

# ipdb: How to inspect complex structures

Python comes with a debugger called pdb. The python debugger is very powerful and really facilitates troubleshooting code. You can find the official documentation [here](#). `ipdb` builds on top of `ipython` and `pdb` to give users a more interactive experience.

This how to isn't very extensive, it's just a very quick and dirty demo to show some of it's capabilities and how to use it with `nornir`. It doesn't assume previous knowledge but doesn't spend too much on explanations either so it expects users to give enough material for further investigation.

## Installing ipdb

First, you will need to install `ipython`, follow the official guide to do so, then you will need to install `ipdb`. You can install the latter via `pip`:

```
pip install ipdb
```

## Inspecting results

For the sake of the demo I have written some code that returns a result. The code is not important but what it does is:

1. Read a yaml file containing a bunch of users we want to have configured on our devices
2. Connect to a couple of network devices and get the users configured
3. Check with users are already configured, which ones we want and compute two lists; a list with the users we need to configure and another list with the ones we need to remove.
4. Finally, we pass those lists to a template and we generate some configuration.

Let's start by insterting a break point right after we get the result (line 56):

```
nvim demo.py
1 demo.py
18 # if they are not we have to remove them
19 users_to_remove = []
20 for user in state_users.result["users"]:
21     if user not in desired_users:
22         users_to_remove.append(user)
23
24 # we render the template for the platform passing
25 # desired_users and users_to_remove
26 task.run(
27     task=template_file,
28     path=f"5_manage_users/templates/{task.host.platform}",
29     template="users.j2",
30     desired_users=desired_users,
31     remove_users=users_to_remove,
32     # severity_level=logging.DEBUG,
33 )
34
35
36 def main():
37     nr = InitNornir(
38         inventory={
39             "options": {
40                 "host_file": "5_manage_users/inventory/hosts.yaml",
41                 "group_file": "5_manage_users/inventory/groups.yaml",
42                 "defaults_file": "5_manage_users/inventory/defaults.yaml",
43             }
44         }
45     )
46
47     # we load from a yaml file the users we want
48     yaml = ruamel.yaml.YAML()
49     with open("5_manage_users/data/users.yaml", "r") as f:
50         desired_users = yaml.load(f.read())
51
52     spines = nr.filter(role="spine")
53
54     # we call manage_users passing the users we loaded from the yaml file
55     r = spines.run(task=manage_users, desired_users=desired_users)
56     import ipdb; ipdb.set_trace()
57     print_result(r)
58
59
60 if __name__ == "__main__":
master> demo.py 33/55/61
```

If we execute the script, we will get a shell right in that point of the code.

```
IPython: normir-workshop/notebooks
(nornir-workshop) → notebooks git:(master) ✕ nvim demo.py
(nornir-workshop) → notebooks git:(master) ✕ python demo.py
> /home/dbarroso/workspace/dravetech/nornir-workshop/notebooks/demo.py(57)main()
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58

ipdb> l
52 spines = nr.filter(role="spine")
53
54 # we call manage_users passing the users we loaded from the yaml file
55 r = spines.run(task=manage_users, desired_users=desired_users)
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58
59
60 if __name__ == "__main__":
61     main()

ipdb> █
```

Now we can start using python code to figure out how the object works:

```
IPython: normir-workshop/notebooks
(nornir-workshop) → notebooks git:(master) ✕ nvim demo.py
(nornir-workshop) → notebooks git:(master) ✕ python demo.py
> /home/dbarroso/workspace/dravetech/nornir-workshop/notebooks/demo.py(57)main()
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58

ipdb> l
52 spines = nr.filter(role="spine")
53
54 # we call manage_users passing the users we loaded from the yaml file
55 r = spines.run(task=manage_users, desired_users=desired_users)
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58
59
60 if __name__ == "__main__":
61     main()

ipdb> print(r)
AggregatedResult (manage_users): {'spine00.bma': MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"], 'spine01.bma': MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]}
ipdb>
```

The output above suggests the object is of the type `AggregatedResult`. The documentation surely will explain how it works but we can keep playing with it a bit. The output also suggests that the object might be a dictionary-like object with keys `spine00.bma` and `spine01.bma`. Let's see keep digging.

```
IPython: normir-workshop/notebooks
(nornir-workshop) → notebooks git:(master) x nvim demo.py
(nornir-workshop) → notebooks git:(master) x python demo.py
> /home/dbarroso/workspace/dravetech/nornir-workshop/notebooks/demo.py(57)main()
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58

ipdb> l
52 spines = nr.filter(role="spine")
53
54 # we call manage_users passing the users we loaded from the yaml file
55 r = spines.run(task=manage_users, desired_users=desired_users)
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58
59
60 if __name__ == "__main__":
61     main()

ipdb> print(r)
AggregatedResult (manage_users): {'spine00.bma': MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"], 'spine01.bma': MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]}
ipdb> r.
```

clear()	failed_hosts	items()	pop()	setdefault()
copy()	fromkeys()	keys()	popitem()	update()
failed	get()	name	raise_on_error()	values()

Something interesting is that you can press `<tab>` to get autocompletion. In this case `r.<tab>` shows us the available methods that the object provides. Seeing methods like `keys`, `items`, etc... plus the output of `print(r)` seems to confirm our theory the object might be a dict-like object.

