

Participation à la vie de la communauté Open Source

Documentation technique

Répertoire GitHub : https://github.com/Hvvvrd/Project_6_OpenClassrooms

Table des matières

Présentation du projet.....	3
Présentation du scénario.....	4
Notre script d'automatisation.....	5
Présentation de Python.....	5
Présentation de dhcp_dns.py.....	5
Présentation de l'en-tête de notre script.....	7
Sys.....	7
Getopt.....	7
Fileinput.....	8
Os.....	8
Os.path.....	8
YAML.....	8
Configparser.....	8
Socket.....	9
Partie «complétion des éléments ».....	9
Partie « DHCP ».....	10
Partie « DNS ».....	12
Partie « Main ».....	13
GitHub.....	15
Présentation de GitHub.....	15
Notre répertoire GitHub.....	15

Présentation du projet

Dans le cadre du projet 6 du parcours Administrateur Infrastructure & Cloud d'Openclassrooms dont l'intitulé est « Participez à la vie de la communauté Open Source », il est demandé :

- D'identifier une tâche complexe ou un ensemble de tâches d'administration qu'on souhaite automatiser
- De créer un répertoire d'hébergement sur GitHub pour notre projet
- De rédiger notre tâche automatisée sous forme de script ou de module Ansible
- De mettre à disposition notre code à la communauté
- De mettre à disposition une documentation du fonctionnement de notre code pour d'éventuels contributeurs.

Les livrables attendus sont la mise à disposition de l'URL de notre répertoire GitHub public hébergeant notre code et sa documentation.

Présentation du scénario

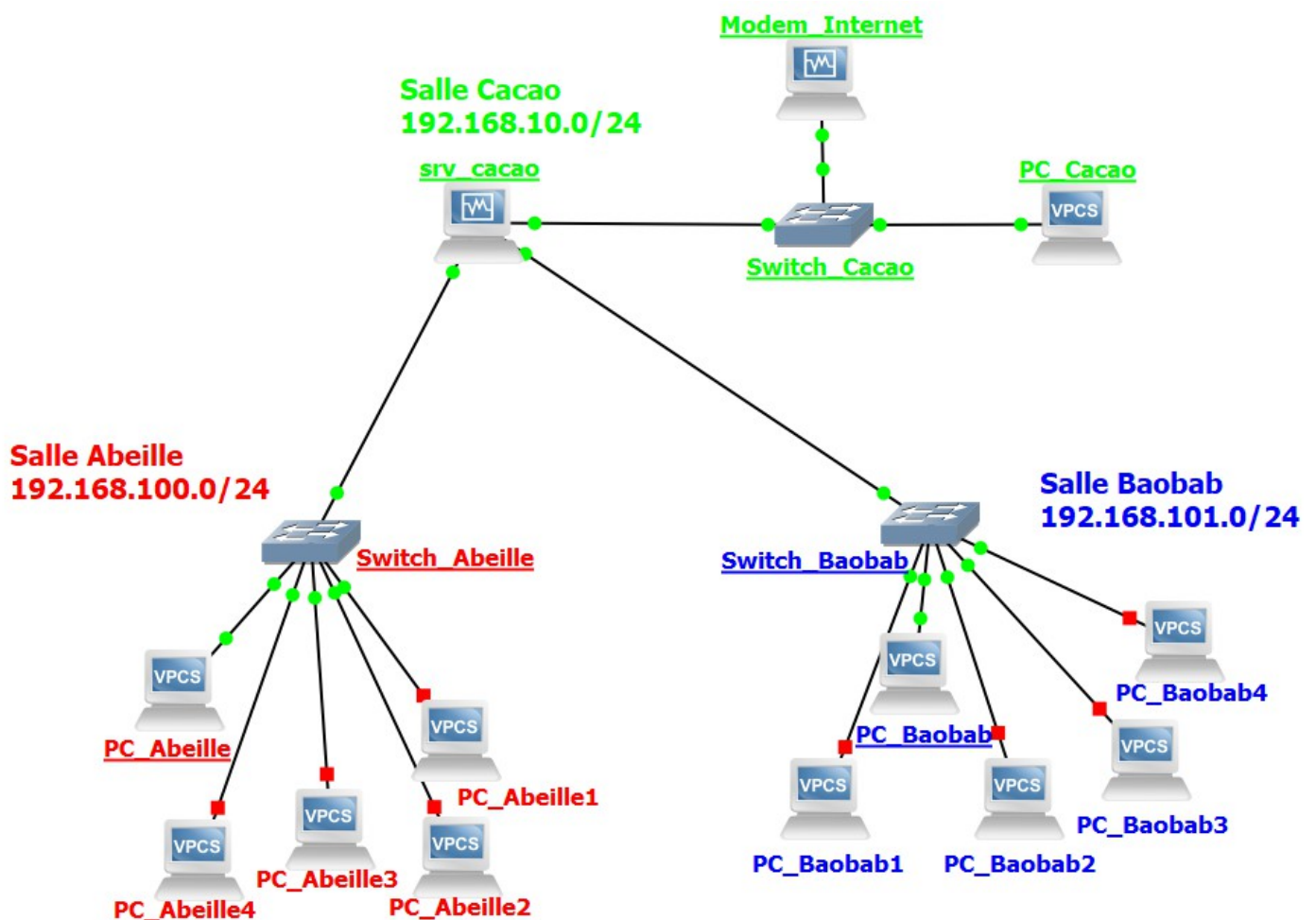
La démonstration de notre projet pour OpenClassrooms se déroule sur une maquette GNS3 qui nous permet de simuler l'architecture réseau décrite ci-dessous.

