ethpm-cli Documentation

Release 0.3.0

The Ethereum Foundation

Oct 29, 2020

Contents

CLI for ethPM

For more information about the Ethereum Package Manager, check out the ethPM Docs.

CHAPTER 1

Contents

1.1 Quickstart

The easiest way to get started with ethPM.

1.1.1 Installation

```
brew update
brew upgrade
brew tap ethpm/ethpm-cli
brew install ethpm-cli
```

Or in a fresh Python virtual environment.

```
pip install ethpm-cli
```

More installation details and options

1.1.2 Setting your environment

Infura (required)

ethPM currently uses Infura to talk to the blockchain, so you must provide an Infura API key to authenticate your requests. It's free and simple to sign up for a key. You can sign up for a key here and then use the following command to set your key as the environment variable `WEB3_INFURA_PROJECT_ID`.

```
export WEB3_INFURA_PROJECT_ID=abc123...xyz890
```

Wallet account (optional)

Setting a wallet account authentication is necessary to use commands that sign a transaction (i.e. deploying an ethPM registry, releasing an ethPM package to a registry). ethPM uses eth-keyfile to interact with your encrypted private keyfile and sign transactions. Follow the steps in the README to generate your keyfile, and then use the following command to link your keyfile to ethPM.

```
ethpm auth --keyfile-path KEYFILE_PATH
```

Test that your keyfile has been properly stored.

```
ethpm auth
> Keyfile stored for address: 0x123abc.....................
```

You can now use your keyfile's password with the flag `--keyfile-password` for any ethPM command, and it will be used to automatically sign any transactions.

1.1.3 Activate

The activate command is the simplest way to start interacting with ethPM. First, find a REGISTRY_URI for the package you want to activate, some popular registries can be found here. Then use the following command to "activate" the package using its ethpm URI. An IPython console will pop up in your terminal, automatically populated with all contract factories, deployments, and web3 instances, ready to use!

```
ethpm activate ethpm://packages.ethpm.eth/package@1.0.0
```

If you have authenticated a wallet account with `ethpm auth`, pass your password in with the `--keyfile-password` flag to automatically configure all contract factories, deployments and web3 instances to sign for your account.

```
ethpm activate ethpm://packages.ethpm.eth/package@1.0.0 --keyfile-password xxx
```

To instantly interact with any verified contract on Etherscan, use an Etherscan URI (though, this will require you setting you Etherscan API key to the environment variable: `ETHPM_CLI_ETHERSCAN_API_KEY`).

```
ethpm activate etherscan://0x123v3r1f13dc0ntractaddr3ss890:1
```

1.1.4 Install

The install command will install any ethPM package to a local `_ethpm_packages/` directory. Think of this directory like `node_modules/` in npm. The files are written to disk according to this scheme. By default, `ethpm install` will look for an `_ethpm_packages/` in the current working directory, but a specific `_ethpm_packages/` directory can be targeted if you pass in its path with the `--ethpm-dir` flag. If you want to install a package under an alias, you can use the `--alias` flag to do so. If you're installing an etherscan verified contract as a package, you **must** pass in `--package-name` and `--package-version` flags.

```
ethpm install ethpm://packages.ethpm.eth/package_name@1.0.0
```

List all installed packages.

```
ethpm list
```

Uninstall a package.

```
ethpm uninstall package_name
```

1.1.5 Create

To create your own ethPM package from local contracts requires compilation. If you don't have the Solidity Compiler installed on your machine, there are frameworks available to help with the compilation and automatically generate your ethPM package.

If you have the Solidity compiler installed on your machine, the best way to get started is with the manifest wizard. The wizard expects a project directory with the following structure.

