PARSEARG: TURNS ARGPARSE ON ITS HEAD, THE DECLARATIVE WAY

THOMAS P. HARTE

Contents

1. Quickstart

1.1. **Overview.** `parsearg` is a Python package for writing command-line interfaces ("CLI") that augments (rather than replaces) the standard Python module for writing CLIs, `argparse`. There is nothing wrong with `argparse`: It's fine in terms of the *functionality* that it provides, but it can be clunky to use, especially when a program's structure has subcommands, or nested subcommands (*i.e.* subcommands that have subcommands). Moreover, because of the imperative nature of `argparse`, it makes it hard to understand how a program's interface is structured (*viz.* the program's "view").

`parsearg` puts a layer on top of `argparse` that makes writing a CLI easy. you declare your CLI, *i.e.* your program's view, with a `dict` so that the view is a data structure (*i.e.* pure configuration). The view declares the *intent* of the CLI and you are relieved of the trouble of having to instruct `argparse` on how to put the CLI together: `parsearg` does that for you.

1.2. **Usage.** Suppose we wish to create a program called `todos.py` to manage the TO-DOs of a set of different users. We want to have subprograms of `todos.py`; for example, we may want to create a user (`python todos.py create user`, say), or we may want to create a TO-DO for a particular user (`python todos.py create todo`, say). We might also want to add optional parameters to each subprogram such as the user's email and phone number, or the TO-DO's due date. An invocation of the program's CLI might look like the following:

```
python todos.py create user Bob --email=bob@email.com --phone=+1-212-555-1234
python todos.py create todo Bob 'taxes' --due-date=2021-05-17
```

With `argparse`, the subprogram `create` would necessitate fiddling with subparsers. With `parsearg`, the CLI for the above is declared with a `dict` and `parsearg.parser.ParseArg` supplants the normal use of `argparse.ArgumentParser`. Moreover, the callback associated with each subcommand is explicitly linked to its declaration.

Date: 2021-06-08 Build: `v-0.2.0 (6732c53204ed83d83e8e22d455528533c851ead0)`.

```python
import sys
from parsearg.parser import ParseArg

def create_user(args):
    print(f'created user: {args.name!r} (email: {args.email}, phone: {args.phone})')

def create_todo(args):
    print(f'created TO-DO for user {args.user!r}: {args.title} (due: {args.due_date})')

view = {
    'create|user': {
        'callback':   create_user,
        'name':       {'help': 'create user name', 'action': 'store'},
        '-e|--email': {'help': "create user's email address", 'action': 'store', 'default': ''},
        '-p|--phone': {'help': "create user's phone number", 'action': 'store', 'default': ''},
    },
    'create|todo': {
        'callback':   create_todo,
        'user':       {'help': 'user name', 'action': 'store'},
        'title':      {'help': 'title of to-do', 'action': 'store'},
        '-d|--due-date': {'help': 'due date for the to-do', 'action': 'store', 'default': None},
    },
}

def main(args):
    # ParseArg takes the place of argparse.ArgumentParser
    parser = ParseArg(d=view)

    # parser.parse_args returns an argparse.Namespace
    ns     = parser.parse_args(args)

    # ns.callback contains the function in the 'callback' key of 'view'
    result = ns.callback(ns)

if __name__ == "__main__":
    args = sys.argv[1:] if len(sys.argv) > 1 else []

    main(' '.join(args))
```

A fully worked version of the `todos.py` example is presented in the docs. The output of the above is:

```
python todos.py create user Bob --email=bob@email.com --phone=212-555-1234
```

```
  created user: 'Bob' (email: bob@email.com, phone: 212-555-1234)
```

```
python todos.py create todo Bob 'taxes' --due-date=2021-05-17
```

```
  created TO-DO for user 'Bob': taxes (due: 2021-05-17)
```

