UCLouvain
Institut de recherche en mathématique et physique
Centre de Cosmologie, Physique des Particules et Phénoménologie

MCnet

Re-create the shell experience

Olivier Mattelaer

User Interface

- Crucial element for any application
  - ➡ Often consider as not crucial
  - ➡ But can also lead to intense debate
- First contact of the user with your code
  - ➡ Intuitive?
- For developer important question is also
  - ➡ Flexible?

This Talk

- Description of the user interface of MG5aMC
  - ➡ Description of the native python module on which it is based
- Goal of the talk
  - ➡ Give you idea
  - ➡ Show you that this is easy

CLI mechanism

Parameters

```
[tmp]$ bzr commit -m "example of partial commit" ./directory
```

Interactive session

```
Do you want to start a new computation (y/n)
>
```

git rebase -i

https://en.wikipedia.org/wiki/Command-line_interface

Pro/Cons

- Interactive session
  - ➡ is more user friendly
  - ➡ can be more difficult to script
    - Bad if a long series of inter-independent question

Mixing the two

- Create a python shell interface
  - ➡ Support a list of command (with arguments)
  - ➡ Smart autocompletion
  - ➡ Allow various mode
    - Tutorial mode
    - Smart help
- Python has a basic built-in package for that:
  - ➡ cmd
- More advanced package does exists but not standard: cmd2

cmd module

```
import cmd
import readline

class myinterface(cmd.Cmd):

    def do_helloworld(self, line):
        print("Hello world " + line)

    def help_helloworld(self):
        print("printing hello world")

    def complete_helloworld(self, text, line, begidx, endidx):
        possibilities = ["user", "olivier", "mattelaer"]
        return  [ f for f in possibilities if f.startswith(text)]

if __name__ == "__main__":
    #readline.parse_and_bind("tab: complete") # for UNIX
    readline.parse_and_bind("bind ^I rl_complete") # for MAC
    interface = myinterface()
    interface.cmdloop()
```

cmd module

```
def do_helloworld(self, line):
    print("Hello world " + line)
```

- Command in the interface starts with “do_”
  - ➡This defines a command hello world

```
def help_helloworld(self):
    print("printing hello world")
```

- Dedicated help command
  - Default is to use the Documentation string

```
def complete_helloworld(self, text, line, begidx, endidx):
    possibilities = ["user", "olivier", "mattelaer"]
    return  [ f for f in possibilities if f.startswith(text)]
```

- Defined auto-completion

cmd module

```
if __name__ == "__main__":
    #readline.parse_and_bind("tab: complete") # for UNIX
    readline.parse_and_bind("bind ^I rl_complete") # for MAC
    interface = myinterface()
    interface.cmdloop()
```

- Running the code
  - ➡ Configure readline for auto-completion
    - Different for different version of readline
  - ➡ Scriptable
    - either via pipe (not advised)
    - Via command file

Demo

```
[tmp]$ python helloworld.py
(Cmd) hel
helloworld help
(Cmd) helloworld
mattelaer olivier   user
(Cmd) helloworld olivier
Hello world olivier
(Cmd)
```

Conclusion

- This is the basic feature
- “Cmd” offer more but not that much
  - ➡ MG5aMC creates new feature to handle that
    - Better auto-completion, transcript,...
  - ➡ cmd2 module actually covers most of our needs and many other cool functionality
    - Argparse definition of command
      - (automatic help/completion)
    - User defined variable/alias/macro

MG5aMC customisation

- most of the customisation are available within the cmd2 module

  ➡ We do not depend on cmd2 to avoid to handle dependencies

- easier and nicer auto-completion.
- possibility to have sub-interface mode/ question mode

  ➡ Lot of care for scripting

- transcript

Cmd2

- User defined Alias/macro
- Variable
- argparse function
  - ➡ Automatic help and auto-completion
- Piping/redirection within the interface
- ...

https://pypi.org/project/cmd2/

Python code

```
import cmd
import readline

class myinterface(cmd.Cmd):

    def do_helloworld(self, line):
        print("Hello world " + line)

    def help_helloworld(self):
        print("printing hello world")

    def complete_helloworld(self, text, line, begidx, endidx):
        possibilities = ["user", "olivier", "mattelaer"]
        return  [ f for f in possibilities if f.startswith(text)]

if __name__ == "__main__":
    #readline.parse_and_bind("tab: complete") # for UNIX
    readline.parse_and_bind("bind ^I rl_complete") # for MAC
    interface = myinterface()
    interface.cmdloop()
```