Improving $ click _ library

Architecture of large projects in bioinformatics final project

Ada Hryniewicka
Maciej Dzikowski
15.06.2022

What is `$ click_`

Command Line Interface Creation Kit

- python package for creating command line interfaces
- arbitrary nesting of commands
- automatic help page generation
- supports lazy loading of subcommands at runtime

Click program example

```
import click

@click.command()
@click.option('--count', default=1, help='Number of greetings.')
@click.option('--name', prompt='Your name',
              help='The person to greet.')
def hello(count, name):
    """Simple program that greets NAME for a total of COUNT times."""
    for x in range(count):
        click.echo(f"Hello {name}!")

if __name__ == '__main__':
    hello()
```

Output

```
$ python hello.py --count=3
Your name: John
Hello John!
Hello John!
Hello John!
```

```
$ python hello.py --help
Usage: hello.py [OPTIONS]

  Simple program that greets NAME for a total of COUNT times.

Options:
  --count INTEGER  Number of greetings.
  --name TEXT      The person to greet.
  --help           Show this message and exit.
```

Parameter Types

Arguments

- arguments can do less than options
- accept an arbitrary number of arguments
- convenient to use

Options

- automatic prompting for missing input
- act as flags (boolean or otherwise)
- option values can be pulled from environment variables, arguments can not
- options are fully documented in the help page, arguments are not

Why choose click?

Click vs Argparse, Docopt etc.

- Click does not just parse, it also dispatches to the appropriate code
- Click has strong information available for all parameters and commands,
- Click has a strong understanding of what types are, and it can give the user consistent error messages
- Click has enough meta information available for its whole program to evolve without forcing developers to adjust their programs

Click as a useful tool for bioinformaticians

Example of usage

- The program that connects to the BLAST based on the given sequences
- Returns the result of program operation as a file in .fasta format

```python
@click.command()
@click.argument("sequence_file")
@click.argument("output")
@click.option("-i", "--identity", "ident", default=0.9,
              help="Minimal percent identity used during searching the NCBI database. Input range: <0, 1>. Default: 0.9.")
@click.option("-e", "--e_value", "e", default=10e-10,
              help="E-value used during searching the NCBI database. Default: 10e-10.")
def run(sequence_file: str, output: str, ident: float, e: float):
    """
    sequence_file: Path to a fasta format file with a protein sequence.

    output: Path to a fasta format output file.
    """
```

```
Usage: extend.py [OPTIONS] SEQUENCE_FILE OUTPUT

  sequence_file: Path to a fasta format file with a protein sequence.

  output: Path to a fasta format output file.

Options:
  -i, --identity FLOAT  Minimal percent identity used during searching the
                        NCBI database. Input range: <0, 1>. Default: 0.9.
  -e, --e_value FLOAT   E-value used during searching the NCBI database.
                        Default: 10e-10.
  --help                Show this message and exit.
```

Our ideas for improvement

1. **help** parameter for **@click.argument** decorator to preserve an UI consistency

2. Parameter for **@click.argument** decorator to give a possibility of creating a Python argument differed from an argument name displayed in a help message Allows to avoid shadowing built-in names, e.g. when argument has to be named „filter"

@click.arguemnt decorator

Basic:

```
@click.command()
@click.argument('filename')
def touch(filename):
    """Print FILENAME."""
    click.echo(filename)
```

```
$ touch foo.txt
foo.txt
```

