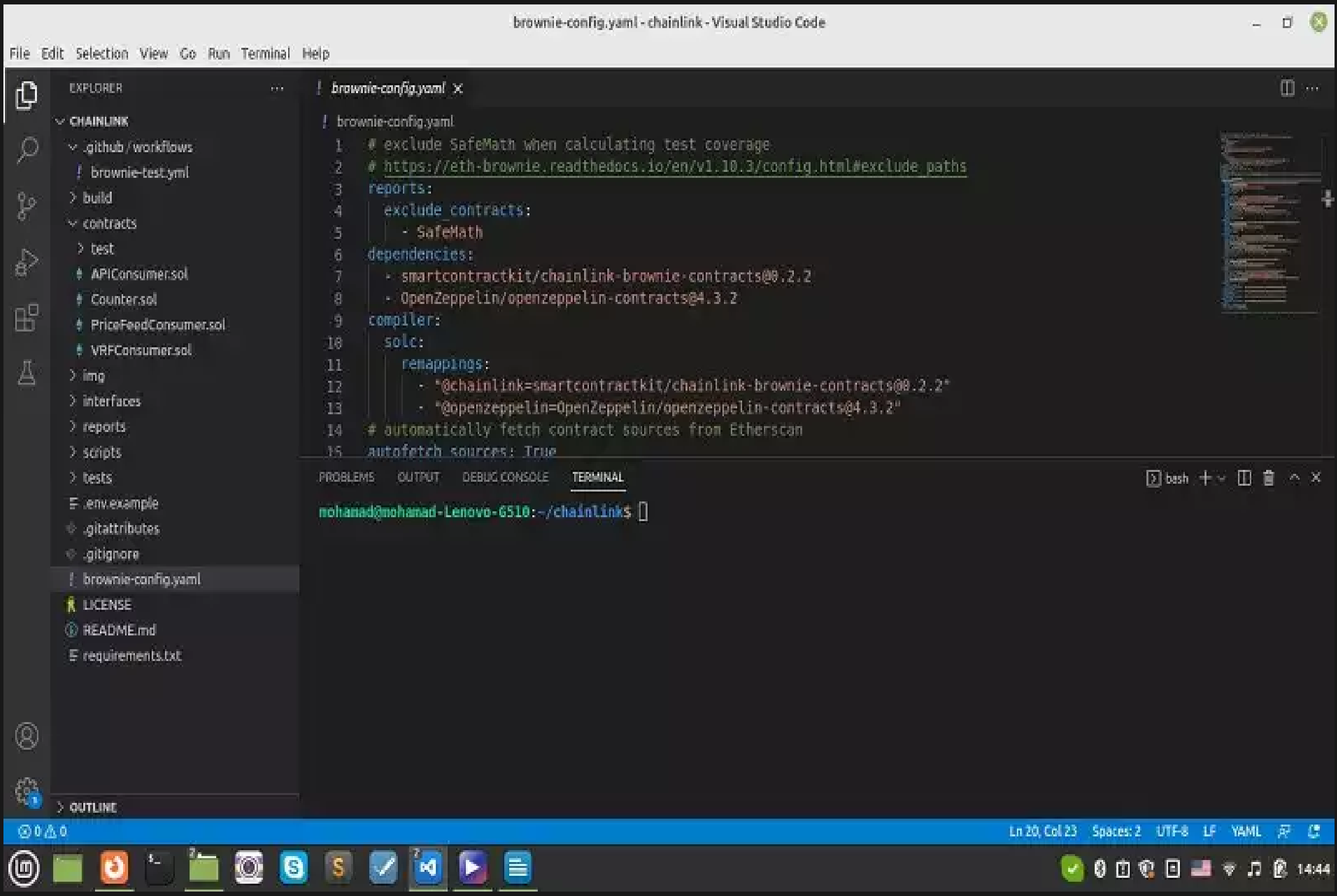
In order to use Brownie-mix boilerplates, we should first write the following command in the terminal:

```
brownie bake
```