- project/ - contracts/
  - xxx.sol
  - yyy.sol

Pass in a path to your project directory under the `--project-dir` flag. The wizard will attempt to compile these contracts using the available `solc` on your machine. The available `solc` version on your machine must be sufficient for compiling the project contracts. After compiliation, the CLI will start the manifest wizard for complete package details.

```
ethpm create wizard --project-dir /path/to/project
```

1.1.6 Registry

ethPM packages are recorded on-chain using package registries. There is no central registry, and everybody who wants to release a package needs to deploy a registry on which they control what packages are released. In the CLI, there is a registry store to manage the different registies that you choose to interact with. If you want to store a registry under an alias, you can use the `--alias` flag to do so

```
ethpm registry list
```

```
ethpm registry add ethpm://ens.ethpm.eth:1 --alias my_favorite_registry
```

```
ethpm registry remove [URI_OR_ALIAS]
```

Active registries are used as the de-facto registry to release an ethPM package to. You can change the active registry with the following command.

```
ethpm registry activate [URI_OR_ALIAS]
```

1.1.7 Deploy

To deploy your own package registry, the following command is available. {link to code} This requires authentication via `ethpm auth`. Once deployed, you can check out your fresh registry on the ethPM explorer.

```
ethpm registry deploy --alias my_favorite_registry chain-id 1 --keyfile-password xxx
```

1.1.8 Release

To release a package to a registry is simple with the cli. First, make sure that the registry you want to release on is the active registry. You can confirm this with the `ethpm registry list` command.

```
ethpm release --package-name my_pkg --version 1.0.0 --manifest-uri ipfs://Qm... --
↪keyfile-password xxx
```

Now your brilliant smart contract ideas are available for the world to use!

1.2 Commands

A command-line tool to help manage ethPM packages and registries.

Warning: `ethPM CLI` is currently in public Alpha:

- It is expected to have bugs and is not meant to be used in production
- Things may be ridiculously slow or not work at all

1.2.1 ethpm activate

Command to help activate packages in your terminal.

```
usage: ethpm activate [-h] [--ethpm-dir ETHPM_DIR]
                      [--keyfile-password KEYFILE_PASSWORD]
                      package_or_uri
```

Positional Arguments

package_or_uri Installed package or URI of package to activate.

Named Arguments

--ethpm-dir Path to specific ethPM directory (Defaults to `./_ethpm_packages`).

--keyfile-password Password to local encrypted keyfile.

1.2.2 ethpm create

Commands to help generate manifests for local smart contracts.

```
usage: ethpm create [-h] {basic,solc-input,wizard} ...
```

Positional Arguments

create Possible choices: basic, solc-input, wizard

Sub-commands:

basic

Automatically generate a basic manifest for given projects dir. The generated manifest will package up all available sources and contract types available in the solidity compiler output found in given project directory.

```
ethpm create basic [-h] [--package-name PACKAGE_NAME]
                   [--package-version PACKAGE_VERSION]
                   [--project-dir PROJECT_DIR]
```

Named Arguments

--package-name	Package name for generating manifest with *basic-manifest* command.
--package-version	Package version for generating manifest with *basic-manifest* command.
--project-dir	Path to specific project directory.

solc-input

Generate solidity compiler standard json input for given project directory.

```
ethpm create solc-input [-h] [--project-dir PROJECT_DIR]
```

Named Arguments

--project-dir	Path to specific project directory.

wizard

Start CLI wizard for building custom manifests from the solidity compiler output found in given project directory.

```
ethpm create wizard [-h] [--manifest-path MANIFEST_PATH]
                    [--project-dir PROJECT_DIR]
```

Named Arguments

--manifest-path	Path of target manifest to amend.
--project-dir	Path to specific project directory.

1.2.3 ethpm install

Install an ethPM package to a local `_ethpm_packages` directory.

```
usage: ethpm install [-h] [--local-ipfs] [--package-name PACKAGE_NAME]
                     [--package-version PACKAGE_VERSION] [--alias ALIAS]
                     [--ethpm-dir ETHPM_DIR]
                     uri
```

Positional Arguments

uri IPFS / Github / Etherscan / Registry URI of target package.

Named Arguments

--local-ipfs Flag to use locally running IPFS node rather than defualting to Infura.

Default: False

--package-name Package name to use when installing a package from etherscan URIs.

--package-version Package version to use when installing a package from etherscan URIs.