Because `parsearg` is built on top of `argparse`, all the usual features are available, such as the extensive help features (essentially making the CLI self-documenting):

```
python todos.py --help
```

```
  usage: todos.py [-h] {create} ...

  positional arguments:
    {create}

  optional arguments:
    -h, --help  show this help message and exit
```

```
python todos.py create --help
```

```
usage: todos.py create [-h] {todo,user} ...

positional arguments:
  {todo,user}

optional arguments:
  -h, --help   show this help message and exit
```

```
python todos.py create user --help
```

```
usage: todos.py create user [-h] [-e EMAIL] [-p PHONE] name

positional arguments:
  name                  create user name

optional arguments:
  -h, --help            show this help message and exit
  -e EMAIL, --email EMAIL
                        create user's email address
  -p PHONE, --phone PHONE
                        create user's phone number
```

```
python todos.py create todo --help
```

```
usage: todos.py create todo [-h] [-d DUE_DATE] user title

positional arguments:
  user                  user name
  title                 title of to-do

optional arguments:
  -h, --help            show this help message and exit
  -d DUE_DATE, --due-date DUE_DATE
                        due date for the to-do
```

2. Overview

The "standard" Python module for writing command-line interfaces ("CLI") is `argparse`. It is standard in so far as it is one of the batteries that comes included with the Python distribution, so no special installation is required. Probably because `argparse` is a bit clunky to use, many other (non-standard) packages have been developed for creating CLIs. Why "clunky"? Putting together a CLI with `argparse` alone is nothing if not an exercise in imperative programming, and this has three very negative consequences:

(1) It obfuscates the intention of the CLI design;

(2) It is prone to errors;

(3) It discourages CLI design in the first instance; it makes debugging a CLI design very difficult; and it makes refactoring or re-configuring the CLI design overly burdensome.

In spite of this clunkiness, `argparse` has everything we need in terms of functionality. `parsearg`, then, is nothing more than a layer over `argparse` that exposes the `argparse` functionality via a `dict`. The `dict` is the View component of the Model-View-Controller ("MVC") design pattern. The `dict` embeds callbacks from the Controller component, thereby achieving a clean separation of duties, which is what the MVC pattern calls for. By separating the View component into a `dict`, the CLI design can be expressed in a declarative way: `parsearg` manifests the *intention* of the CLI design without having to specify how that design is implemented in terms of `argparse`'s parsers and subparsers (`parsearg` does that for you).

Other packages—such as `click` and `plac`—effectively decorate functions that are part of the Controller with functionality from the View. Unfortunately, while this may expose the functionality of `argparse` in a more friendly way via the packages' decorators, it dissipates the elements of the View across the Controller and in so doing it makes the CLI design difficult to grasp.

The `parsearg` philosophy is that `argparse` is already good enough in terms of the functionality that it provides, but that it just needs a little nudge in terms of how it's used. Arguments to be added to a CLI with `argparse` can be clearly specified as data, as can the callbacks that consume these arguments. `parsearg` takes advantage of this by specifiying everything (in the View component of MVC) as a `dict`, from which `parsearg` then generates a parser (or set of nested parsers) using `argparse`. The Controller is then free to use the generated parser.

The `parsearg` approach is declarative because it manifests the CLI design in a data structure: a `dict`, which is one of Python's built-in data structures. The keys of this `dict` form a flattened tree of the CLI's subcommands. Keys like `A|B|C` are easy to specify and neatly summarize the nested hierarchy of subcommands: `A -> B -> C`. They are also easy to change. The magic, such as it is, of `parsearg` is that it unflattens this flattened tree into a tree of `argparse` parsers. `parsearg` requires nothing special: It works with Python out of the box, and therefore uses what's already available without introducing dependencies.

Simple? The following examples should help.

3. PARSEARG.EXAMPLES.TODOS

Yet another To-Do app? Yes, because:

(1) It illustrates a sufficiently realistic, but not overly complex, problem;

(2) It also illustrates the MVC pattern in the wild;

(3) It shows how `parsearg` neatly segments the View component into a `dict`.