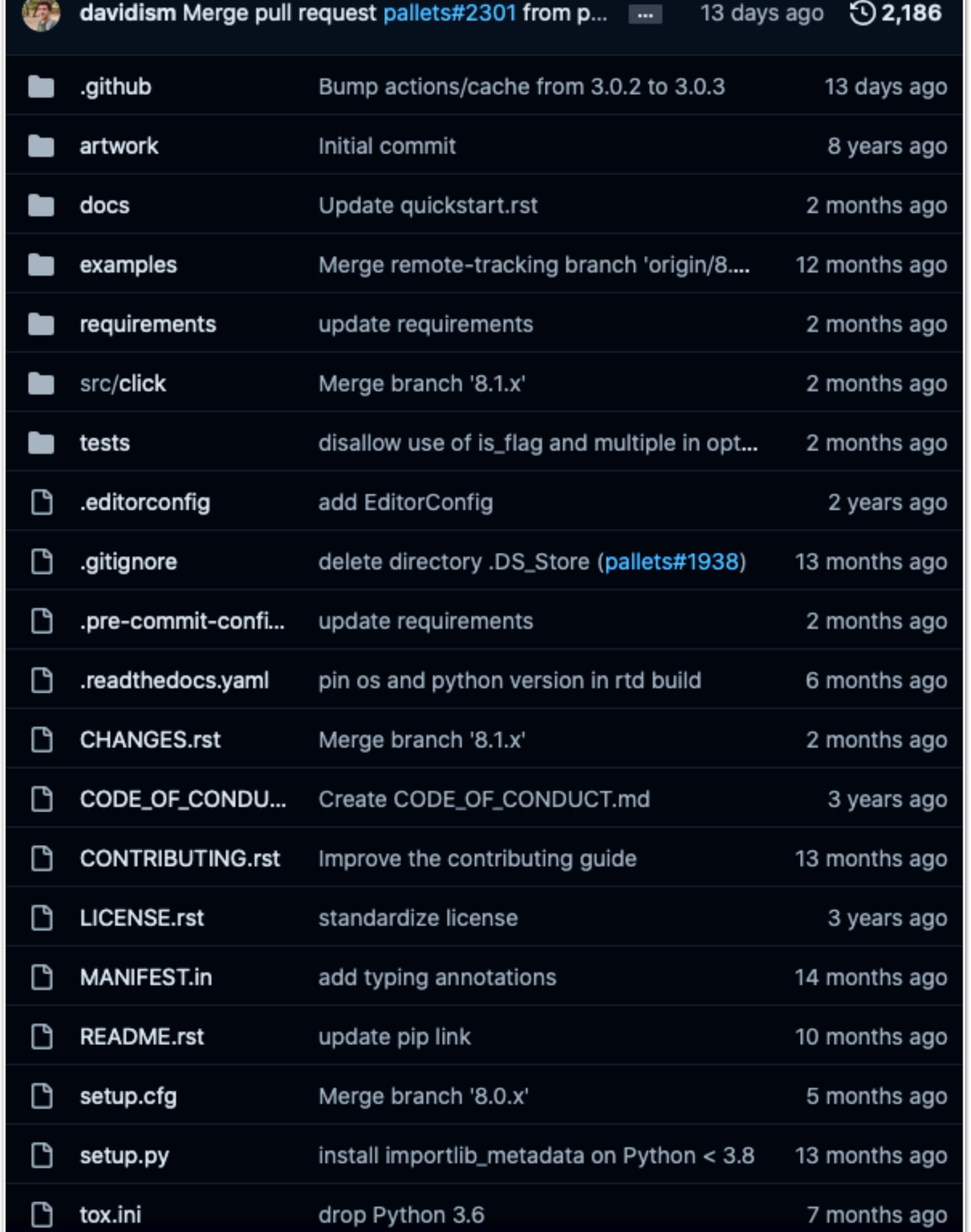
Variadic:

```
@click.command()
@click.argument('src', nargs=-1)
@click.argument('dst', nargs=1)
def copy(src, dst):
    """Move file SRC to DST."""
    for fn in src:
        click.echo(f"move {fn} to folder {dst}")
```

```
$ copy foo.txt bar.txt my_folder
move foo.txt to folder my_folder
move bar.txt to folder my_folder
```

