ether-py Documentation

Release 2021.3.0

Dave Dittrich

Jun 16, 2021

CONTENTS:

This document (version 2021.3.0) describes the ether-py Ethereum command line interface (`ether-py` for short).

CHAPTER

ONE

INTRODUCTION

This chapter introduces the ether-py Ethereum command line interface (`ether-py` for short).

This CLI was inspired, in part, by the article Creating a Python Ethereum Interface: Part 1 for similar reasons. I not only wanted to do the same things, but to take advantage of the rich feature set provided by cliff for more output control, built-in command help, modularity and extensibility.

1.1 Features

- `ether-py` provides a general Python command line interface (CLI) built on the OpenStack cliff – Command Line Interface Formulation Framework.
- `cliff` provides many useful features like modularizing subcommands into groups, built-in help for internally documenting commands, and producing output in clean tabular form or in one of several data formats you can feed into other tools or automation platforms.
- Sphinx documentation for generation with ReadTheDocs including `cliff` autoprogram Sphinx integration for documenting commands from the same `--help` output you can get at the command line.
- Uses the python_secrets package (`psec`) to manage endpoint configuration settings and access control tokens to prevent secrets leakage and to make it easy to switch between local development/testing using ganache and accessing a live Ethereum blockchain using Infura endpoints.
- Uses py-solc-x for installing Solidity compilers and compiling Solidity smart contracts (`.sol` files) dependabot.

1.2 Contact

Dave Dittrich <dave.dittrich@gmail.com>

1.3 Credits

This package was created with Cookiecutter from the <https://github.com/davedittrich/cookiecutter-cliffapp-template> project template. It derives some of its features and inspiration from <https://github.com/veit/cookiecutter-namespace-template> and <https://github.com/audreyfeldroy/cookiecutter-pypackage>.

1.4 Related Projects

- https://pypi.org/project/web3/ (Web3.py)
- https://pypi.org/project/ethereum/ (Next generation cryptocurrency network)
- https://pypi.org/project/ethereum-tools/ (High-level tools and library to interact with Ethereum)
- https://pypi.org/project/ethereum2-etl/ (Tools for exporting Ethereum 2.0 blockchain data to CSV and JSON)
- https://pypi.org/project/pytest-ethereum/ (pytest-ethereum: Pytest library for ethereum projects.)
- https://pypi.org/project/ethereum-gasprice/ (Tool for fetching actual gasprice in ethereum blockchain)
- The Eth2 Upgrades: Upgrading Ethereum to radical new heights

1.5 Creating tokens and smart contracts

- Non-fungible tokens (NFTs)
- https://docs.ethhub.io/built-on-ethereum/erc-token-standards/erc721/

1.6 Other references

- Deep dive into Ethereum logs, by banteg, codeburst.io (Medium), Jan 4, 2018

CHAPTER TWO

USAGE

Subcommand groups in `ether-py` are divided by categories reflecting specific features, data sources, etc.

2.1 Getting help

To get help information on global command arguments and options, use the `help` command or `--help` option flag. The usage documentation below will detail help output for each command.

2.2 Formatters

The cliff – Command Line Interface Formulation Framework provides a set of formatting options that facilitate accessing and using stored secrets in other applications. Data can be passed directly in a structured format like CSV, or passed directly to programs like Ansible using JSON.

Attention: The formatter options are shown in the `--help` output for individual commands. For the purposes of this chapter, including the lengthy formatter options on every command would be quite repetitive and take up a lot of space. For this reason, the formatter options will be suppressed for commands as documented below. You can see the differences in this functional example.

2.3 Logging

Cliff also includes a mechanism for writing log output from the program to a user-specified file at runtime. This is useful for debugging, as well as for monitoring long-running actions.

Here is an example of logging output of the `about` command:

```
$ ether-py -vvvv --log-file logfile about
initialize_app
[+] command line: /usr/local/Caskroom/miniconda/base/envs/test/bin/ether-py -vvvv --log-
↪file logfile about
prepare_to_run_command About
ether-py version 2021.3.0rc1

This program was bootstrapped from a ``cookiecutter`` template created
by Dave Dittrich <dave.dittrich@gmail.com>:
```

(continues on next page)

(continued from previous page)

```
    https://github.com/davedittrich/cookiecutter-cliffapp-template.git
    https://cookiecutter-cliffapp-template.readthedocs.io

Author:    Dave Dittrich <dave.dittrich@gmail.com>
Copyright: 2021, Dave Dittrich. All rights reserved.
License:   Apache Software License 2.0
URL:       https://pypi.python.org/pypi/{{cookiecutter.project_name}}
[!] clean_up About
```

Here is what the output would look like:

```
$ cat logfile
[2021-06-08 14:31:57,050] DEBUG    ether-py initialize_app
[2021-06-08 14:31:57,051] INFO     ether-py [+] command line: /usr/local/Caskroom/
↪miniconda/base/envs/test/bin/ether-py -vvvv --log-file logfile about
[2021-06-08 14:31:57,052] DEBUG    ether-py prepare_to_run_command About
[2021-06-08 14:31:57,073] DEBUG    ether-py [!] clean_up About
```

2.4 Command groups

2.4.1 About

about

About the `ether-py` CLI

```
ether_py about [--readthedocs] [--browser BROWSER] [--force]
```

`--readthedocs`
Open a browser to the `ether-py` readthedocs page (default: False{}).

`--browser` <BROWSER>
Browser to use for viewing (default: None).

`--force`
Open the browser even if process has no TTY (default: False)

Shows information about the `ether-py` CLI.

```
$ ether-py about
ether-py version 0.1.dev25+g8f92cdc
```

It will also print out copyright and related information (which isn't easy to force `autoprogram-cliff` to parse correctly in help output).