Let's start with the outer layer of the onion. How do we interact with `todos.py`? First, create some users in the `User` table with the `create user` subcommand of `todos.py`. Note that we do not (yet) have a phone number for user `bar`, nor do we have an email address for user `qux`:

```
python todos.py create user foo -e foo@foo.com -p 212-555-1234
python todos.py create user bar -e bar@bar.com
python todos.py create user qux -p 212-123-5555
```

```
created user: 'foo' (email: foo@foo.com, phone: 212-555-1234)
created user: 'bar' (email: bar@bar.com, phone: )
created user: 'qux' (email: , phone: 212-123-5555)
```

Second, create some to-dos in the `Todo` table with the `create todo` subcommand of `todos.py`.

```
python todos.py create todo foo title1 -c description1 -d 2020-11-30
python todos.py create todo foo title2 -c description2 --due-date=2020-12-31
python todos.py create todo qux todo-1 --description=Christmas-party -d 2020-11-30
python todos.py create todo qux todo-2 --description=New-Year-party
```

```
usage: todos.py [-h] {create} ...
todos.py: error: unrecognized arguments: -c description1
usage: todos.py [-h] {create} ...
todos.py: error: unrecognized arguments: -c description2
usage: todos.py [-h] {create} ...
todos.py: error: unrecognized arguments: --description=Christmas-party
usage: todos.py [-h] {create} ...
todos.py: error: unrecognized arguments: --description=New-Year-party
```

Let's make some changes to the records entered so far. We can add an email address for user `qux` and a phone number for user `bar` using the `update user email` and `update user phone` subcommands, respectively:

```
python todos.py update user email qux qux@quxbar.com
python todos.py update user phone bar 203-555-1212
```

```
usage: todos.py [-h] {create} ...
todos.py: error: invalid choice: 'update' (choose from 'create')
usage: todos.py [-h] {create} ...
todos.py: error: invalid choice: 'update' (choose from 'create')
```

Now update two of the to-dos, changing the `title` and the `description` in the fourth to-do using the `update todo title` and `update todo description` subcommands, respectively:

```
python todos.py update todo title 4 most-important
python todos.py update todo description 4 2021-party
```

```
usage: todos.py [-h] {create} ...
todos.py: error: invalid choice: 'update' (choose from 'create')
usage: todos.py [-h] {create} ...
todos.py: error: invalid choice: 'update' (choose from 'create')
```

```
python todos.py show users
python todos.py show todos
```

```
usage: todos.py [-h] {create} ...
todos.py: error: invalid choice: 'show' (choose from 'create')
usage: todos.py [-h] {create} ...
todos.py: error: invalid choice: 'show' (choose from 'create')
```

The result of these commands is that the two tables (`User` and `Todo`) are populated in a SQLite database:

```
sqlite3 todo.db 'select * from User;'
```

```
Error: no such table: User
```

```
sqlite3 todo.db 'select * from Todo;'
```

```
Error: no such table: Todo
```

Let's look at the Python code for `todos.py`.