Let's put the theory to test:

```
IPython: normir-workshop/notebooks
(nornir-workshop) → notebooks git:(master) x nvim demo.py
(nornir-workshop) → notebooks git:(master) x python demo.py
> /home/dbarroso/workspace/dravetech/nornir-workshop/notebooks/demo.py(57)main()
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58

ipdb> l
52 spines = nr.filter(role="spine")
53
54 # we call manage_users passing the users we loaded from the yaml file
55 r = spines.run(task=manage_users, desired_users=desired_users)
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58
59
60 if __name__ == "__main__":
61     main()

ipdb> print(r)
AggregatedResult (manage_users): {'spine00.bma': MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"], 'spine01.bma': MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]}
ipdb> print(r["spine00.bma"])
MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]
ipdb> spine00 = r["spine00.bma"]
ipdb> spine00
MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]
ipdb> █
```

Ok, looks like we were right. Notice we extracted a `MultiResult` that belonged to the key `spine00.bma` and assigned it to a variable for further inspection. This new `MultiResult` object looks like a list. Let's use `<tab>` again to see which methods provide:

```

IPython: normir-workshop/notebooks
(nornir-workshop) → notebooks git:(master) x nvim demo.py
(nornir-workshop) → notebooks git:(master) x python demo.py
> /home/dbarroso/workspace/dravetech/nornir-workshop/notebooks/demo.py(57)main()
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58

ipdb> l
52 spines = nr.filter(role="spine")
53
54 # we call manage_users passing the users we loaded from the yaml file
55 r = spines.run(task=manage_users, desired_users=desired_users)
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58
59
60 if __name__ == "__main__":
61     main()

ipdb> print(r)
AggregatedResult (manage_users): {'spine00.bma': MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"], 'spine01.bma': MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]}
ipdb> print(r["spine00.bma"])
MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]
ipdb> spine00 = r["spine00.bma"]
ipdb> spine00
MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]
ipdb> spine00.

```

append()	copy()	failed	name	remove()
changed	count()	index()	pop()	reverse()
clear()	extend()	insert()	raise_on_error()	sort()

With methods like `append`, `extend`, etc., this surely looks like a list. Based on the previous output the element `1` seems to be the result of the task `napalm_get`, let's see if we can extract it:

```
IPython: normir-workshop/notebooks
(nornir-workshop) → notebooks git:(master) x nvim demo.py
(nornir-workshop) → notebooks git:(master) x python demo.py
> /home/dbarroso/workspace/dravetech/nornir-workshop/notebooks/demo.py(57)main()
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58

ipdb> l
52 spines = nr.filter(role="spine")
53
54 # we call manage_users passing the users we loaded from the yaml file
55 r = spines.run(task=manage_users, desired_users=desired_users)
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58
59
60 if __name__ == "__main__":
61     main()

ipdb> print(r)
AggregatedResult (manage_users): {'spine00.bma': MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"], 'spine01.bma': MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]}
ipdb> print(r["spine00.bma"])
MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]
ipdb> spine00 = r["spine00.bma"]
ipdb> spine00
MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]
ipdb> ng_result = spine00[1]
ipdb> ng_result
Result: "napalm_get"
ipdb> █
```

Great, now `ng_result` has the result of running the task `napalm_get`. Let's keep digging:

```
IPython: nornir-workshop/notebooks
(nornir-workshop) → notebooks git:(master) x nvim demo.py
(nornir-workshop) → notebooks git:(master) x python demo.py
> /home/dbarroso/workspace/dravetech/nornir-workshop/notebooks/demo.py(57)main()
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58

ipdb> l
52 spines = nr.filter(role="spine")
53
54 # we call manage_users passing the users we loaded from the yaml file
55 r = spines.run(task=manage_users, desired_users=desired_users)
56 import ipdb; ipdb.set_trace()
--> 57 print_result(r)
58
59
60 if __name__ == "__main__":
61     main()

ipdb> print(r)
AggregatedResult (manage_users): {'spine00.bma': MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"], 'spine01.bma': MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]}
ipdb> print(r["spine00.bma"])
MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]
ipdb> spine00 = r["spine00.bma"]
ipdb> spine00
MultiResult: [Result: "manage_users", Result: "napalm_get", Result: "template_file"]
ipdb> ng_result = spine00[1]
ipdb> ng_result
Result: "napalm_get"
ipdb> ng_result.


| changed   | failed | result         | stdout |
|-----------|--------|----------------|--------|
| diff      | host   | severity_level |        |
| exception | name   | stderr         |        |


```

Voila, this new object has attributes like `changed`, `result`, `diff`, etc., so looks like we finally manage to dig down our object and figure out how get what we want.