The `--readthedocs` option will open a browser to the `ether-py` documentation web page.

ABOUT THE BROWSER OPEN FEATURE

This program uses the Python `webbrowser` module to open a browser.

https://docs.python.org/3/library/webbrowser.html

This module supports a large set of browsers for various operating system distributions. It will attempt to chose an appropriate browser from operating system defaults. If it is not possible to open a graphical browser application, it may open the `lynx` text browser.

You can choose which browser `webbrowser` will open using the identifier from the set in the `webbrowser` documentation. Either specify the browser using the `--browser` option on the command line, or export the environment variable `BROWSER` set to the identifier (e.g., `export BROWSER=firefox`).

It is also possible to set the `BROWSER` environment variable to a full path to an executable to run. On Windows 10 with Windows Subsystem for Linux, you can use this feature to open a Windows executable outside of WSL. (E.g., using `export BROWSER='/c/Program Files/Mozilla Firefox/firefox.exe'` will open Firefox installed in that path).

Also note that when this program attempts to open a browser, an exception may be thrown if the process has no TTY. If this happens, use the `--force` option to bypass this behavior and attempt to open the browser anyway.

2.4.2 Block

block get

Get Ethereum block

```
ether_py block get BLOCK
```

BLOCK
Ethereum block number

Get an Ethereum block.

The block number should be the block's number, its hash, or the word "latest" to get the most recent block.

```
$ ether-py block get latest
ether-py                  ERROR    [!] NOT IMPLEMENTED
```

block show

Show Ethereum block

```
ether_py block show [--prefix PREFIX] BLOCK [FIELD]
```

--prefix <PREFIX>
add a prefix to all variable names

BLOCK
Ethereum block number

FIELD
Block metadata field

Get an Ethereum block.

The block number should be the block's number, its hash, or the word "latest" to get the most recent block.

```
$ ether-py block show latest --fit-width
+-------------------+--------------------------------------------------------------------------
↪----------------------------------------------------+
```

(continues on next page)

(continued from previous page)

```
| Field            | Value                                                              ␣
↪                                                     |
+------------------+--------------------------------------------------------------------
↪-----------------------------------------------------+
| number           | 15                                                                 ␣
↪                                                     |
| hash             | 0x008547e530fe0965d3711d25fbb1d20264c16d525f02aa280633c1a721ff5720 ␣
↪                                                     |
| parentHash       | 0xc95783fb95338588b8bffe4c88eb979086e0c6b6fdd1a78bc591319c3270906d ␣
↪                                                     |
| mixHash          | 0x0000000000000000000000000000000000000000000000000000000000000000 ␣
↪                                                     |
| nonce            | 0x0000000000000000                                                 ␣
↪                                                     |
| sha3Uncles       | 0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347 ␣
↪                                                     |
| logsBloom        |␣
↪0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪|
|                  |␣
↪00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪|
|                  |␣
↪00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪|
|                  |␣
↪00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪|
|                  | 00000000000000000000000000000000000000                             ␣
↪                                                     |
| transactionsRoot | 0x9f619679ac0f15e5725d63c06bf352f88a2eb002d7bbb983188a929c10f38de1 ␣
↪                                                     |
| stateRoot        | 0xeb0ae3b8c7beb461793f9780d4e180515843f60227768c3b19b0472216673ed6 ␣
↪                                                     |
| receiptsRoot     | 0xd5420f0d6143865fa94e3464abb47a054c4f83dd5b9159603616fe39b97dd2b2 ␣
↪                                                     |
| miner            | 0x0000000000000000000000000000000000000000                         ␣
↪                                                     |
| difficulty       | 0                                                                  ␣
↪                                                     |
| totalDifficulty  | 0                                                                  ␣
↪                                                     |
| extraData        | 0x                                                                 ␣
↪                                                     |
| size             | 1000                                                               ␣
↪                                                     |
| gasLimit         | 6721975                                                            ␣
↪                                                     |
| gasUsed          | 313249                                                             ␣
↪                                                     |
| timestamp        | 1618011641                                                         ␣
↪                                                     |
```

(continues on next page)

(continued from previous page)

```
| transactions     | [HexBytes(
↪'0x07a137a05974311c877874d5fd699d90adfeb4fca10c95d989285a504af39b2d')]         ␣
↪                          |
| uncles           | []                                                         ␣
↪                                                       |
+------------------+-------------------------------------------------------------------
↪----------------------------------------------------+
```

2.4.3 Contract

contract list

List contract files

```
ether_py contract list [--sort-ascending | --sort-descending] [NAME]
```

--sort-ascending
sort the column(s) in ascending order

--sort-descending
sort the column(s) in descending order

NAME
Solidity contract name

List Solidity contracts and related files.

Solidity contracts are compiled from source code in `.sol` files. By convention, `ether-py` expects the contact name to be the same as the `.sol` file without the extension (so `Greeter.sol` is the source file for the contract `Greeter`).