Nous disposons de trois salles informatiques :

- La salle Cacao qui se compose de notre serveur d'administration de notre réseau faisant office de serveur DHCP et DNS. Se trouve dans cette salle notre modem Internet et un poste de travail.
- La salle Abeille composée d'un ensemble de poste de travail administrés depuis Cacao.
- La salle Baobab qui comme la salle Abeille est simplement composé d'un ensemble de poste de travail administrés depuis Cacao.

Notre infrastructure se compose d'un réseau principal « **Cacao** » en 192.168.10.0/24 et des sous-réseaux propres aux salles « **Abeille** » en 192.168.100.0/24 et « **Baobab** » en 192.168.101.0/24.

L'objectif est de réaliser un script en Python qui automatiserait la configuration des services DHCP et DNS sur l'ensemble de notre infrastructure.



Notre script d'automatisation

Présentation de Python



Le langage Python est un langage de programmation open source multi-plateformes et orienté objet. Grâce à des bibliothèques spécialisées, Python s'utilise pour de nombreuses situations comme le développement logiciel, l'analyse de données, ou la gestion d'infrastructures.

Langage de programmation interprété, Python permet l'exécution du code sur n'importe quel ordinateur. Utilisable aussi bien par des programmeurs débutants qu'experts, Python permet de créer des programmes de manière simple et rapide.

Présentation de *dhcp_dns.py*

L'objectif de ce script est de configurer en quelques instants un serveur DHCP et DNS sur une machine fonctionnant sous Linux.

Il propose deux modes de lancement :

- mode interactif : `sudo dhcp_dns.py -i`
- mode argument : `sudo dhcp_dns.py -d <domain> -a <addr ip> -n <server name> -m <subnet mask> -o <option dns> -r <nombre de sous-réseaux> --interfaces=<"interface1 interface2 ...">`

La différence entre les deux modes étant que le mode argument la valeur des variables est mentionné directement dans les arguments tandis que, le mode interactif proposera de saisir les valeurs dans le terminal, au fur et à mesure, de la configuration du script, en réponse à une interaction avec l'utilisateur, sous-forme de questions. Pour cela, il suffira de saisir dans le terminal la valeur souhaitée et de faire « entrer ».