--alias Alias to use in reference to this target registry / package.

--ethpm-dir Path to specific ethPM directory (Defaults to `./_ethpm_packages`).

1.2.4 ethpm list

List all installed ethPM packages in a local `_ethpm_packages` directory.

```
usage: ethpm list [-h] [--ethpm-dir ETHPM_DIR]
```

Named Arguments

--ethpm-dir Path to specific ethPM directory (Defaults to `./_ethpm_packages`).

1.2.5 ethpm update

Update the version of an installed ethPM package from a local `_ethpm_packages` directory. Since ethPM does not enforce semver - this command will look for all available versions of the package on the active registry, and prompt you to choose the version to install.

```
usage: ethpm update [-h] [--ethpm-dir ETHPM_DIR] package
```

Positional Arguments

package Package name / alias of target package to update.

Named Arguments

--ethpm-dir Path to specific ethPM directory (Defaults to `./_ethpm_packages`).

1.2.6 ethpm uninstall

Uninstall an ethPM package from a local `_ethpm_packages` directory.

```
usage: ethpm uninstall [-h] [--ethpm-dir ETHPM_DIR] package
```

Positional Arguments

package Package name / alias of target package to uninstall.

Named Arguments

--ethpm-dir Path to specific ethPM directory (Defaults to `./_ethpm_packages`).

1.2.7 ethpm release

Release a package on the currently active registry. Requires an active registry set via `ethpm registry` and authentication for tx signing set via `ethpm auth`.

```
usage: ethpm release [-h] [--manifest-uri MANIFEST_URI]
                     [--manifest-path MANIFEST_PATH]
                     [--package-name PACKAGE_NAME]
                     [--package-version PACKAGE_VERSION]
                     [--keyfile-password KEYFILE_PASSWORD]
```

Named Arguments

--manifest-uri Content addressed URI at which the manifest for released package is located.

--manifest-path Local path to target manifest used for release.

--package-name Package name of package you want to release. Must match *package_name* in manifest.

--package-version Version of package you want to release. Must match the *version* field in manifest.

--keyfile-password Password to local encrypted keyfile.

1.2.8 ethpm registry

Commands to help manage your local registry store.

```
usage: ethpm registry [-h] {deploy,list,add,remove,activate,explore} ...
```

Positional Arguments

registry Possible choices: deploy, list, add, remove, activate, explore

Sub-commands:

deploy

Deploy a new ERC1319 registry on the chain associated with provided chain ID.

```
ethpm registry deploy [-h] [--alias ALIAS] [--chain-id CHAIN_ID]
                      [--keyfile-password KEYFILE_PASSWORD]
```

Named Arguments

--alias	Alias to use in reference to this target registry / package.
--chain-id	Chain ID of target blockchain.
--keyfile-password	Password to local encrypted keyfile.

list

List all of the available registries in registry store.

```
ethpm registry list [-h]
```

add

Add a registry to registry store.

```
ethpm registry add [-h] [--alias ALIAS] uri
```

Positional Arguments

uri	Registry URI for target registry.

Named Arguments

--alias	Alias to use in reference to this target registry / package.

remove

Remove a registry from the registry store.

```
ethpm registry remove [-h] uri_or_alias
```

Positional Arguments

uri_or_alias	Registry URI or alias for registry to remove.

activate

Activate a registry to be used as the default registry for releasing new packages.

```
ethpm registry activate [-h] uri_or_alias
```

Positional Arguments

uri_or_alias Registry URI or alias for target registry.

explore

Explore a registry's list of released packages and manifest uris.

```
ethpm registry explore [-h] uri_or_alias
```

Positional Arguments

uri_or_alias Registry URI for target registry.