The View component is entirely contained within a single `dict`, *viz.* `view`, which has been formatted here for clarity using `parsearg.utils.show`:

```
from parsearg.examples.todos import view
from parsearg.utils import show

show(view)
```

```
'purge|users':
    'callback':
    <function purge_users at 0x7f11d6ff6700>
'purge|todos':
    'callback':
    <function purge_todos at 0x7f11d6fcfaf0>
'show|users':
    'callback':
    <function show_users at 0x7f11d6fcfb80>
'show|todos':
    'callback':
    <function show_todos at 0x7f11d6fcfc10>
'create|user':
    'callback':
    <function create_user at 0x7f11d6fcfca0>
    'name':
    {'help': 'create user name', 'action': 'store'}
    '-e|--email':
```

```
    {'help': "create user's email address", 'action': 'store', 'default': ''}
    '-p|--phone':
    {'help': "create user's phone number", 'action': 'store', 'default': ''}
'create|todo':
    'callback':
    <function create_todo at 0x7f11d6fcfd30>
    'user':
    {'help': 'user name', 'action': 'store'}
    'title':
    {'help': 'title of to-do', 'action': 'store'}
    '-c|--description':
    {'help': 'description of to-do', 'action': 'store', 'default': ''}
    '-d|--due-date':
    {'help': 'due date for the to-do', 'action': 'store', 'default': None}
'update|user|email':
    'callback':
    <function update_user_email at 0x7f11d6fcfdc0>
    'name':
    {'help': 'user name', 'action': 'store'}
    'email':
    {'help': 'user email', 'action': 'store'}
'update|user|phone':
    'callback':
    <function update_user_phone at 0x7f11d6fcfe50>
    'name':
    {'help': 'user name', 'action': 'store'}
    'phone':
    {'help': 'user phone', 'action': 'store'}
'update|todo|title':
    'callback':
    <function update_todo_title at 0x7f11d6fcfee0>
    'id':
    {'help': 'ID of to-do', 'action': 'store'}
    'title':
    {'help': 'title of to-do', 'action': 'store'}
'update|todo|description':
    'callback':
    <function update_todo_description at 0x7f11d6fcff70>
    'id':
    {'help': 'ID of to-do', 'action': 'store'}
    'description':
    {'help': 'description of to-do', 'action': 'store'}
```

We can generate a tree view of the CLI design specified by the above `dict`, namely `view`, as follows:

```
from parsearg.parser import ParseArg

print(
    ParseArg(d=view, root_name='TODO').tree.show(quiet=True)
)
```

```
TODO
    purge
        todos
        users
    update
        user
            phone
            email
        todo
            description
            title
    show
        todos
        users
    create
        user
        todo
```

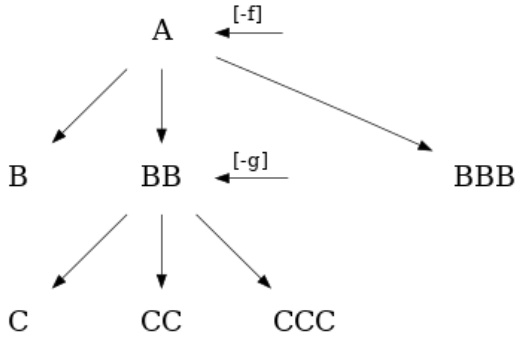
4. How PARSEARG works

It is easier to explain how `parsearg` works with a simpler abstraction than the above example `parsearg.examples.todos.py`. Here, we will consider nested parsers as trees. We introduce two trees for this purpose:

(1) The “A tree”, and

(2) The “A-AA tree”.

4.1. **The A-tree.** The “A tree” has three levels. As each node of the tree must necessarily occupy a positional argument of the command line, `[-f]` and `[-g]` are correspondingly *optional arguments* that attach to the nodes (`A` and `BB`, respectively, in the below diagram).



Let’s look at the Python code for `a.py`.

Consider the `dict` that represents the View component:

```
from parsearg.examples.a import view

show(view)
```

```
'A':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7d10c10>
    '-c':
    {'help': 'A [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A verbosity', 'action': 'store_true'}
'A|B':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d70409d0>
    '-c':
    {'help': 'A B [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A B verbosity', 'action': 'store_true'}
'A|BB':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7040a60>
    '-c':
    {'help': 'A BB [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A BB erbosity', 'action': 'store_true'}
'A|BB|C':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7040af0>
    '-c':
    {'help': 'A BB C [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A BB C verbosity', 'action': 'store_true'}
'A|BB|CC':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7040b80>
    '-c':
    {'help': 'A BB CC [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A BB CC verbosity', 'action': 'store_true'}
'A|BB|CCC':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7040c10>
```