Les paramètres que nous retrouvons pour notre script `dhcp_dns.py` sont :

- h : aide
- i : lancer le mode interactif
- d : rentrer votre nom de domaine dns (ex : mon-entreprise.fr)
- a : rentrer votre adresse IP (ex : 10.0.1.2)
- n : rentrer votre nom de serveur dhcp (ex : ubuntu.lan)
- m : rentrer votre masque pour le serveur (ex : 255.255.255.0)
- o : rentrer les options dns (ex : 8.8.8.8, 1.1.1.1)
- r : rentrer le nombre de sous réseau que vous avez configuré dans le fichier `res.ini`
- interfaces : rentrer vos interfaces d'écoute du serveur dhcp (ex : "enp0s3 enp0s8")

Notre script `dhcp_dns.py` est accompagné d'un second fichier : `res.ini`. Ce second fichier doit se trouver impérativement dans le même répertoire que `dhcp_dns.py`. Dedans, il sera déclarer nos sous-réseaux de la façons suivante, en prenant exemple ici pour 2 sous-réseaux :

```
[reseau0]
subnet: 10.0.0.0
netmask: 255.255.255.0
broadcast: 10.0.0.255
ntp: 10.0.0.1
routers: 10.0.0.1
pool: 10.0.0.2 10.0.0.254
```

```
[reseau1]
subnet: 10.0.0.0
netmask: 255.255.255.0
broadcast: 10.0.0.255
ntp: 10.0.0.1
routers: 10.0.0.1
pool: 10.0.0.2 10.0.0.254
```

A noter, si nous prenons notre exemple pour 2 sous-réseaux, que lors de l'exécution du mode argument de notre script, l'argument `-r <nombre de sous-réseaux>` sera donc `-r 2`.

Présentation de l'en-tête de notre script

```
1  #!/usr/bin/python
2  #-*-coding: utf-8 -*-
3
4  import sys, getopt
5  import fileinput
6  import os
7  from socket import *
8  import yaml
9  import os.path
10 import configparser
--
```

Dans notre en-tête, nous distinguons 3 éléments :

- `#!/usr/bin/env python` qui correspond au Shebang d'exécution des scripts Python sur Linux.
- `-*-coding utf-8-* -` qui correspond à l'encodage d'intégration de notre programme.
- et enfin, l'énumération des modules a importé dont la description est détaillée ci-dessous.

Sys

Ce module fournit un accès à certaines variables utilisées et maintenues par l'interpréteur, et à des fonctions interagissant fortement avec ce dernier.

Il contient des informations système, comme la version de Python que vous utilisez (`sys.version` ou `sys.version_info`) et des options système comme la profondeur maximale de récursion autorisée (`sys.getrecursionlimit()` et `sys.setrecursionlimit()`).

`sys.modules` est un dictionnaire qui contient tous les modules qui ont été importés depuis que Python a été démarré. La clé est le nom de module, la valeur est l'objet module. Notez que cela comprend plus que les modules que votre programme a importé. Python charge certains modules au démarrage et si vous êtes dans une IDE Python, `sys.modules` contient tous les modules importés par tous les programmes que vous avez exécutés dans l'IDE.

Getopt

Ce module aide les scripts à analyser les arguments de ligne de commande contenus dans `sys.argv`. Il prend en charge les mêmes conventions que la fonction UNIX `getopt()` (y compris les significations spéciales des arguments de la forme `-` et `--`). De longues options similaires à celles prises en charge par le logiciel GNU peuvent également être utilisées via un troisième argument facultatif.

Fileinput

Ce module offre une classe auxiliaire et des fonctions pour lire facilement l'entrée standard ou bien les fichiers d'une liste.

Ce code traite une à une les lignes des fichiers de `sys.argv[1:]`. Si cette liste est vide (pas d'argument en ligne de commande), il lit l'entrée standard.

On peut aussi passer la liste des fichiers comme argument à `input()`, voire un nom de fichier unique.

Os

Le module `os` en Python permet d'interagir avec les fonctionnalités du système d'exploitation et d'accéder aux informations. De plus, le module `os` nous permet de travailler avec les fichiers et les répertoires. Il est indépendant par rapport au système d'exploitation de la machine. Ce qui signifie que ce module peut fonctionner sur n'importe quelle système d'exploitation.

Os.path

Ce module implémente certaines fonctions utiles sur le nom des chemins. Il est toujours le module de chemin adapté au système d'exploitation sur lequel Python tourne, et donc adapté pour les chemins locaux.

Pour lire ou écrire des fichiers, voir `open()`, et pour accéder au système de fichier, voir le module `os`.