1.2.9 ethpm auth

Link a keyfile to authorize on-chain transactions (i.e. deploying a registry / releasing a package). To generate a keyfile, use eth-keyfile.

```
# Example script to generate your own keyfile
import json
from pathlib import Path
from eth_keyfile import create_keyfile_json

keyfile_json = create_keyfile_json(
   private_key = b"11111111111111111111111111111111",  # A bytestring of length 32
   password = b"foo"  # A bytestring which will be the password that can be used to␣
↪decrypt the resulting keyfile.
)
keyfile_path = Path.cwd() / 'keyfile.json'
keyfile_path.touch()
keyfile_path.write_text(json.dumps(keyfile_json))
```

```
usage: ethpm auth [-h] [--keyfile-path KEYFILE_PATH]
```

Named Arguments

--keyfile-path Path to the keyfile you want to set as default.

1.2.10 ethpm scrape

Scrape a blockchain for all IPFS data associated with any package release. This command will scrape for all `VersionRelease` events (as specified in ERC 1319). It will lookup all associated IPFS assets with that package, and write them to your ethPM XDG directory.

```
usage: ethpm scrape [-h] [--ipfs-dir IPFS_DIR] [--start-block START_BLOCK]
                    [--chain-id CHAIN_ID]
```

Named Arguments

--ipfs-dir	Path to specific IPFS directory.
--start-block	Block number to begin scraping from (defaults to blocks from ~ March 14, 2019).
--chain-id	Chain ID of target blockchain.

1.3 Installation

1.3.1 Pypi

- Create your virtual environment
- `pip install ethpm-cli`

1.3.2 Docker

- `docker pull ethpm/ethpm:latest`
- `docker run ethpm/ethpm:latest`

1.3.3 Homebrew (recommended)

- `brew update`
- `brew upgrade`
- `brew tap ethpm/ethpm-cli`
- `brew install ethpm-cli`

1.3.4 Setting up your environment vars

Before you can use ethPM CLI, you must provide an API key to interact with Infura. If you don't have an API key, you can sign up for one here. Then set your environment variable with `export WEB3_INFURA_PROJECT_ID="INSERT_KEY_HERE"`

If you plan to generate packages from Etherscan verified contracts, you must also provide an API key for Etherscan. `export ETHPM_CLI_ETHERSCAN_API_KEY="INSERT_KEY_HERE"`

If you're using Docker to run ethPM CLI, you must pass Docker the environment variables and mount volumes, like so...

```
docker run -i -e WEB3_INFURA_PROJECT_ID="INSERT_KEY_HERE" -v '/absolute/path/to/ethpm-
↪cli/:/absolute/path/to/ethpm-cli/' -v '/$HOME/.local/share/ethpmcli/:/root/.local/
↪share/ethpmcli/' ethpm/ethpm:latest list
```

1.3.5 Setting up your private key

If you plan to use the CLI to send any transactions over an Ethereum network (eg. deploying a new registry, releasing a package to a registry), you must link a private key keyfile to sign these transactions. ethPM CLI uses eth-keyfile to handle private keys. Follow the steps in the README to generate your encrypted keyfile. Make sure you don't lose the password, as you'll need to provide for any tx-signing commands. Once you have your encrypted keyfile, you can link it to the ethPM CLI with the following command.

```
ethpm auth --keyfile-path KEYFILE_PATH
```

1.4 Creating an ethPM manifest

ethPM CLI offers a couple options for creating your own ethPM manifest for local smart contracts. All options expect a project directory in the following format.

- **project_name/**
  - `solc_input.json`
  - `solc_output.json`
  - **contracts/**
    - `ContractA.sol`
    - `ContractB.sol`

In order to create a manifest, the CLI starts with your project's solidity compiler output. Use the following steps to generate `solc_output.json` for your project.

1.4.1 Generate the solidity compiler input

To generate your project's solidity compiler output, the solidity compiler needs a JSON input to know which contracts to compile. If you don't want to create your own `solc_input.json`, you can use the following command which will automatically generate the `solc_input.json` for all contracts found in your project's `contracts/` directory. However, if any of your contracts require special behavior, such as remappings, you will have to manually edit the generated `solc_input.json` as necessary.

```
ethpm create solc-input --project-dir
```

1.4.2 Generate the solidity compiler output

To generate the solidity compiler output for your project, you must have the appropriate solidity compiler version installed locally, or you can use the docker image. For more information on how to use the solidity compiler, read this.