Only contracts with files having `.sol` extensions are listed. Initially, just those files exist and none of the related file types:

```
$ ether-py contract list
+-----------------------+-----+---------+----------+---------+
| name                  | abi | address | bytecode | receipt |
+-----------------------+-----+---------+----------+---------+
| Greeter               | No  | No      | No       | No      |
| SimpleCollectible     | No  | No      | No       | No      |
+-----------------------+-----+---------+----------+---------+
```

As contracts are compiled and loaded, additional files with file extensions in the following set are created:

{ `.abi`, `.address`, `.bytecode`, `.receipt` }

After compiling the contract, the `.abi` and `.bytecode` files will exist:

```
$ ether-py demo greeter compile
solcx                     INFO     Using solc version 0.7.6
[+] created /Users/dittrich/git/ether-py/contracts/Greeter.bytecode
[+] created /Users/dittrich/git/ether-py/contracts/Greeter.abi
$ ether-py contract list
+-----------------------+-----+---------+----------+---------+
| name                  | abi | address | bytecode | receipt |
```

(continues on next page)

(continued from previous page)

```
+----------------------+-----+---------+----------+---------+
| Greeter              | Yes | No      | Yes      | No      |
| SimpleCollectible    | No  | No      | No       | No      |
+----------------------+-----+---------+----------+---------+
```

After loading the contract into the Ethereum blockchain, the `.address` and `.receipt` files will exist:

```
$ ether-py demo greeter load
0x22785519732f4623B9D3096bE3bCDF47053cA035
[+] greeter says 'Hello'
$ ether-py contract list
+----------------------+-----+---------+----------+---------+
| name                 | abi | address | bytecode | receipt |
+----------------------+-----+---------+----------+---------+
| Greeter              | Yes | Yes     | Yes      | Yes     |
| SimpleCollectible    | No  | No      | No       | No      |
+----------------------+-----+---------+----------+---------+
```

You can limit the output to one or more contracts by naming them as arguments on the command line:

```
$ ether-py contract list Greeter
+---------+-----+---------+----------+---------+
| name    | abi | address | bytecode | receipt |
+---------+-----+---------+----------+---------+
| Greeter | Yes | Yes     | Yes      | Yes     |
+---------+-----+---------+----------+---------+
```

Use the `-v` option to show the path to the contracts directory.

contract show

Show Solidity contract files

```
ether_py contract show
    [--sort-ascending | --sort-descending]
    NAME
    [TYPE]
```

`--sort-ascending`
sort the column(s) in ascending order

`--sort-descending`
sort the column(s) in descending order

`NAME`
Solidity contract name

`TYPE`
Associated file type

List Solidity contract files.

2.4.4 Demo

demo Greeter call

Call `Greeter` contract

```
ether_py demo Greeter call MESSAGE
```

MESSAGE
New Greeter message

Call the Greeter contract with a message.

demo Greeter compile

Compile `Greeter` contract

```
ether_py demo Greeter compile [--solc-version VERSION]
```

--solc-version `VERSION`
Use solc compiler version (default: 'latest')

Compile the `Greeter` contract.

If no compatible compiler is installed, you will get a message showing the pragma specified in the `.sol` file:

```
$ ether-py demo Greeter compile
[-] No compatible solc version installed matching 'pragma solidity >=0.6.0 <0.8.0;': see
↪'ether-py solc install --help'
```

Identify a compatible version using `ether-py solc versions --installable` and install before trying again:

```
$ ether-py solc install  0.7.6
solcx                     INFO     Downloading from https://solc-bin.ethereum.org/macosx-
↪amd64/solc-macosx-amd64-v0.7.6+commit.7338295f
solcx                     INFO     solc 0.7.6 successfully installed at: /Users/dittrich/
↪.solcx/solc-v0.7.6
[+] installed solc version '0.7.6'
$ ether-py demo Greeter compile -v
initialize_app
[+] command line: ether-py demo Greeter compile -v
[+] established connection to ganache endpoint at http://127.0.0.1:7545
solcx                     INFO     Using solc version 0.7.6
[+] created /Users/dittrich/git/ether-py/contracts/Greeter.bytecode
[+] created /Users/dittrich/git/ether-py/contracts/Greeter.abi
```

demo Greeter load

Load Greeter contract

```
ether_py demo Greeter load
```

Saves the Greeter contract to the ethereum blockchain.

```
$ ether-py demo -v Greeter load
initialize_app
[+] command line: ether-py demo -v Greeter load
[+] established connection to ganache endpoint at http://127.0.0.1:7545
[+] transaction 0xF43Dd5d4f35D468c65B96901B93e8BCaD6F3C210 received
[+] Greeter says 'Hello'
```

2.4.5 Eth

eth send

Send Ethereum

```
ether_py eth send ETH
```

ETH
Transaction amount in eth

Send Ethereum from one address to another.

```
$ ether_py eth send FROM TO ETH
```

eth show

Show Ethereum blockchain information

```
ether_py eth show [--prefix PREFIX] [FIELD]
```

--prefix <PREFIX>
add a prefix to all variable names

FIELD
Blockchain metadata field

Shows attributes about Ethereum blockchain.

```
$ ether-py eth show
+-------------------+--------------------------------------------+
| Field             | Value                                      |
+-------------------+--------------------------------------------+
| block_number      | 15                                         |
| chain_id          | 1337                                       |
| coinbase          | 0xB9f74d880185873808D363f9295BBC91314B0759 |
| default_account   | None                                       |
| default_block     | latest                                     |
```

(continues on next page)

(continued from previous page)

```
| gas_price        | 20000000000                              |
| hashrate         | 0                                        |
| is_async         | False                                    |
| mining           | True                                     |
| protocol_version | 63                                       |
| syncing          | False                                    |
+------------------+------------------------------------------+
```

2.4.6 Net

net show

Show Ethereum net

```
ether_py net show [--prefix PREFIX] [FIELD]
```

--prefix <PREFIX>
 add a prefix to all variable names

FIELD
 Ethereum network metadata field

Shows attributes about Ethereum net.

```
$ ether_py net show
+------------+-------+
| Field      | Value |
+------------+-------+
| is_async   | False |
| listening  | True  |
| peer_count | 0     |
| version    | 5777  |
+------------+-------+
```

2.4.7 Solc

solc install

Install solc compiler version(s)

```
ether_py solc install VERSION [VERSION ...]
```

VERSION
 Solidity compiler version

Install one or more `solc` compiler versions.

