http://www.capitchilog.fr

Démarrer

le scripting

ou comment se lancer dans le développement...
...quand on n'aime pas ça

Date	Version	Auteur(s)	Commentaires
01/10/2019	1.0	Serge COUDÉ	Initialisation du document

Table des matières

Introduction

Ce document s'adresse à tous les futurs techniciens et administrateurs réseaux qui débutent la programmation de scripts et qui se sentent parfois désarmés devant la méthode à suivre pour réaliser un script dans les règles de l'art. Lorsqu'on a jamais fait de développement, ce n'est pas évident de savoir quoi utiliser, comment s'y prendre, dans quel ordre et faire correctement les choses. C'est normal, surtout si cela ne vous passionne pas au premier abord ! Mais dans tout métier ayant trait au Système & Réseau, il est obligatoire et impératif de savoir développer un minimum, pas forcément pour votre patron, mais surtout pour vous ! En créant vos scripts, vous allez vous supprimer beaucoup temps bêtement perdu à faire 10 ou 15 fois la même chose. Vous pouvez vous dire aussi que ce qui est bon pour vous l'est également pour votre entreprise dans laquelle vous évoluez si cela peut vous faire plaisir ! Donc, passage obligé, nous allons voir comment aborder la création de script (pour ne pas dire programmation si vous êtes allergique à ce terme!)

Les différents langages

Comme souvent, il n'y a pas qu'une seule façon de créer ses scripts : on peut utiliser une multitude de langages de programmation. Un point particulier de la création de scripts, c'est justement qu'il s'agit de script : on va donc « interpréter » les instructions, pas les compiler comme on le fait en C ou en Java.
Interpréter veut dire que le système (l'interpréteur) va exécuter au fur et à mesure chaque instruction présente dans le script, ligne après ligne. Si une instruction comporte une erreur, alors l'interpréteur s'arrêtera au moment où il atteindra cette dernière.
Dans le cas du langage Java (ou le C), il y a une compilation avant l'exécution. Si une erreur est présente dans une instruction, alors le compilateur s'arrêtera pour indiquer l'erreur et le programme ne sera pas créé, donc pas utilisable.
Bien évidemment, tout langage peut être utilisé pour faire une application, mais il sera plus ou moins aisé d'atteindre notre objectif selon notre choix de langage. On peut très bien réaliser un programme qui va dupliquer une arborescence d'un serveur vers un autre en Java. Il sera plus judicieux d'utiliser du bash, du python ou du perl pour réaliser cette action, mais on peut le faire aussi en Java, en C, etc.
En résumé, pour créer un script on va utiliser généralement un langage interprété comme le bash (ou le csh ou le ksh, d'autres langages shell utilisables dans le même « style »), le python, le perl, le tcl, etc. Un des plus classique et peut-être le plus ancien est le bash (ou un de ses « cousins »). D'autres langages ont eu leur effet de mode (comme perl il y a 15 ans, ou plus récemment python). D'autres ont été « détourné » de leur objectif premier comme le PHP ou le tcl. PHP me dites-vous ? Mais c'est pour le Web ! Oui, effectivement, mais il y a aussi la version « cliente » que l'on peut utiliser en ligne de commande dans un terminal, et on bénéficie alors de toutes les puissantes librairies qui l'accompagne !
Les langages auxquels je fais référence sont plutôt orientés système Unix/Linux mais présents également pour certains sous Windows. Vous ne trouverez pas un bash sous Windows à moins d'avoir installé un émulateur comme cygwin. Windows a lui aussi des langages de scripting comme le PowerShell ou le batch qui lui sont spécifiques.