YAML

Le module `YAML` va nous permettre d'analyser des fichiers `YAML`.

`YAML` est un format de représentation de données par sérialisation Unicode. Il reprend des concepts d'autres langages comme `XML`. Son objet est de représenter des informations plus élaborées que le simple `CSV` en gardant cependant une lisibilité presque comparable, et bien plus grande en tout cas que du `XML`.

Il est principalement utilisé dans les fichiers de configurations car facile de lecture et de compréhension.

Configparser

Ce module fournit la classe ConfigParser. Cette classe implémente un langage de configuration basique, proche de ce que l'on peut trouver dans les fichiers *INI* de Microsoft Windows.

Dans notre script Python, celui-ci vient parser notre fichier de paramètres res.ini.

En bref, chaque fichier INI est constitué de sections, chacune des sections comprenant des clés associées à des valeurs. Les classes du module configparser peuvent écrire et lire de tels fichiers.

Socket

Le module socket de Python permet de gérer les connexions par socket.

Sur Python, on définira un socket comme un objet qui permet d'ouvrir une connexion avec une machine, locale ou distante, et d'échanger avec elle.

En anglais un socket est un "trou" qui laisse passer des choses, comme une prise électrique, l'emplacement du processeur, ou une bouche -on va s'arrêter la pour les exemples -

Le socket est donc dans notre cas une association au niveau de l'OS entre un programme qui tourne en boucle et le port de la machine qui lui a été dédié. On dit d'ailleurs que le programme écoute le port qui lui a été réservé. Il écoute le port et répond aux demandes faites par ce port.

Partie «complétion des éléments »

```
12 #####
13 #####Partie complétion des éléments #####
14 #####
15
16 def replaceAll(file,searchExp,replaceExp):
17     for line in fileinput.input(file, inplace=1):
18         if searchExp in line:
19             line = line.replace(searchExp,replaceExp)
20             sys.stdout.write(line)
21
```

La définition de cette partie a pour rôle de fournir le remplacement dans le script aux endroits adéquats des éléments saisis lors de l'exécution en *mode interactif* ou en *mode argument* du script.

On note l'utilisation du module fileinput qui en effet permet de lire les entrées standards et les traitent une à une.

On peut voir l'utilisation de la fonction replaceAll dans les parties de configuration DHCP et DNS.

Partie « DHCP »

```
22 #####
23 #####Partie configuration du DHCP #####
24 #####
25
26 def dhcp_conf(server_name,subnet_mask,domain,option_dns,sous_res,interfaces):
27     os.system("apt-get install isc-dhcp-server")
28
29     fichier = open("/etc/dhcp/dhcpd.conf","w")
30     fichier.write("##### Option générale par défaut #####\n")
31     fichier.write("\n### RÉSEAU #####\n")
32     fichier.write("\nserver-name \"\"+server_name+\"\";") #nom du serveur dns
33     fichier.write("\nauthoritative;")
34     fichier.write("\noption subnet-mask "+subnet_mask+";")
35     fichier.write("\noption domain-name \"\"+ domain +\"\";")
36     fichier.write("\noption domain-name-servers "+ option_dns +";")
37     fichier.write("\nddns-update-style none;")
38     fichier.write("\ndefault-lease-time 3600;")
39     fichier.write("\nmax-lease-time 7200;")
40     fichier.write("\nlog-facility local7;\n")
41     fichier.write("\n##### RÉSEAUX #####\n")
42     fichier.write("\n## Déclaration sous réseaux")
43
44     interfaces= ''
45     config = configparser.RawConfigParser() # On crée un nouvel objet "config"
46     config.read('res.ini')
47
```

On définit la fonction `dhcp_conf` qui possède en paramètres les différents éléments configurant notre service DHCP et qui seront récupérés dans la partie main lors de l'exécution par l'utilisateur en *mode interactif* ou *mode argument* du script.

L'utilisation du module `os` avec la fonction `system` permet de lancer l'installation du service *isc-dhcp-server* via la commande *apt-get install* du terminal. Le script ayant été lancé avec un `sudo`, nous ne rencontrons pas de problème de droit ici.