Solidity smart contracts (`.sol` files) usually specify a particular `solc` compiler version, or a range of compatible versions, using a `pragma` statement that looks like this:

```
pragma solidity >=0.6.0 <0.8.0;
```

ether-py will extract the pragma statement and pass it along to solc when compiling the contract. If no compatible compiler can be found, you will get an error message that shows the pragma statement. Select a compiler version that matches the range from a list of installable solc versions shown by ether-py solc versions --installable (0.7.6 will work in this case.) You can then install it, and the most recent compiler version, like this:

```
$ ether-py solc install 0.7.6 latest
solcx                     INFO     Downloading from https://solc-bin.ethereum.org/macosx-
↪amd64/solc-macosx-amd64-v0.7.6+commit.7338295f
solcx                     INFO     solc 0.7.6 successfully installed at: /Users/dittrich/
↪.solcx/solc-v0.7.6
[+] installed solc version '0.7.6'
solcx                     INFO     Downloading from https://solc-bin.ethereum.org/macosx-
↪amd64/solc-macosx-amd64-v0.8.3+commit.8d00100c
solcx                     INFO     solc 0.8.3 successfully installed at: /Users/dittrich/
↪.solcx/solc-v0.8.3
[+] installed solc version 'latest'
$ ether-py solc versions
+---------+
| version |
+---------+
| 0.8.3   |
| 0.8.0   |
| 0.7.6   |
+---------+
```

solc remove

Remove solc compiler version(s)

```
ether_py solc remove VERSION [VERSION ...]
```

VERSION
Solidity compiler version

Remove one or more solc compiler versions.

Specify one or more compiler versions by their version number, by a substring (to select more than one version in a series), the word latest to remove the highest numbered version, or all to remove all versions.

```
$ ether-py solc versions 0.8
+---------+
| version |
+---------+
| 0.8.3   |
| 0.8.0   |
+---------+
$ ether-py solc remove 0.8.0
[+] removed /Users/dittrich/.solcx/solc-v0.8.0
$ ether-py solc versions
+---------+
| version |
+---------+
| 0.8.3   |
```

(continues on next page)

(continued from previous page)

```
| 0.7.6   |
+---------+
```

solc show

Show solc compiler information

```
ether_py solc show [--prefix PREFIX] [FIELD]
```

--prefix <PREFIX>
    add a prefix to all variable names

FIELD
    Solidity compiler metadata field

Show information about the active `solc` compiler.

```
$ ether-py solc show
+---------------------+-----------------------------------+
| Field               | Value                             |
+---------------------+-----------------------------------+
| active_version      | 0.8.0                             |
| active_version_hash | 0.8.0+commit.c7dfd78e             |
| executable          | /Users/dittrich/.solcx/solc-v0.8.0 |
| installed_versions  | 0.8.0,0.7.6                       |
+---------------------+-----------------------------------+
```

solc versions

Show solc compiler versions

```
ether_py solc versions
    [--sort-ascending | --sort-descending]
    [--installable]
    [VERSION]
```

--sort-ascending
    sort the column(s) in ascending order

--sort-descending
    sort the column(s) in descending order

--installable
    Show installable versions (default: False)

VERSION
    Solidity compiler version

Show `solc` compiler versions.

By default, you will be shown the list of `solc` compilers that are currently installed and available for use in compiling Solidity smart contract (`.sol` files).

```
$ ether-py solc versions
+---------+
| version |
+---------+
| 0.8.0   |
| 0.7.6   |
+---------+
```

To instead see a list of installable versions, use the `--installable` flag.

```
$ ether-py solc versions --installable
+---------+
| version |
+---------+
| 0.8.3   |
| 0.8.2   |
| 0.8.1   |
| 0.8.0   |
| 0.7.6   |
| 0.7.5   |
|  . . .  |
| 0.4.12  |
| 0.4.11  |
+---------+
```

To see a subset of versions, include an argument with the substring to match on:

```
$ ether-py solc versions 0.7 --installable
+---------+
| version |
+---------+
| 0.7.6   |
| 0.7.5   |
| 0.7.4   |
| 0.7.3   |
| 0.7.2   |
| 0.7.1   |
| 0.7.0   |
+---------+
```

See also `ether-py solc install --help`.

2.4.8 Tx

tx show

Show Ethereum transaction

```
ether_py tx show [--prefix PREFIX] TRANSACTION [FIELD]
```

--prefix <PREFIX>
    add a prefix to all variable names

TRANSACTION
: Transaction ID

FIELD
: Transaction metadata field

Show an Ethereum transaction (tx).

The transaction is identified by hash.

```
$ ether-py tx show 0xf357f2c33c3793ffaa2f4c98c22790d7b587aa30c3df4fdd65143a8a2a50d523
+------------------+--------------------------------------------------------------------+
| Field            | Value                                                              |
+------------------+--------------------------------------------------------------------+
| hash             | 0xf357f2c33c3793ffaa2f4c98c22790d7b587aa30c3df4fdd65143a8a2a50d523 |
| nonce            | 1                                                                  |
| blockHash        | 0x07df29f220a31c43cd2792a2effa3a1187eb6c66d71a6f2cbbff29c60c3270f6 |
| blockNumber      | 2                                                                  |
| transactionIndex | 0                                                                  |
| from             | 0xBe50e2b648e9A0e7E1e2B1b517C53cDAB6424355                         |
| to               | 0x3b4720e34496A6b2357045Bf129a40bCaC87B6e1                         |
| value            | 5000000000000000000                                                |
| gas              | 2000000                                                            |
| gasPrice         | 50000000000                                                        |
| input            | 0x                                                                 |
| v                | 27                                                                 |
| r                | 0x4b5a13f54f054ab12cd3f2b38c3d8b8de9cfc3cb7b431e7270135f00f7402510 |
| s                | 0x5dd02463f52bced1b990d1fc213e06930597a3fc5ef0e6b29156efe634a33748 |
+------------------+--------------------------------------------------------------------+
```

CHAPTER

THREE

DEVELOPMENT LIFECYCLE TASKS

This section covers tasks related to software development and release.