Cette multitude de langages ne doit pas vous faire peur : ils sont peu ou proue utilisables de la même façon, seules certaines instructions possèdent des noms particuliers, mais majoritairement ce

sont souvent les mêmes. Ensuite, il se peut que la façon d'écrire les instructions diffèrent un peu comme le PHP vs Python mais cela ne va pas bien loin au niveau des différences.
Ce qu'il faut retenir, c'est que vous devez avoir des bases d'algorithmie, c'est à dire savoir décrire l'enchaînement des instructions à réaliser pour répondre à une problématique.
C'est cette partie qui pose souvent problème au débutant développeur : savoir organiser les instructions pour atteindre le résultat escompté, savoir quels « outils algorithmiques » utiliser, où et quand (exemples variable, tableau, boucle, condition, procédure, etc.). Après, le langage utilisé, c'est de la « rigolade », ils ont presque tous la même syntaxe (sauf python qui remplace les accolades par de l'indentation, mais bon, ce n'est pas grand chose !) et il vous faut connaître un peu l'anglais....
Donc lorsque vous savez organiser et décrire votre pensée (c'est à dire l'ensemble des choses réaliser) vous savez scripter !

Commençons par le commencement

Nous allons prendre un cas concret que j'ai quotidiennement à mon travail. Dans un des deux établissements dont j'ai la gestion du parc informatique, j'ai déployé 8 écrans d'affichages de « news ». Ainsi les élèves sont informés quotidiennement des actualités de l'établissement. J'ai conçu tout le système (architecture logicielle, programme) et mon établissement a réalisé l'acquisition des divers matériels (écran + Raspberry Pi). Chaque Raspberry Pi embarque donc un raspbian et un navigateur web qui affiche un site intranet spécial « news ».
De temps à autre le site est modifié (pas au niveau des news, mais dans son affichage global) et au lieu d'attendre la nuit suivante que le Raspberry Pi reboot (il redémarre chaque nuit), j'ai créé un script installé sur chaque Raspberry Pi qui redémarre juste le navigateur pour forcer la prise en compte du nouveau site.
Pour que l'ensemble du parc soit mis à jour, il faut alors que je me connecte 8 fois à distance sur un Raspberry Pi et que je lance à chaque fois mon script. Or comme j'oublie souvent les adresses IP de mes Rapberry Pi, je suis obligé d'afficher le fichier texte qui contient l'ensemble des IP des Raspberry Pi.
Je vous l'accorde, pour quelqu'un qui fait déjà du scripting, on va me dire « il y a moyen de fortement optimiser tout cela ! ». Oui, mais ça, c'est lorsqu'on connaît déjà : je propose une situation qu'un technicien ou un administrateur peut rencontrer à ses débuts et qui lui permet de voir et d'apprécier tout l'intérêt de se mettre un minimum au développement de scripts !

L'idée que je vous propose est donc de voir comment élaborer un script qui va vous permettre de « piloter » ce redémarrage des navigateurs.

Comment démarrer ?

Pour débuter la réalisation d'un script, je vais imaginer la façon dont je vais l'utiliser et ce qu'il va réaliser. Ce script va avoir besoin d'information : comment vais-je les lui fournir ?

- Qui va utiliser le script ?
- Où sera-t-il exécuté ?
- Que fait-il ?
- De quoi a-t-il besoin ?
- Comment l'utilise-t-on ?

C'est en répondant au fur et à mesure à ces questions que je vais concevoir le script.

Répondre aux questions !

Qui va utiliser le script ?

Moi pour le moment, mais peut-être à terme une autre personne. Il faut donc que je puisse savoir comment l'utiliser si je ne m'en rappelle plus ou bien qu'une autre personne le lance. Il faut donc que le script puisse afficher son modus operandi si je le lui demande.

Où sera-t-il exécuté ?

Ce script sera exécuté depuis mon PC professionnel sous Linux. Je vais plutôt utiliser un langage comme le shell (ou bien python si je veux être à la page !), mais comme je suis un « vieux réac », je vais choisir le bon vieux shell, plus exactement le Bourn Again Shell ou Bash.
Ce script va être composé en fait de 2 scripts. Le premier que je vais exécuter depuis mon poste et le second qui sera présent sur chaque raspberry pi. On pourrait faire en sorte que les instructions de ce dernier soient exécutées par le premier, mais pour l'exemple, je vais scinder l'application en 2 parties. Il est fréquent qu'une application soit découpée en plusieurs petites applications qui vont interagir ensemble.
Le premier script va piloter l'ensemble des raspberry pi, le second va relancer le navigateur web du raspberry pi, c'est pour cela qu'il est plus facile qu'il soit exécuté depuis le raspberry pi.