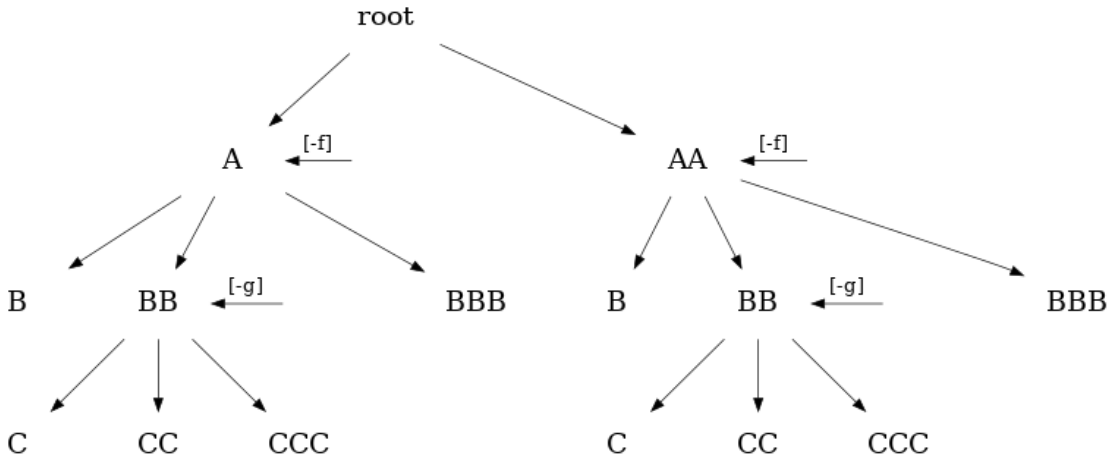
```
    '-c':
    {'help': 'A BB CCC [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A BB CCC verbosity', 'action': 'store_true'}
'A|BBB':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7040ca0>
    '-c':
    {'help': 'A BBB [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A BBB verbosity', 'action': 'store_true'}
```

and then the tree that the parsed `dict` is represented by:

```
print(
    ParseArg(d=view, root_name='TODO').tree.show(quiet=True)
)
```

```
TODO
    A
        B
        BBB
        BB
            C
            CC
            CCC
```

4.2. **The A-A tree.** The A-AA tree simply extends the A tree one level:



Let's look at the Python code for a_aa.py.

Consider the `dict` that represents the View component:

```
from parsearg.examples.a_aa import view

show(view)
```

```
'A':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7044670>
    '-c':
    {'help': 'A [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A verbosity', 'action': 'store_true'}
'A|B':
    'callback':
```

```
    <function make_callback.<locals>.func at 0x7f11d70445e0>
    '-c':
    {'help': 'A B [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A B verbosity', 'action': 'store_true'}
'A|BB':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7044550>
    '-c':
    {'help': 'A BB [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A BB erbosity', 'action': 'store_true'}
'A|BB|C':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d70444c0>
    '-c':
    {'help': 'A BB C [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A BB C verbosity', 'action': 'store_true'}
'A|BB|CC':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7044430>
    '-c':
    {'help': 'A BB CC [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A BB CC verbosity', 'action': 'store_true'}
'A|BB|CCC':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d70443a0>
    '-c':
    {'help': 'A BB CCC [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A BB CCC verbosity', 'action': 'store_true'}
'A|BBB':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d70440d0>
    '-c':
    {'help': 'A BBB [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'A BBB verbosity', 'action': 'store_true'}
'AA':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7044310>
    '-c':
    {'help': 'AA [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'AA verbosity', 'action': 'store_true'}
'AA|B':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7044280>
    '-c':
    {'help': 'AA B [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'AA B verbosity', 'action': 'store_true'}
'AA|BB':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d70441f0>
    '-c':
    {'help': 'AA BB [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'AA BB verbosity', 'action': 'store_true'}
'AA|BB|C':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7044160>
    '-c':
    {'help': 'AA BB C [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'AA BB C verbosity', 'action': 'store_true'}
'AA|BB|CC':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7044040>
    '-c':
    {'help': 'AA BB CC [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'AA BB CC verbosity', 'action': 'store_true'}
'AA|BB|CCC':
    'callback':
```

```
    <function make_callback.<locals>.func at 0x7f11d7044c10>
    '-c':
    {'help': 'AA BB CCC [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'AA BB CCC verbosity', 'action': 'store_true'}
'AA|BBB':
    'callback':
    <function make_callback.<locals>.func at 0x7f11d7044ca0>
    '-c':
    {'help': 'AA BBB [optional pi]', 'action': 'store_const', 'const': 3.141593}
    '-v|--verbose':
    {'help': 'AA BBB verbosity', 'action': 'store_true'}
```