3.1 Development Testing

GitHub Actions are configured in the `.github/workflows/test-build-publish.yml` file to always run tests when you `push` to GitHub.

```
    - name: Run tests
      run: make test
```

As you can see, the command it runs is the same one you would run at the command line to test locally:

```
$ make test
```

Note: When you are about to make a release it is always a good idea to make *sure* that all tests pass *before* you bump the version number or tag a test release and then `push` to trigger the release.

At the minimum, get used to running the `pep8` Python syntax checks as you edit your code, as well as use a linter in your integrated development environment editor (such as VSCode). You can do this with:

```
$ tox -e pep8
```

It is easier to fix any `pep8` (or `bandit`) findings and commit working changes *before* you push than it is to deal with the GitHub Actions failure and repeat the tagging and/or version bumping.

3.2 Documentation

Documentation is written in ReStructured Text in files located in the `docs/` directory and rendered locally with Sphinx using the `make docs` command and remotely on ReadTheDocs.

Configuration settings for ReadTheDocs are found in the file `.readthedocs.yml`:

```
# .readthedocs.yml
# Read the Docs configuration file
# See https://docs.readthedocs.io/en/stable/config-file/v2.html for details

# Required
```

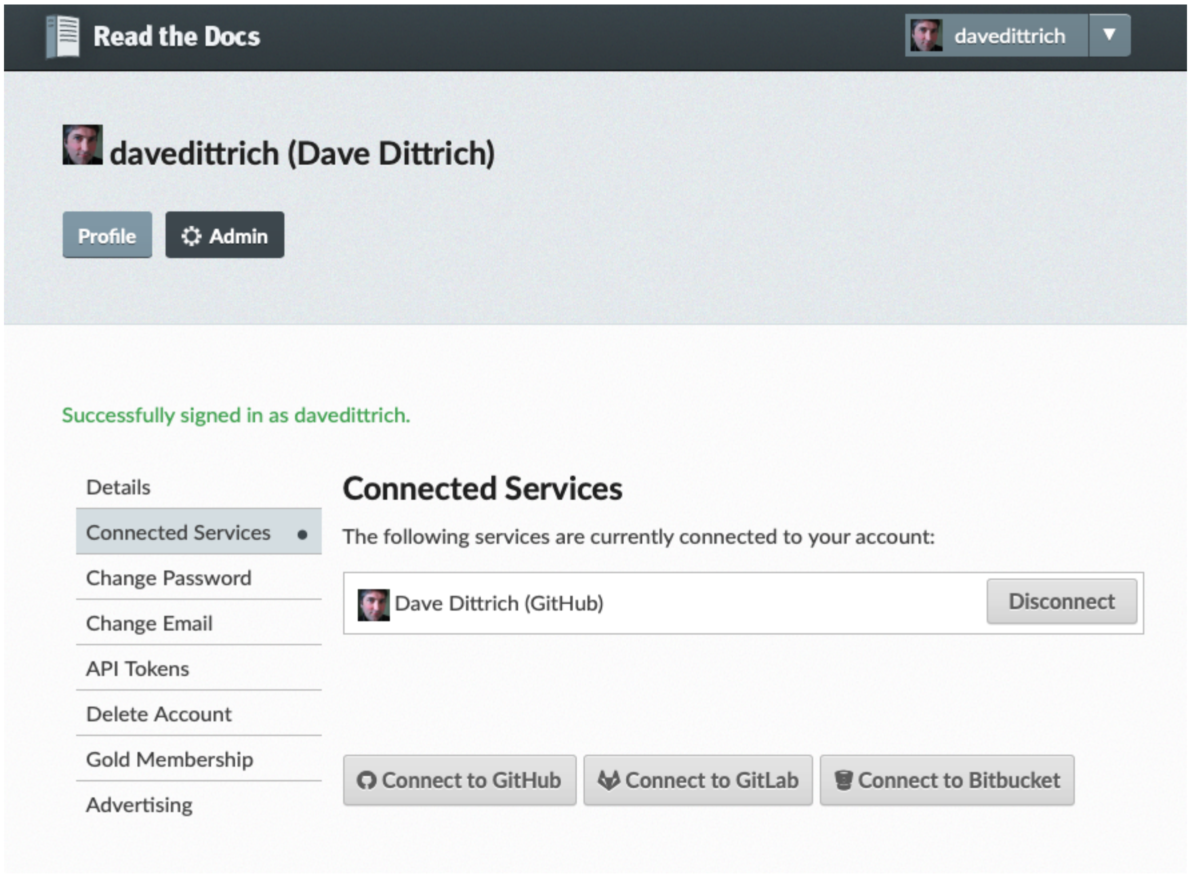
(continues on next page)

(continued from previous page)

```
version: 2

build:
  image: latest

python:
   version: 3.8
   install:
      - requirements: requirements.txt
      - method: pip
        path: .
        extra_requirements:
           - docs

# Build documentation in the docs/ directory with Sphinx
sphinx:
  configuration: docs/conf.py

formats:
  - pdf
```

Two things must be configured for new documentation to be generated when you push to GitHub.