Que fait-il ?

Le script va exécuter à distance le script de redémarrage du navigateur sur le ou les Raspberry Pi. En effet, il me permettra d'indiquer sur quel(s) « Pi » je souhaite relancer le navigateur. Pour connaître quels sont les « Pi » disponibles, il m'affichera la liste des écrans si besoin.

De quoi a-t-il besoin ?

Le script doit connaître la liste des « Pi » (adresse IP, identifiant, mot de passe pour se connecter à distance en ssh). Il doit connaître le nom du script à exécuter à distance. Il doit savoir analyser les « paramètres » que je peux lui passer éventuellement (par exemple *1,3,8* ou *tous*). Il doit savoir exécuter une commande à distance sur une autre machine sous Linux ou « assimilé ».

Comment l'utilise-t-on ?

Si je lance le script sans paramètre, c'est à dire sans information placée à la suite de son nom, il doit m'afficher toutes les informations dont il a besoin pour bien fonctionner. Imaginons que mon script s'appelle pilotepi.sh. Si j'appelle le script sans paramètre :

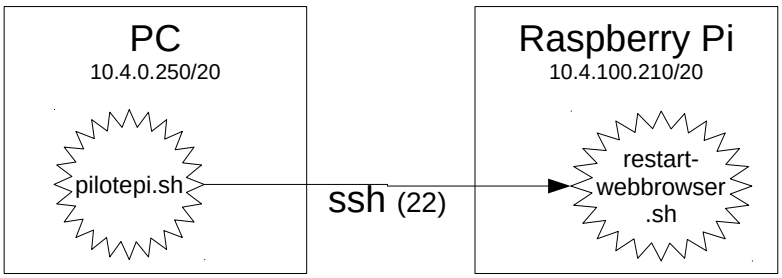
```
$ pilotepi.sh
        Usage : pilotepi.sh <1,...,8|tous>
                1 : Accueil 10.3.100.200
                2 : Bureau 10.3.100.201
                …
```

Avec toutes ces réponses que nous avons apportés, on est en mesure de commencer la rédaction du script !

On avance étape par étape

Schéma de déploiement

Pour bien démarrer le codage, il faut d'abord bien appréhender l'architecture de notre programme. On a vu précédemment que l'application serait composées de 2 sous-applications. Faisons un schéma pour représenter l'architecture de déploiement :



Il est important de créer un schéma de déploiement : cela permet à toute personne de bien se représenter où est située l'application et comment tout communique ensemble.

Cela permet aussi de voir d'éventuels problématiques de communication, comme par exemple le passage de proxy ou de pare-feux, etc. Dans mon cas, c'est très simple, même réseau, pas de pare-feux ou de proxy. Mais dans de grandes structures, ce n'est souvent pas le cas : on va devoir passer par des serveurs pour atteindre les machines souhaitées, ces serveurs sont appelés des serveurs de « rebond ». Il faut alors dans le schéma de déploiement indiquer tous les « intermédiaires » qui sont présents, sans en oublier...

Dans le schéma indiqué précédemment, on peut comprendre que le script pilotepi.sh se trouve sur un PC qui a pour adresse IP 10.4.0.250 et il se connecte en ssh (port 22) sur le raspberry pi qui a pour adresse 10.4.100.210 pour exécuter le script restart-webbrowser.sh. Un schéma est quand même plus simple qu'une longue description, non ?

Maintenant, passons à la rédaction de notre script de pilotage !

Passage de paramètres au script

Première chose que doit faire notre script c'est de savoir s'il est appelé avec ou sans paramètre passé (0, 1, n).