In our case, for example, we want chainlink-mix (using which we have managed to write and deploy our contracts, so far).

```
brownie bake chainlink-mix
cd chainlink
```

And you will see that all the folders and files alongside all the necessary routine codes are provided for you in your directory called chainlink.



```
# exclude SafeMath when calculating test coverage
# https://eth-brownie.readthedocs.io/en/v1.10.3/config.html#exclude_paths
reports:
  exclude_contracts:
    - SafeMath
dependencies:
  - smartcontractkit/chainlink-brownie-contracts@0.2.2
  - OpenZeppelin/openzeppelin-contracts@4.3.2
compiler:
  solc:
    remappings:
      - "@chainlink=smartcontractkit/chainlink-brownie-contracts@0.2.2"
      - "@openzeppelin=OpenZeppelin/openzeppelin-contracts@4.3.2"
# automatically fetch contract sources from Etherscan
autofetch_sources: True
```

```
mohamad@mohamad-Lenovo-G510:~/chainlink$
```

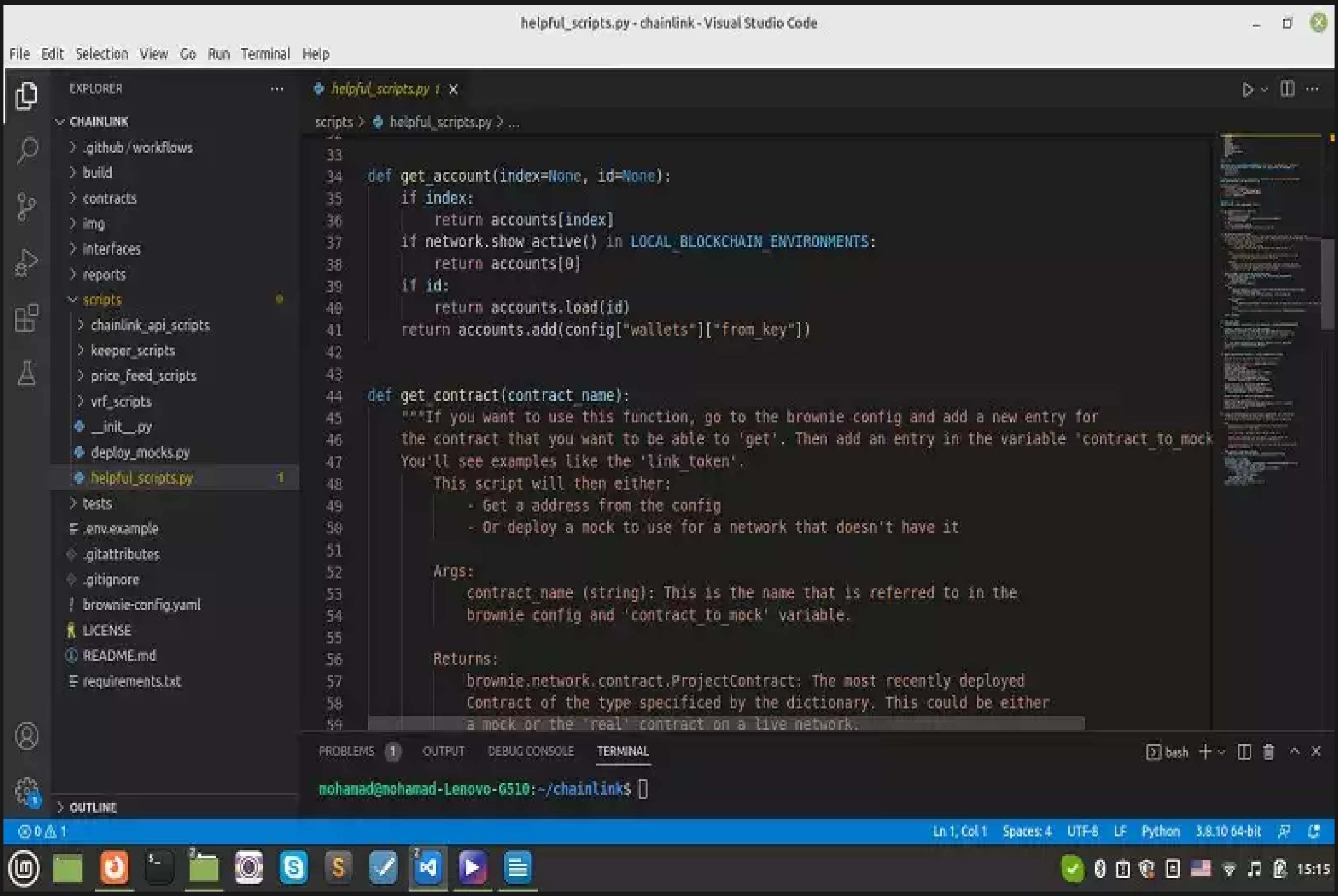
Modifying brownie_config.yaml

Notice that in the brownie-config.yaml, you should modify some parts:

```
# exclude SafeMath when calculating test coverage
# https:
//eth-brownie.readthedocs.io/en/v1.10.3/config.html#exclude_paths
reports:
 exclude_contracts:
   - SafeMath
dependencies:
 - smartcontractkit/chainlink-brownie-contracts@.2.2
 - OpenZeppelin/openzeppelin-contracts@.3.2
compiler:
 solc:
  remappings:
  "@chainlink=smartcontractkit/chainlink-brownie-contracts@0.2.2"
  "@openzeppelin=OpenZeppelin/openzeppelin-contracts@4.3.2"
# automatically fetch contract sources from Etherscan
autofetch_sources: True
# Uncomment to use the .env file
# dotenv: .env
# set a custom mnemonic for the development network
networks:
   default: development
   development:
  keyhash:
"0x6c3699283bda56ad74f6b855546325b68d482e983852a7a82979cc4807b641f4"
  fee:100000000000000000
  jobId:"29fa9aa13bf1468788b7cc4a500a45b8"
  update_interval:60
  verify:False
   kovan:
 vrf_coordinator:"0xdD3782915140c8f3b190B5D67eAc6dc5760C46E9"