Suite à l'installation de *isc-dhcp-server*, on enregistre en valeur à la variable `fichier`, le fichier de configuration « `/etc/dhcp/dhcpd.conf` » que l'on met en mode « `w` » pour `write` soit écriture. Le fichier `dhcpd.conf` est le fichier de configuration du service DHCP.

Les lignes 30 à 42 de notre script vont venir écrire dans le fichier de configuration les éléments attendus et les paramètres correspondants. On note que la syntaxe mettant entre des symboles « `+` » nos paramètres de la fonction `dhcp_conf` permet d'indiquer qu'ils seront remplacés lors de l'exécution du script par leurs valeurs propres. *Exemple* « `+subnet_mask+` » ligne 34.

A partir de la Ligne 44, on note l'utilisation du module `configparser` dont le rôle est de venir parser notre fichier de configuration `res.ini` accompagnant notre script.

```

48     i=0
49     for i in range(0,sous_res):
50
51         reseau = str("reseau"+str(i))
52         subnet = config.get(reseau,'subnet')
53         netmask = config.get(reseau,'netmask')
54         broadcast = config.get(reseau,'broadcast')
55         ntp = config.get(reseau,'ntp')
56         routers = config.get(reseau,'routers')
57         pool = config.get(reseau,'pool')
58
59         fichier.write("\nsubnet "+subnet+" netmask "+netmask+" {")
60         fichier.write("\n  option domain-name \""+domain+"\";")
61         fichier.write("\n  option broadcast-address "+broadcast+";")
62         fichier.write("\n  option ntp-servers "+ntp+";")
63         fichier.write("\n  option routers "+routers+";")
64         fichier.write("\n  range "+pool+";")
65         fichier.write("\n  ping-check = 1;")
66         fichier.write("\n}\n")
67     fichier.close()
68
69     replaceAll("/etc/default/isc-dhcp-server","INTERFACESv4=\"\"","INTERFACESv4=\""+interfaces+"\"")
70     os.system("service isc-dhcp-server restart")
71

```

Une boucle for est générée en ligne 49 en prévision du nombre de réseaux configurés dans le res.ini.

Les valeurs de res.ini sont enregistrées dans les variables correspondantes de la ligne 51 à 57 afin d'être ensuite correctement ré-employées dans notre script grâce au parsing.

Les lignes 59 jusqu'à la fermeture fichier de configuration de notre DHCP en ligne 67, viennent finir la configuration de notre service en remplaçant les éléments récupérées directement dans les lignes de notre fichier. Ainsi, contrairement aux éléments sous la syntaxe de remplacement entre deux « + » précédemment vus, ceux entre les lignes 59 et 63, sont des éléments qui sont récupérés dans le res.ini, tandis que les précédents étaient récupérés plus tardivement, avec la saisie en entrée standard des utilisateurs du script.

La ligne 69 permet de configurer le fichier « /etc/default/isc-dhcp-server » de notre service DHCP, celui-ci contient les interfaces en écoute par le service. On indique d'écrire les nouvelles interfaces récupérées par saisie de l'utilisateur (ligne 143) du script, directement à la ligne « INTERFACESv4= » de notre fichier.

Os.system rentre dans notre terminal la commande permettant de redémarrer notre service DHCP afin que celui-ci prenne en compte nos configurations.

Partie « DNS »

```
72 #####
73 ##### Partie configuration du DNS #####
74 #####
75
76 def dns_conf(domain,ip):
77     os.system("apt-get install bind9")
78     os.system("cp /etc/bind/db.local /etc/bind/"+domain)
79
80     a = open("/etc/bind/named.conf.local","w")
81     a.write("\nzone \""+domain+"\" {\n")
82     a.write("        type master;\n")
83     a.write("        file \""+domain+"/\";\n")
84     a.write("        allow-query { any; };\n")
85     a.write("};\n")
86     a.close()
87
88     replaceAll("/etc/bind/"+domain,"localhost. root.localhost.", "ns.site."+domain+". root."+domain+".")
89     replaceAll("/etc/bind/"+domain,"localhost.", "ns")
90     replaceAll("/etc/bind/"+domain,"127.0.0.1",ip)
91     replaceAll("/etc/bind/"+domain,"@      IN      AAAA    ::1", "ns      IN      A      "+ip)
92     f=open("/etc/bind/"+domain,"a")
93     f.write("www      IN      A      "+ip)
94     f.write("\nmailx      IN      A      "+ip)
95     f.write("\n@      IN MX 100 mailx."+domain+".")
96     f.close()
97     os.system("service bind9 restart")
98
```