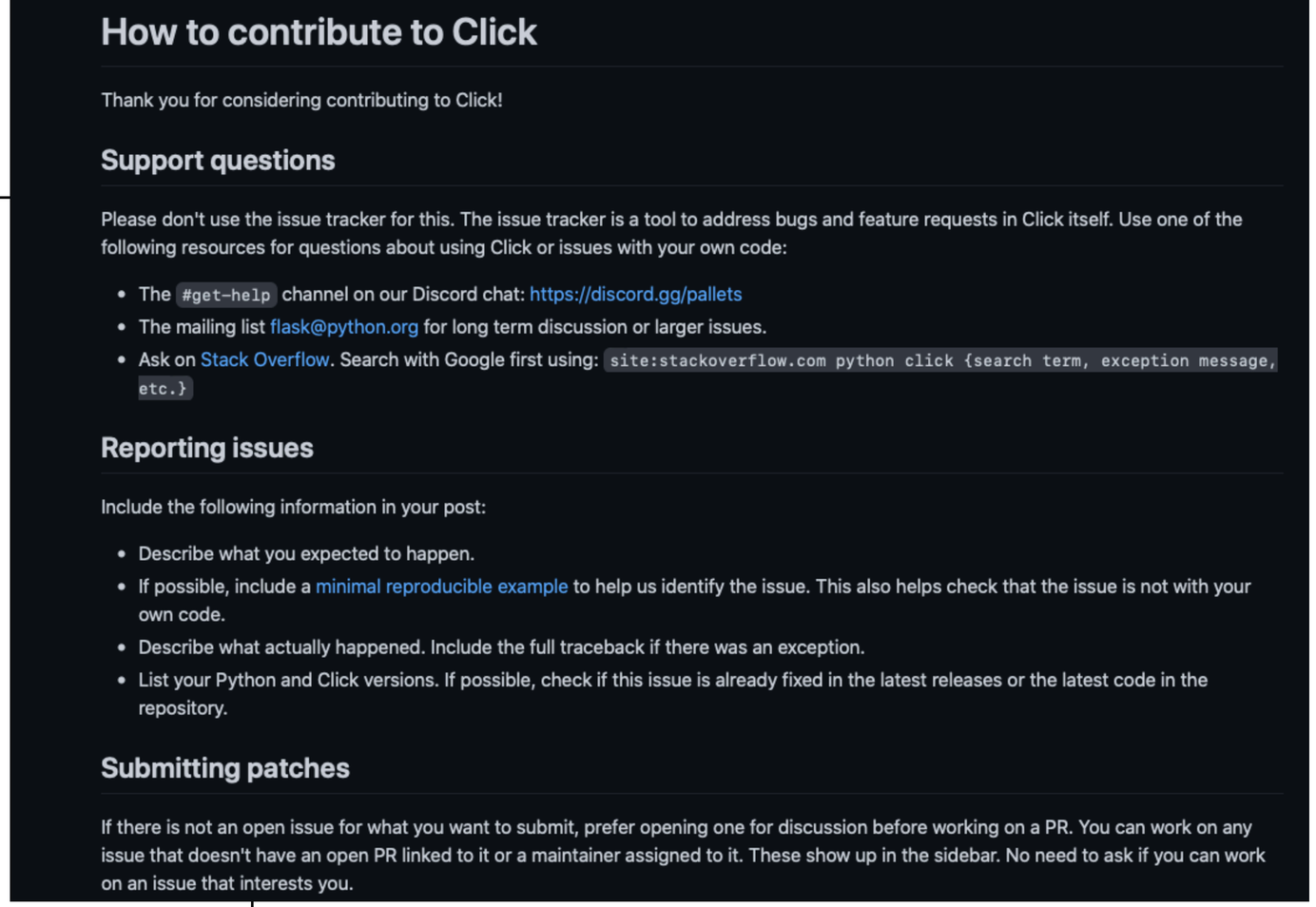
Challenges

- Massive documentation
- Learning about the existing code
- The improvement could not affect the current user scripts



GitHub workflow

1. First time setup
   - Configure git
   - Clone main repository
   - Create a virtualenv
   - Upgrade pip and setuptools
   - Install the development dependencies
   - Install the pre-commit hooks
2. Start coding
   - Push your commits to your fork on GitHub
3. Run the tests
4. Run the test coverage
   - Generate the report
5. Build the docs



How to contribute to Click

Thank you for considering contributing to Click!