1. A connection between your GitHub account and your ReadTheDocs account must be created. If one does not already exist, create it now.



Successfully signed in as davedittrich.

Connected Services

The following services are currently connected to your account:

Dave Dittrich (GitHub)

2. A webhook URL from ReadTheDocs must be created on ReadTheDocs and added to your GitHub repo.

   To create the URL, go to the ReadTheDocs project **Admin** page, select **Integrations** from the left hand menu, select **GitHub incoming webhook**, then copy the webhook URL.

   To add it to your project's GitHub repo, select the **Settings** tab, then select **Webhooks** from the left hand menu, then **Add Webhook**. Paste the webhook URL into the **Payload URL** dialog box (making sure the URL starts with `https://`) and finally clicking the **Add webook** button at the bottom of the page.

When a push to GitHub is made, the webhook is triggered signaling to ReadTheDocs to pull the most recent code and render the documentation.

3.3 Version numbering

This repository is set up to use date based, or calendar versioning, for release version numbers. Version numbers use the full **year** as the first (major) component, the **month** as the second (minor) component, and patch releases using the third (normal patch) component.

To illustrate how this works for development release candidate and full release version numbers, let's assume the last full release was made in March 2021 and there was only one release. The corresponding version number would be reflected in the `VERSION` file this way:

```
$ cat VERSION
2021.3.0
```

When you build the package with `setup.py`, or when the `ether-py` program wants to get the version number, they both use setuptools_scm.

Someone running `ether-py --version` after installing the program with `python -m pip install ether-py` will get a version number similar to the example above:

```
$ ether-py --version
ether-py 2021.3.0
```

When you are developing in a clone of the GitHub repo, the result will be a more precise version number that reflects the *next* patch number or release candidate number, including specifics about the state of the Git repo beyond the previous tag.

Let's say tags in the repo look like this:

```
$ git tag -l | grep v2021.3
v2021.3.0
v2021.3.1rc1
v2021.3.1rc2
v2021.3.1rc3
```

Given the last release candidate was `v2021.3.1rc3`, the final component of the *current* development version number would be `1rc4`. If the repo is sitting at ten commits past the last tag with the last commit having hash `g9a790a9`, the resulting version number as generated on the 29th of March would look like this:

```
$ python setup.py --version
2021.3.1rc4.dev10+g9a790a9.d20210329
```

The version number string exists in several files, which all need to be updated at the same time in a consistent manner. The program `bump2version` handles that task as configured in the file `setup.cfg`.

The contents of that file at the time this document was generated are:

```
[bumpversion]
current_version = 2021.3.0
commit = False
tag = False

[bumpversion:file:README.rst]

[bumpversion:file:VERSION]

[bumpversion:file:ether_py/__init__.py]

[bumpversion:file:docs/conf.py]
```

Warning: If you need to add a new file containing the version number, make sure to add the file path to this file.

3.4 Releasing on PyPI or Test PyPI

The GitHub Actions workflow file (`.github/workflows/test-build-publish.yml`) is set up to publish artifacts to PyPI or Test PyPi based on pushed tags. Here is the portion of the file that handles this task.