and then the tree that the parsed `dict` is represented by:

```
print(
    ParseArg(d=view, root_name='TODO').tree.show(quiet=True)
)
```

```
TODO
    AA
        B
        BBB
        BB
            C
            CC
            CCC
    A
        B
        BBB
        BB
            C
            CC
            CCC
```

We can now run the A-tree and the A-AA-tree examples, respectively:

```
from parsearg.examples.a import main
main()
```

```
NODE :: 'A':
--------------
usage:  A [-h] [-c] [-v] {B,BBB,BB} ...

positional arguments:
  {B,BBB,BB}

optional arguments:
  -h, --help     show this help message and exit
  -c             A [optional pi]
  -v, --verbose  A verbosity


'A':
----
        args: {'c': None, 'verbose': False}
        <Mock name='mock.A' id='139714611964512'>
'A -v':
-------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.A' id='139714611964512'>
'A -c':
-------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.A' id='139714611964512'>
'A -v -c':
----------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.A' id='139714611964512'>


NODE :: 'A B':
```

```
----------------
usage: A B [-h] [-c] [-v]

optional arguments:
  -h, --help     show this help message and exit
  -c             A B [optional pi]
  -v, --verbose  A B verbosity


'A B':
------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.A_B' id='139714598437792'>
'A B -v':
---------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.A_B' id='139714598437792'>
'A B -c':
---------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.A_B' id='139714598437792'>
'A B -v -c':
------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.A_B' id='139714598437792'>


NODE :: 'A BB':
-----------------
usage: A BB [-h] [-c] [-v] {C,CC,CCC} ...

positional arguments:
  {C,CC,CCC}

optional arguments:
  -h, --help     show this help message and exit
  -c             A BB [optional pi]
  -v, --verbose  A BB erbosity


'A BB':
-------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.A_BB' id='139714598483088'>
'A BB -v':
----------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.A_BB' id='139714598483088'>
'A BB -c':
----------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.A_BB' id='139714598483088'>
'A BB -v -c':
-------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.A_BB' id='139714598483088'>


NODE :: 'A BB C':
-------------------
usage: A BB C [-h] [-c] [-v]

optional arguments:
  -h, --help     show this help message and exit
  -c             A BB C [optional pi]
  -v, --verbose  A BB C verbosity


'A BB C':
---------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.A_BB_C' id='139714598483280'>
'A BB C -v':
------------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.A_BB_C' id='139714598483280'>
'A BB C -c':
```

```
------------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.A_BB_C' id='139714598483280'>
'A BB C -v -c':
---------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.A_BB_C' id='139714598483280'>


NODE :: 'A BB CC':
--------------------
usage: A BB CC [-h] [-c] [-v]

optional arguments:
  -h, --help     show this help message and exit
  -c             A BB CC [optional pi]
  -v, --verbose  A BB CC verbosity


'A BB CC':
----------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.A_BB_CC' id='139714598483472'>
'A BB CC -v':
-------------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.A_BB_CC' id='139714598483472'>
'A BB CC -c':
-------------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.A_BB_CC' id='139714598483472'>
'A BB CC -v -c':
----------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.A_BB_CC' id='139714598483472'>


NODE :: 'A BB CCC':
---------------------
usage: A BB CCC [-h] [-c] [-v]

optional arguments:
  -h, --help     show this help message and exit
  -c             A BB CCC [optional pi]
  -v, --verbose  A BB CCC verbosity


'A BB CCC':
-----------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.A_BB_CCC' id='139714598483664'>
'A BB CCC -v':
--------------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.A_BB_CCC' id='139714598483664'>
'A BB CCC -c':