Support questions

Please don't use the issue tracker for this. The issue tracker is a tool to address bugs and feature requests in Click itself. Use one of the following resources for questions about using Click or issues with your own code:

- The `#get-help` channel on our Discord chat: https://discord.gg/pallets
- The mailing list flask@python.org for long term discussion or larger issues.
- Ask on Stack Overflow. Search with Google first using: `site:stackoverflow.com python click {search term, exception message, etc.}`

Reporting issues

Include the following information in your post:

- Describe what you expected to happen.
- If possible, include a minimal reproducible example to help us identify the issue. This also helps check that the issue is not with your own code.
- Describe what actually happened. Include the full traceback if there was an exception.
- List your Python and Click versions. If possible, check if this issue is already fixed in the latest releases or the latest code in the repository.

Submitting patches

If there is not an open issue for what you want to submit, prefer opening one for discussion before working on a PR. You can work on any issue that doesn't have an open PR linked to it or a maintainer assigned to it. These show up in the sidebar. No need to ask if you can work on an issue that interests you.

Results

First assumption

Before

```python
@click.command()
@click.argument('argument')
@click.option('-o', "--option", "opt", help='Option help text')
def run(argument, opt):
    """
    argument: Argument help text.
    """
    print(argument, opt)
```

```
Usage: example.py [OPTIONS] ARGUMENT

  argument: Argument help text.

Options:
  -o, --option TEXT  Option help text
  --help             Show this message and exit.
```

After

```python
@click.command()
@click.argument('argument', help="It's a feature!")
@click.option('-o', "--option", "opt", help='Option help text')
def run(argument, opt):
    print(argument, opt)
```

```
Usage: example.py [OPTIONS] ARGUMENT

Arguments:
  ARGUMENT  It's a feature!  [required]

Options:
  -o, --option TEXT  Option help text
  --help             Show this message and exit.
```

Results

First assumption cd.

```
@click.command()
@click.argument('argument', help="It's a feature!", hidden=True)
@click.option('-o', "--option", "opt", help='Option help text')
def run(argument, opt):
    print(argument, opt)
```

```
Usage: example.py [OPTIONS] ARGUMENT

Options:
  -o, --option TEXT  Option help text
  --help             Show this message and exit.
```

Results
Second assumption

Before

```
@click.command()
@click.argument('argument')
@click.option('-o', "--option", "opt", help='Option help text')
def run(argument, opt):
    """
    argument: Argument help text.
    """
    print(argument, opt)

Usage: example.py [OPTIONS] ARGUMENT

  argument: Argument help text.

Options:
  -o, --option TEXT  Option help text
  --help             Show this message and exit.
```

After

```
@click.command()
@click.argument('argument', "arg")
@click.option('-o', "--option", "opt", help='Option help text')
def run(arg, opt):
    print(arg, opt)


Usage: example.py [OPTIONS] ARGUMENT

Options:
  -o, --option TEXT  Option help text
  --help             Show this message and exit.
```

Tests

PASSED!

```
============================= test session starts ==============================
platform linux -- Python 3.9.7, pytest-7.1.2, pluggy-1.0.0
rootdir: /home/macius/PycharmProjects/click/click, configfile: setup.cfg, testpaths: tests
plugins: anyio-2.2.0
collected 598 items

tests/test_arguments.py ....................................                [  5%]
tests/test_basic.py ........................................................ [ 15%]
tests/test_chain.py .............x                                          [ 17%]
tests/test_command_decorators.py ...                                       [ 18%]
tests/test_commands.py ......................                              [ 22%]
tests/test_compat.py .                                                     [ 22%]
tests/test_context.py ......................                               [ 26%]
tests/test_custom_classes.py ......                                        [ 27%]
tests/test_defaults.py ...                                                 [ 27%]
tests/test_formatting.py ................                                  [ 30%]
tests/test_imports.py .                                                    [ 30%]
tests/test_info_dict.py ....................                               [ 34%]
tests/test_normalization.py ...                                            [ 34%]
tests/test_options.py ..................................................... [ 53%]
tests/test_parser.py .......                                               [ 55%]
tests/test_shell_completion.py .....................................       [ 61%]
tests/test_termui.py ..........................................sssssssssssssssssss [ 74%]
tests/test_testing.py ......................                               [ 78%]
tests/test_types.py ................................                       [ 84%]
tests/test_utils.py ........................................................ [100%]

======================== 576 passed, 21 skipped, 1 xfailed in 1.87s =========================
```

References

- https://click.palletsprojects.com/en/8.1.x/