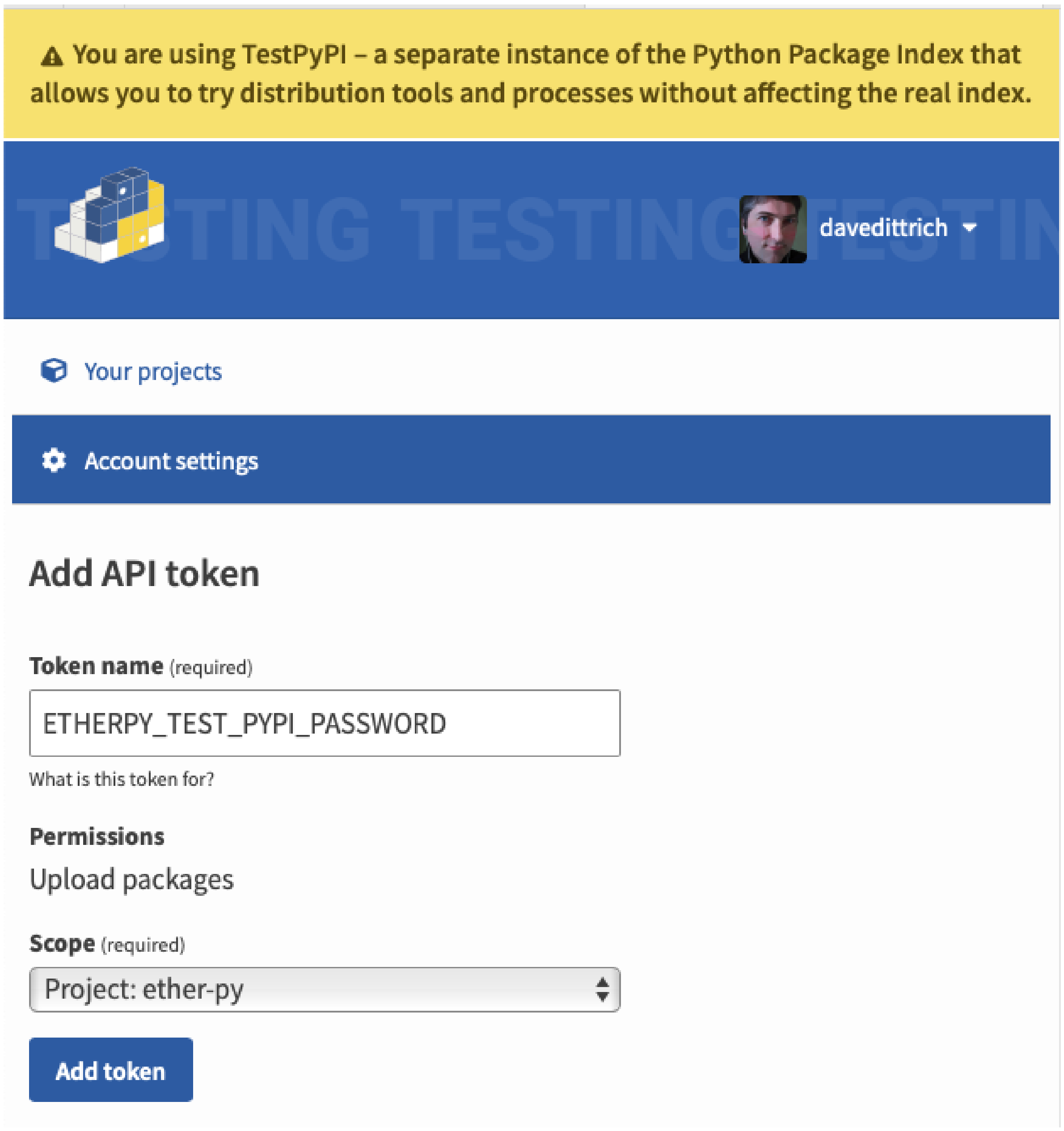
```
- name: Publish release candidate artifacts to TestPyPI
  if: >-
    github.event_name == 'push' &&
    startsWith(github.ref, 'refs/tags') &&
    contains(github.ref, 'rc') == true
  uses: pypa/gh-action-pypi-publish@release/v1
  with:
    repository_url: https://test.pypi.org/legacy/
    user: __token__
    password: ${{ secrets.ETHERPY_TEST_PYPI_PASSWORD }}
    packages_dir: ${{ steps.get_vars.outputs.ARTIFACT }}
    verify_metadata: false
- name: Publish tagged artifacts to PyPI
  if: >-
    github.event_name == 'push' &&
    startsWith(github.ref, 'refs/tags') &&
    contains(github.ref, 'rc') == false
  uses: pypa/gh-action-pypi-publish@release/v1
  with:
    user: __token__
    password: ${{ secrets.ETHERPY_PYPI_PASSWORD }}
    packages_dir: ${{ steps.get_vars.outputs.ARTIFACT }}
    verify_metadata: false
```

Before this will work, you must have first created a token named `ETHERPY_TEST_PYPI_PASSWORD` on Test PyPI and another named `ETHERPY_PYPI_PASSWORD` on PyPI and stored them both in *encrypted secrets* within your GitHub repo.

Note: You will be copying and pasting the token values from one system to another, and we will be limiting the scope of the tokens to the *project* level and not for your entire account. This means one token per GitHub project per package index. It is best to use the same token names on both GitHub and Test PyPI or PyPI, for consistency and less confusion later on.

You may also want to use separate browser windows to have them visible at the same time to ensure the right tokens are used.

1. Open a browser tab and log into your GitHub account. Go to your project's repo page, then select **Settings**, then select **Secrets** from the menu on the left.
2. Open a new browser tab to https://test.pypi.org/ and log into your account. Select **Account settings** on Test PyPI from the menu on the left, then then choose **Create a token for ether-py**. Enter `ETHERPY_TEST_PYPI_PASSWORD` for the token name. For scope, select this project to limit the scope. Finally, press **Add token**. You will only be able to see the token value once. Get ready to copy it to paste into the **Value** field in the GitHub project window.



⚠ You are using TestPyPI – a separate instance of the Python Package Index that allows you to try distribution tools and processes without affecting the real index.

Add API token

Token name (required)

ETHERPY_TEST_PYPI_PASSWORD

What is this token for?