--------------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.A_BB_CCC' id='139714598483664'>
'A BB CCC -v -c':
-----------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.A_BB_CCC' id='139714598483664'>
```

```
1 from parsearg.examples.a_aa import main
2 main()
```

```
NODE :: 'A':
--------------
usage:  A [-h] [-c] [-v] {B,BBB,BB} ...

positional arguments:
  {B,BBB,BB}

optional arguments:
```

```
  -h, --help     show this help message and exit
  -c             A [optional pi]
  -v, --verbose  A verbosity


'A':
----
        args: {'c': None, 'verbose': False}
        <Mock name='mock.A' id='139714598483904'>
'A -v':
-------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.A' id='139714598483904'>
'A -c':
-------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.A' id='139714598483904'>
'A -v -c':
----------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.A' id='139714598483904'>


NODE :: 'A B':
----------------
usage: A B [-h] [-c] [-v]

optional arguments:
  -h, --help     show this help message and exit
  -c             A B [optional pi]
  -v, --verbose  A B verbosity


'A B':
------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.A_B' id='139714598562448'>
'A B -v':
---------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.A_B' id='139714598562448'>
'A B -c':
---------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.A_B' id='139714598562448'>
'A B -v -c':
------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.A_B' id='139714598562448'>


NODE :: 'A BB':
-----------------
usage: A BB [-h] [-c] [-v] {C,CC,CCC} ...

positional arguments:
  {C,CC,CCC}

optional arguments:
  -h, --help     show this help message and exit
  -c             A BB [optional pi]
  -v, --verbose  A BB erbosity


'A BB':
-------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.A_BB' id='139714598562592'>
'A BB -v':
----------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.A_BB' id='139714598562592'>
'A BB -c':
----------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.A_BB' id='139714598562592'>
'A BB -v -c':
```

```
-------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.A_BB' id='139714598562592'>


NODE :: 'A BB C':
-------------------
usage: A BB C [-h] [-c] [-v]

optional arguments:
  -h, --help     show this help message and exit
  -c             A BB C [optional pi]
  -v, --verbose  A BB C verbosity


'A BB C':
---------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.A_BB_C' id='139714598562256'>
'A BB C -v':
------------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.A_BB_C' id='139714598562256'>
'A BB C -c':
------------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.A_BB_C' id='139714598562256'>
'A BB C -v -c':
---------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.A_BB_C' id='139714598562256'>


NODE :: 'A BB CC':
--------------------
usage: A BB CC [-h] [-c] [-v]

optional arguments:
  -h, --help     show this help message and exit
  -c             A BB CC [optional pi]
  -v, --verbose  A BB CC verbosity


'A BB CC':
----------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.A_BB_CC' id='139714598562496'>
'A BB CC -v':
-------------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.A_BB_CC' id='139714598562496'>
'A BB CC -c':
-------------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.A_BB_CC' id='139714598562496'>
'A BB CC -v -c':
----------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.A_BB_CC' id='139714598562496'>


NODE :: 'A BB CCC':
---------------------
usage: A BB CCC [-h] [-c] [-v]

optional arguments:
  -h, --help     show this help message and exit
  -c             A BB CCC [optional pi]
  -v, --verbose  A BB CCC verbosity


'A BB CCC':
-----------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.A_BB_CCC' id='139714598562016'>
'A BB CCC -v':
--------------
```

```
        args: {'c': None, 'verbose': True}
        <Mock name='mock.A_BB_CCC' id='139714598562016'>
'A BB CCC -c':
--------------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.A_BB_CCC' id='139714598562016'>
'A BB CCC -v -c':