La description de cette partie est relativement simple et reprenant le principe de complétion par remplacement de certains paramètres à l'instar de la partie décrite précédemment dhcp. Ici, sur le même modèle, il s'agit des paramètres « domain » et « ip », qui se retrouvent dans la fameuse syntaxe, entre deux « + » et reviennent à plusieurs reprises.

Dans un premier temps, nous avons l'exécution du module os afin que le terminal lance l'installation du service DNS « bind9 » et la réalisation d'une copie intégrale du fichier de zone modèle « /etc/bind/db.local » en « /etc/bind/ nom_domaine » dans un soucis de préservation du fichier initial et de personnalisation de notre configuration.

Nous attribuons à la variable « a » l'ouverture du fichier « /etc/bind/named.conf.local » en écriture « w » (pour *write* en anglais). Ce fichier contient la configuration locale du serveur DNS, on y déclare les zones associées au domaine de la ligne 80 à la ligne 86 de notre script.

A partir de la ligne 88, il s'agit de la configuration de notre fichier de zone personnalisé avec les paramètres lui correspondant. A la ligne 92, nous notons qu'on attribue à la variable f , l'ouverture de ce fichier de zone en mode « a » qui permet d'écrire à la fin du fichier existant si celui-ci contient déjà du contenu et non de venir « écraser » comme le ferait le mode « w ».

Suite à la fermeture du fichier de zone, en ligne 97, il est demandé au module os d'exécuter dans le terminal, un redémarrage du service bind9.

Partie « Main »

```
99 #####
100 #####      main      #####
101 #####
102
103 def main(argv):
104
105     domain = ''
106     ip = ''
107     server_name = ''
108     subnet_mask = ''
109     option_dns = ''
110     sous_res = ''
111     interfaces = ''
112
113     try:
114         opts, args = getopt.getopt(argv, "hid:a:n:m:o:r:f:", ["domain=", "addr=", "name=", "mask=", "optdns=", "reseau=", "interfaces="])
115     except getopt.GetoptError:
116         print ('dhcp_dns.py -i pour le mode interactif ou dhcp_dns.py -d <domain> -a <addr ip> -n <server name> -m <subnet mask> -o <op'
117             sys.exit(2)
118     for opt, arg in opts:
119         if opt == '-h':
120             print ('dhcp_dns.py -i mode interactif ou dhcp_dns.py -d <domain> -a <addr ip> -n <server name> -m <subnet mask> -o <option'
121                 sys.exit()
122         elif opt in ("-d", "--domain"):
123             domain = arg
124         elif opt in ("-a", "--addr"):
125             ip = arg
126         elif opt in ("-n", "--name"):
127             server_name = arg
```

Cette partie traite les lignes de commande et la façon dont les arguments sont passés au programme Python. Chaque argument de la ligne de commande passé au programme est ajouté à `sys.argv`, qui est un objet de liste.

Ce qu'il faut retenir est que l'objet `sys.argv` contient le nom du script que nous appelons. Les arguments de la ligne de commande sont séparés par des espaces et chacun se présente comme un élément distinct dans la liste `sys.argv`. Les drapeaux de la ligne de commande, comme par exemple `--help`, se présentent également comme des éléments propres de la liste `sys.argv` et il est à noter que, certains de ces drapeaux de la ligne de commande peuvent eux-mêmes prendre des arguments. Exemple de « `-h` » qui prend un `print` en argument en ligne 120. Aussi bien le drapeau que l'argument sont présentés comme des éléments distincts dans la liste de `sys.argv`. Rien n'est fait pour les associer, on obtient alors rien de plus qu'une liste.