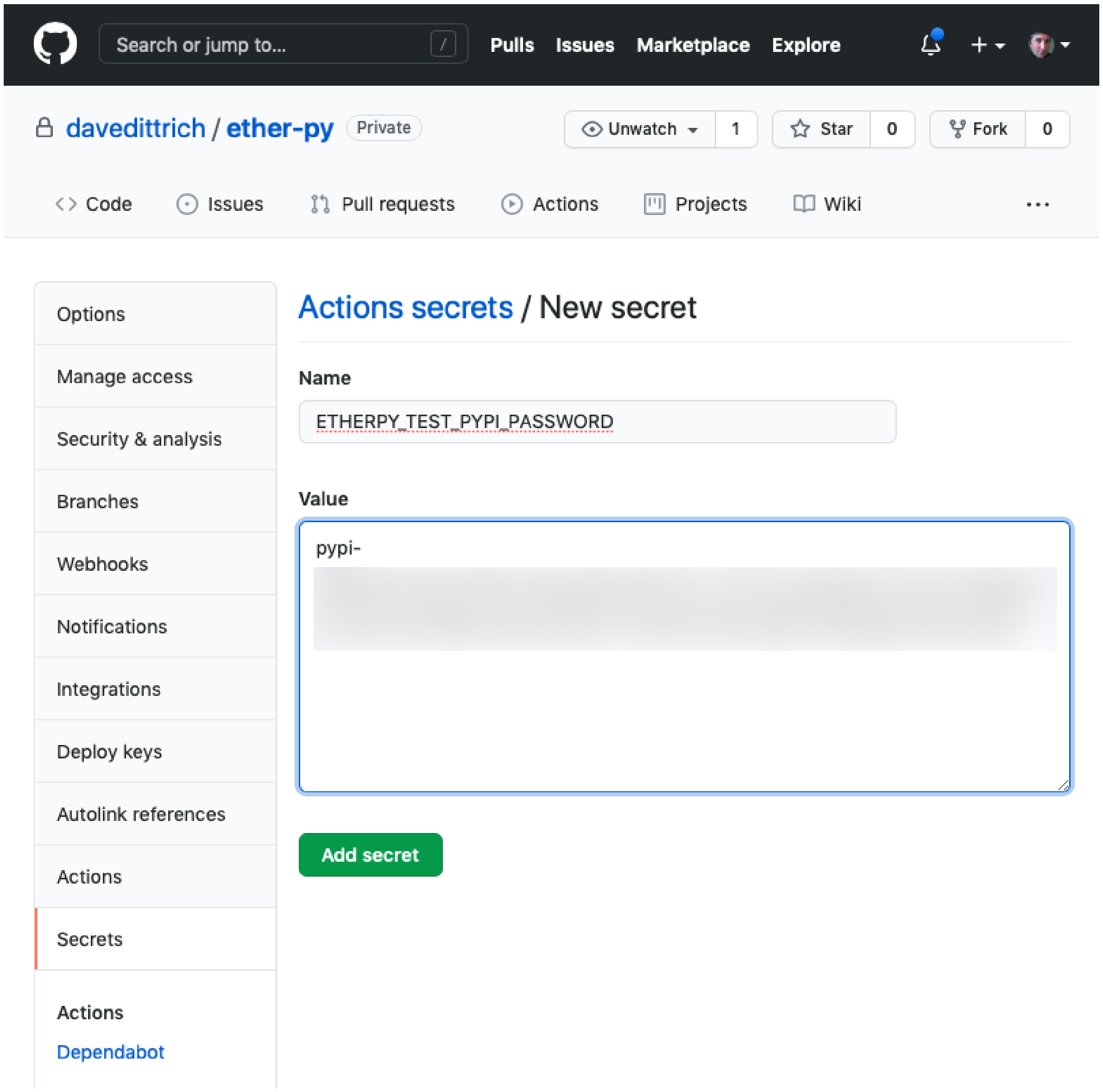
Permissions

Upload packages

Scope (required)

Project: ether-py

3. In the GitHub window, create a new secret named `ETHERPY_TEST_PYPI_PASSWORD`. Paste the token into the **Value** field, then select **Add Secret**.



Actions secrets / New secret

Name

ETHERPY_TEST_PYPI_PASSWORD

Value

pypi-
[redacted]

4. Repeat the same process for PyPI using the token name `ETHERPY_PYPI_PASSWORD`.

3.4.1 For Every Release

The publishing workflow is triggered by pushing a new tag, so the `push` always has to be the *last* step in the release process.

Before doing that push, make sure you update `HISTORY.rst` by analyzing your commit messages since the last release and summarizing the highlights.

Note: While there are tools you can use to auto-generate `ChangeLog` files, using every Git commit message is a little too verbose.

Add and commit the changes:

```
$ git add HISTORY.rst
$ git commit -m "Changelog for upcoming release 2021.3.1"
```

Warning: DO NOT FORGET to run `make test` when you think you are ready to make a release. This helps make sure no remaining bugs or coding errors will cause the GitHub workflow to fail before it gets to the publish step, which would require an additional tag following code fixes.

3.4.2 For Test Releases

To publish a test release on the `develop` branch, you only need to add an annotated tag with `rc` in the tag string. The next candidate release tag in March following the example above would thus be `v2021.3.1rc4`:

```
$ git tag -a v2021.3.rc4
```

Pushing the tag will trigger the release to Test PyPI.

3.4.3 For Full Releases

Full releases are more involved.

First make sure that all tests pass and that you are satisfied all code and documentation changes are ready. Then merge all the new commits into the `master` branch and resolve any merge conflicts.

If you are using the Git HubFlow tool, you will now start a new release with `git hf release start` with the new version number.

You are now ready to bump the version number in source files and update the history file. Assuming we are starting with version `2021.3.0`, there are two cases we need to consider in terms of choosing the new release number.

1. If you are making a release *within the same month* as the prior release, the new version number will only need the patch component to be incremented. To go from `2021.3.0` to `2021.3.1`, you just need to do:

   ```
   $ bump2version patch
   ```

2. If you are making release in a *different month or year* from the previous release, the new version number will have a different major and or minor number change.

   In the case where the release is in the *next* month (e.g., going from `2021.3.0` to `2021.4.0`), you only need to do:

   ```
   $ bump2version minor
   ```

   If the time difference is longer than one month (e.g., going from `2021.3.0` to `2021.8.0`), you need to do:

   ```
   $ bump2version --current-version $(cat VERSION) --new-version 2021.8.0 patch
   ```

If you are using the Git HubFlow tool, do your normal `release finish` and it will handle the tagging and pushing. Otherwise, manually tag and push the `master` branch and associated tag:

```
$ git tag -a v$(cat VERSION)
$ git push && git push --tags
```

Either way, the pushed tag will create the new release on both GitHub and PyPI.

CHAPTER FOUR

CREDITS

4.1 Development Lead

- Dave Dittrich <dave.dittrich@gmail.com>

4.2 Contributors

None yet. Why not be the first?

4.3 Project source template

This documentation and the repository for the `ether-py` project package were created with the Cookiecutter templating engine from the <https://github.com/davedittrich/cookiecutter-cliffapp-template.git> project template.

CHAPTER FIVE

LICENSE

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

INDEX