-----------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.A_BB_CCC' id='139714598562016'>


NODE :: 'AA':
---------------
usage:  AA [-h] [-c] [-v] {B,BBB,BB} ...

positional arguments:
  {B,BBB,BB}

optional arguments:
  -h, --help     show this help message and exit
  -c             AA [optional pi]
  -v, --verbose  AA verbosity


'AA':
-----
        args: {'c': None, 'verbose': False}
        <Mock name='mock.AA' id='139714598576960'>
'AA -v':
--------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.AA' id='139714598576960'>
'AA -c':
--------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.AA' id='139714598576960'>
'AA -v -c':
-----------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.AA' id='139714598576960'>


NODE :: 'AA B':
-----------------
usage: AA B [-h] [-c] [-v]

optional arguments:
  -h, --help     show this help message and exit
  -c             AA B [optional pi]
  -v, --verbose  AA B verbosity


'AA B':
-------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.AA_B' id='139714598576912'>
'AA B -v':
----------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.AA_B' id='139714598576912'>
'AA B -c':
----------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.AA_B' id='139714598576912'>
'AA B -v -c':
-------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.AA_B' id='139714598576912'>


NODE :: 'AA BB':
------------------
usage: AA BB [-h] [-c] [-v] {C,CC,CCC} ...

positional arguments:
  {C,CC,CCC}
```

```
optional arguments:
  -h, --help     show this help message and exit
  -c             AA BB [optional pi]
  -v, --verbose  AA BB verbosity


'AA BB':
--------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.AA_BB' id='139714598576864'>
'AA BB -v':
-----------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.AA_BB' id='139714598576864'>
'AA BB -c':
-----------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.AA_BB' id='139714598576864'>
'AA BB -v -c':
--------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.AA_BB' id='139714598576864'>


NODE :: 'AA BB C':
--------------------
usage: AA BB C [-h] [-c] [-v]

optional arguments:
  -h, --help     show this help message and exit
  -c             AA BB C [optional pi]
  -v, --verbose  AA BB C verbosity


'AA BB C':
----------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.AA_BB_C' id='139714598577824'>
'AA BB C -v':
-------------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.AA_BB_C' id='139714598577824'>
'AA BB C -c':
-------------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.AA_BB_C' id='139714598577824'>
'AA BB C -v -c':
----------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.AA_BB_C' id='139714598577824'>


NODE :: 'AA BB CC':
---------------------
usage: AA BB CC [-h] [-c] [-v]

optional arguments:
  -h, --help     show this help message and exit
  -c             AA BB CC [optional pi]
  -v, --verbose  AA BB CC verbosity


'AA BB CC':
-----------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.AA_BB_CC' id='139714598577536'>
'AA BB CC -v':
--------------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.AA_BB_CC' id='139714598577536'>
'AA BB CC -c':
--------------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.AA_BB_CC' id='139714598577536'>
'AA BB CC -v -c':
-----------------
```

```
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.AA_BB_CC' id='139714598577536'>


NODE :: 'AA BB CCC':
----------------------
usage: AA BB CCC [-h] [-c] [-v]

optional arguments:
  -h, --help     show this help message and exit
  -c             AA BB CCC [optional pi]
  -v, --verbose  AA BB CCC verbosity


'AA BB CCC':
------------
        args: {'c': None, 'verbose': False}
        <Mock name='mock.AA_BB_CCC' id='139714598576384'>
'AA BB CCC -v':
---------------
        args: {'c': None, 'verbose': True}
        <Mock name='mock.AA_BB_CCC' id='139714598576384'>
'AA BB CCC -c':
---------------
        args: {'c': 3.141593, 'verbose': False}
        <Mock name='mock.AA_BB_CCC' id='139714598576384'>
'AA BB CCC -v -c':
------------------
        args: {'c': 3.141593, 'verbose': True}
        <Mock name='mock.AA_BB_CCC' id='139714598576384'>
```