Example..

```
solc --standard-json --allow-paths [path/to/project_dir] < [path/to/project_dir/solc_
↪input.json] > [path/to/project_dir/solc_output.json]
```

1.4.3 Creating your ethPM manifest

Now that you have the solidity compiler output for your project, there are two options for creating an ethPM manifest.

Manifest Wizard

The most comprehensive option for generating a manifest is the manifest wizard. The wizard will walk you through the steps and the available options to customize your manifest. Finally, it will write the generated manifest to your project's directory in the format `[package_version].json`.

```
ethpm create wizard --project-dir
```

Basic Manifest

If you want a quick and easy option to generate a valid manifest for your project, you can use the `basic-manifest` command. This will automatically package up all available sources and contract types found in your project's `solc_output.json`, and create a manifest with the provided `--package-name` and `--package-version`. Finally, it will write the generated manifest to your project's directory in the format `[package_version].json`.

```
ethpm create basic-manifest --project-dir /my_project --package-name my-package --
↪package-version 1.0.0
```

1.5 Disk Format

1.5.1 _ethpm_packages/

A user can have one or many different `_ethpm_packages/` local directories. Think of it like the `node_modules/` directory in Node or a virtual environment in Python.

- By default, `ethpm-cli` will target the `./_ethpm_packages/` directory available under the current working directory.
- If `--ethpm-dir` flag is specified on a cli command, the cli will target the provided directory.
- If the environmnet variable `ETHPM_CLI_PACKAGES_DIR` is set, the cli will use this directory if one is not specified using the `--ethpm-dir` flag.

ethPM cli writes ethPM package assets to your disk using the following format.

- `.cwd/` (current working directory)
  - `_ethpm_packages/`
    - `ethpm.lock`
    - `package_name/`
      - `manifest.json`
      - `_ethpm_packages/` (build dependencies if present in manifest)
      - `_src/`
      - Resolved source tree

1.5.2 ethpm.lock

A root-level JSON lockfile that manages what packages are currently installed. Everytime a package is installed or uninstalled, `ethpm.lock` must be updated with the corresponding package information.

- `installed_package_name`
  - `alias`
    - Package alias, if one is used to install pacakge, else package name.
  - `registry_address`
    - If package is installed with a registry URI, else `null`.
  - `resolved_content_hash`
    - Validation hash of fetched contents. If re-generated, **MUST** match hash of given URI.
  - `resolved_package_name`
    - `"package_name"` resolved from target manifest.
  - `resolved_uri`
    - Content-addressed URI of manifest.
  - `resolved_version`
    - `"version"` resolved from target manifest.
  - `install_uri`
    - Content addressed / etherscan / registry URI used to install package.

1.5.3 ethPM XDG

For storing IPFS assets and the registry config file, ethPM-CLI uses the XDG Base Directory Specification https://specifications.freedesktop.org/basedir-spec/basedir-spec-0.6.html. These files are written to `$XDG_DATA_HOME / 'ethpmcli'`. A user will only have one local ethPM XDG directory.

1.6 URIs

ethPM Cli supports the following URI schemes.

- IPFS
  - `ipfs://[IPFS_HASH]`
- Etherscan
  - `etherscan://[CONTRACT_ADDRESS]:[CHAIN_ID]`
  - `CONTRACT_ADDRESS` and `CHAIN_ID` must represent a Verified Contract on Etherscan.
  - `CONTRACT_ADDRESS` **MUST** be a valid, checksummed address.
  - Supported values for `CHAIN_ID`

CHAIN_ID	CHAIN
1	Mainnet
3	Ropsten
4	Rinkeby
5	Goerli
42	Kovan

- Github Blob
  - `https://api.github.com/repos/[OWNER]/[REPO]/git/blobs/[FILE_SHA]`

1.7 Release Notes

1.7.1 v0.1.0-alpha.1

- Launched repository, claimed names for pip, RTD, github, etc

CHAPTER 2

Indices and tables

- genindex