Cas A : sans paramètre

Il doit alors afficher comment il s'utilise et la liste des Raspberry Pi. Cela veut donc dire que la liste des « Pi » est disponible dans le script. On pourrait se contenter d'afficher la liste avec la commande *echo* (en shell, elle permet d'afficher du texte dans la console). Mais ce n'est pas une bonne solution : notre script va avoir besoin plus tard de ou des adresses IP des « Pi » au moment de l'exécution à distance. Il faut donc que les adresses soient stockées dans une « liste » (on va utiliser la structure d'un tableau) qui sera utilisée pour :

- l'affichage des « Pi » disponibles
- l'exécution à distance du script de redémarrage

De plus, il sera facile d'ajouter ou supprimer des « Pi » sans avoir à modifier à plusieurs endroits les informations.
On va pousser l'optimisation en externalisant la déclaration de ce tableau dans un autre fichier car il se peut qu'on ait à réaliser plus tard un autre script qui aura besoin lui aussi de la liste des « Pi ».
Au lieu de reécrire deux fois le tableau, il suffira au 2ème script de, lui aussi, charger le tableau ; et la mise à jour de la liste ne sera à faire qu'une seule fois !

Penser toujours à ne pas avoir de données dupliquées, une donnée doit être définie à un seul endroit.

Cas B : avec paramètre

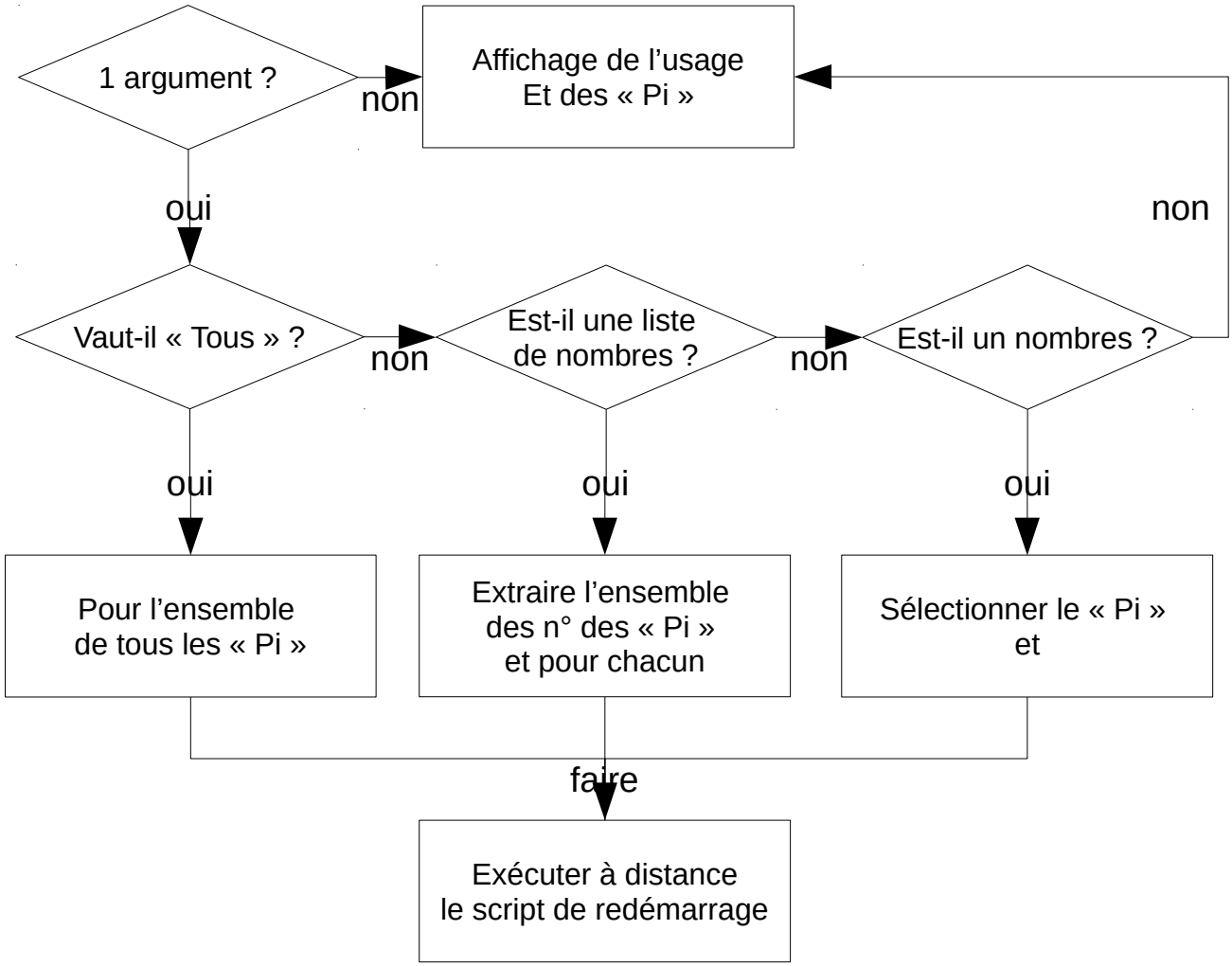
Le script doit alors n'avoir qu'un seul paramètre. S'il y en a deux ou plus, ce n'est pas bon, il faut alors qu'il affiche la façon de l'utiliser (comme dans le cas A) : on ne va pas réécrire deux fois l'usage du script !
On va donc « prendre » toutes les instructions qui affichent l'usage du script et les placer dans une fonction qui sera appelée dans les 2 cas : s'il n'y a aucun paramètre ou s'il y en a plus d'un. Elle pourra être également appelée dans le 3ème cas lorsqu'un seul paramètre est passé mais qui ne correspond pas à un nombre, une suite de nombres séparés par une virgule ou encore le mot « tous ». Optimisons !
Récapitulons :