 link_token:"0xa36085F69e2889c224210F603D836748e7dC0088"
 keyhash:
"0x6c3699283bda56ad74f6b855546325b68d482e983852a7a82979cc4807b641f4"
 fee:100000000000000000
 oracle:"0xc57b33452b4f7bb189bb5afae9cc4aba1f7a4fd8"
 jobId:"d5270d1c311941d0b08bead21fea7747"
 eth_usd_price_feed:"0x9326BFA02ADD2366b30bacB125260Af641031331"
# Change to True if you have an Etherscan API key and want to verify
 verify:True
 update_interval:60
   ganache:
 keyhash:
"0x6c3699283bda56ad74f6b855546325b68d482e983852a7a82979cc4807b641f4"
 fee:100000000000000000
 jobId:"29fa9aa13bf1468788b7cc4a500a45b8"
 update_interval:60
 verify:False
 rinkeby:
 vrf_coordinator:"0xb3dCcb4Cf7a26f6cf6B120Cf5A73875B7BBc655B"
 link_token:"0x01be23585060835e02b77ef475b0cc51aa1e0709"
 keyhash:
"0x2ed0feb3e7fd2022120aa84fab1945545a9f2ffc9076fd6156fa96eaff4c1311"
 fee:100000000000000000
 oracle:"0xc57b33452b4f7bb189bb5afae9cc4aba1f7a4fd8"
 jobId:"6b88e0402e5d415eb946e528b8e0c7ba"
 eth_usd_price_feed:"0x8A753747A1Fa494EC906cE90E9f37563A8AF630e"
# Change to True if you have an Etherscan API key and want to verify
 verify:False
   fuji:
 link_token:"0x0b9d5D9136855f6FEc3c0993feE6E9CE8a297846"
 fee:100000000000000000
 oracle:"0xcc80934eaf22b2c8dbf7a69e8e0d356a7cac5754"
 jobId:"5ca4fa9b2d64462290abfbda84e38cf4"
   mumbai:
 eth_usd_price_feed:"0x0715A7794a1dc8e42615F059dD6e406A6594651A"
 link_token:"0x326C977E6efc84E512bB9C30f76E30c160eD06FB"
 vrf_coordinator:"0x8C7382F9D8f56b33781fE506E897a4F1e2d17255"
 keyhash:
```

```
"0x6e75b569a01ef56d18cab6a8e71e6600d6ce853834d4a5748b720d06f878b3a4"
 fee:100000000000000000
   binance:
# link_token: ??
 eth_usd_price_feed:"0x9ef1B8c0E4F7dc8bF5719Ea496883DC6401d5b2e"
   binance-fork:
 eth_usd_price_feed:"0x9ef1B8c0E4F7dc8bF5719Ea496883DC6401d5b2e"
   mainnet-fork:
 eth_usd_price_feed:"0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419"
   matic-fork:
 eth_usd_price_feed:"0xF9680D99D6C9589e2a93a78A04A279e509205945"
wallets:
 from_key:${PRIVATE_KEY}
 from_mnemonic:${MNEMONIC}
```