La partie intéressante du traitement est que la fonction `getopt` du module `getopt` prend trois paramètres ligne 114 du script : la liste des arguments qu'on obtient à partir de `sys.argv[1:]`, une chaîne contenant l'ensemble des drapeaux courts possibles et acceptés par le programme et une liste des drapeaux plus longs qui correspondent aux versions courtes.

Si un dysfonctionnement survient au moment d'analyser les drapeaux de ligne de commande, `getopt` déclenche une exception (ligne 115), que nous récupérons ensuite. Comme indiqué à `getopt` tous les drapeaux connus, il y a fort à parier que l'utilisateur final a passé des drapeaux de ligne de commande qui sont inconnus. Quand le script reçoit des drapeaux qu'il ne connaît pas, on met fin au programme de la manière la plus élégante qui soit, en fournissant un résumé des règles de bon usage (ligne 116).

```
128     elif opt in ("-m", "--mask"):
129         subnet_mask = arg
130     elif opt in ("-o", "--optdns"):
131         option_dns = arg
132     elif opt in ("-r", "--reseau"):
133         sous_res = int(arg)
134     elif opt in ("-f", "--interfaces"):
135         interfaces = arg
136     elif opt in ("-i", "--i"):
137         domain = input("Entrez le nom de domaine : ")
138         ip = input("Entrez l'ip du serveur dns : ")
139         server_name = input("Entrez le nom du serveur DHCP : ")
140         subnet_mask = input("Entrez le masque : ")
141         option_dns = input("Entrez les options dns : ")
142         sous_res=int(input("Entrez le nombre de sous réseaux : "))
143         interfaces=input("Entrez les interfaces d'écoute : ")
144     dns_conf(domain,ip)
145     dhcp_conf(server_name,subnet_mask,domain,option_dns,sous_res,interfaces)
146
147
148 if __name__ == "__main__":
149     main(sys.argv[1:])
```

Les lignes 144 et 145 permettent à la suite de l'exécution de cette partie « Main » ne venir exécuter leurs définitions avec les nouveaux éléments apportés par la manipulation des arguments en ligne de commande.

Les lignes 148 et 149 correspondent à la définition de main comme module principal de ce script. Il encourage la réutilisation du code en la rendant plus facile.

GitHub

Présentation de GitHub



GitHub est un site web et un service de cloud qui aide les développeurs à stocker et à gérer leur code, ainsi qu'à suivre et contrôler les modifications qui lui sont apportées.

Les deux principes qui tournent autour de GitHub sont le contrôle de version et le Git :


- Le contrôle de version aide les développeurs à suivre et à gérer les modifications apportées au code d'un projet logiciel. Au fur et à mesure qu'un projet logiciel prend de l'ampleur, le contrôle de version devient essentiel.
- Git est un système de contrôle de version distribué, ce qui signifie que l'ensemble de la base du code et de l'historique est disponible sur l'ordinateur de chaque développeur, ce qui permet des branchements et une fusion faciles.






Notre répertoire GitHub

Notre répertoire est joignable à l'adresse

https://github.com/Hvvvrd/Project_6_OpenClassrooms étant dans une démarche de participation à la vie de la communauté Open Source, notre répertoire est en accès public.

main ▾ 1 branch 0 tags Go to file Add file ▾ Code ▾

 Hvvvrd Update README.md ec1fef2 18 minutes ago 🕒 14 commits

 LICENSE	Initial commit	1 hour ago
 NOTICE	Create NOTICE	1 hour ago
 README.md	Update README.md	18 minutes ago
 res.ini	Create res.ini	1 hour ago
 script_config_DHCP_DNS.py	Update script_config_DHCP_DNS.py	22 minutes ago

Nous retrouvons dans notre répertoire :

- **script_config_DHCP_DNS.py** notre script lui même,
- accompagné de **res.ini** nécessaire à la configuration de notre DHCP,
- **README.md** est un fichier permettant l'introduction et la présentation de notre répertoire GitHub,
- **LICENSE** contient les dispositions légales de distribution de notre projet Open Source que nous avons décidé de fixer comme licence publique générale GNU,
- **NOTICE** écrit succinct sur les prérogatives exclusives de l'auteur sur le projet et de la nature de la licence,
- **Documentation_technique.pdf** cette présente documentation.