Notice that if you use mnemonic, you should use accounts.from_mnemonic to be able to use from_mnemonic and if you use the private key, you should use accounts.add instead.
Also, make sure to uncomment the dotenv: .env if you want to keep your private data somewhere safe.
By using the above .yaml file, you can use any networks that you want and be sure that there is nothing else needed to add to this file. In the contracts folder, you will also be able to see some useful contracts that work as a dependency.
The most important folder is the scripts, inside which we have helpful_scripts.py, deploy_mocks.py, and some useful scripts that will help using them in your main deploy.py file.



```
def get_account(index=None, id=None):
    if index:
        return accounts[index]
    if network.show_active() in LOCAL_BLOCKCHAIN_ENVIRONMENTS:
        return accounts[0]
    if id:
        return accounts.load(id)
    return accounts.add(config["wallets"]["from_key"])


def get_contract(contract_name):
    """If you want to use this function, go to the brownie config and add a new entry for
    the contract that you want to be able to 'get'. Then add an entry in the variable 'contract_to_mock
    You'll see examples like the 'link_token'.
        This script will then either:
            - Get a address from the config
            - Or deploy a mock to use for a network that doesn't have it

        Args:
            contract_name (string): This is the name that is referred to in the
            brownie config and 'contract_to_mock' variable.

        Returns:
            brownie.network.contract.ProjectContract: The most recently deployed
            Contract of the type specificed by the dictionary. This could be either
            a mock or the 'real' contract on a live network.
```

```
mohamad@mohamad-Lenovo-G510:~/chainlink$
```

As you can see, some of the important and useful functions of useful_scripts.py are as follows:

```
get_account

get_contract

fund_with_link

deploy_mocks
```

Using the above functions, you can write your test files and deploy.py file much easier.

Final Word on Brownie-mix

In this article, we have introduced Brownie-mixes as a boilerplate (template) for blockchain projects and provided guides on how to quickly modify them so that you can run your desired project instantly. These projects could be creating an ERC-20 token, NFT, Aave protocol, Chainlink, and so on.

Join Arashtad Community

Follow Arashtad on Social Media

We provide variety of content, products, services, tools, tutorials, etc. Each social profile according to its features and purpose can cover only one or few parts of our updates. We can not upload our videos on SoundCloud or provide our eBooks on Youtube. So, for not missing any high quality original content that we provide on various social networks, make sure you follow us on as many social networks as you're active in. You can find out Arashtad's profiles on different social media services.

Get Even Closer!

Did you know that only one universal Arashtad account makes you able to log into all Arashtad network at once? Creating an Arashtad account is free. Why not to try it? Also, we have regular updates on our newsletter and feed entries. Use all these benefitial free features to get more involved with the community and enjoy the many products, services, tools, tutorials, etc. that we provide frequently.

SIGN UP NEWSLETTER RSS FEED