On affiche l'usage si :
- Cas 1 : script appelé sans paramètre
- Cas 2 : script appelé avec au moins deux paramètres
- Cas 3 : script appelé avec un paramètre, mais ce paramètre n'est pas
  - un nombre
  - une suite de nombres séparés par une virgule
  - le mot « tous »

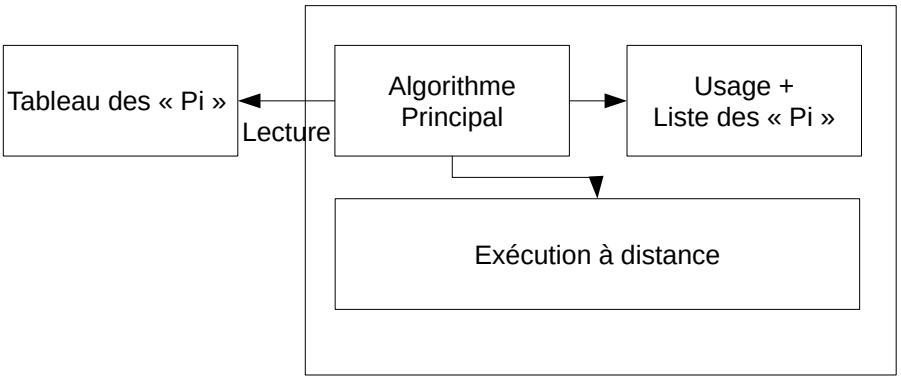
En regardant ce que nous venons d'indiquer, nous pouvons observer un début d'algorithme !
Si le paramètre vaut « tous », alors il faut exécuter à distance le redémarrage du navigateur sur l'ensemble des « Pi ». Si le paramètre est composé d'une série de nombres entiers séparés par une virgule, alors il faut exécuter à distance le redémarrage du navigateur sur les « Pi » correspondant aux numéros. Si le paramètre n'est qu'un nombre entier, alors il faut exécuter à distance le redémarrage du navigateur sur le « Pi » correspondant.

On remarque donc qu'il s'agit toujours de la même action à réaliser sur un ou plusieurs « Pi ». On va donc regrouper les instructions permettant le redémarrage à distance du navigateur d'un « Pi » dans une fonction qui sera « générique » (pas spécifique à un « Pi »). Cette fonction a besoin d'information pour réaliser cette action : le n° du « Pi ». A partir de ce n°, elle devra être en mesure de récupérer l'adresse IP, l'identifiant à utiliser ainsi que le mot de passe pour se connecter à distance. Cette fonction (dans mon cas) sera appelée au plus 8 fois ou au pire 1 fois.

En ayant réfléchi à ce qu'on voulait que le programme fasse, on peut modéliser son algorithme comme ceci :



Cette façon de présenter le fonctionnement du script est quand même plus simple à comprendre que de longues phrases, non ?
On peut aussi présenter le script de façon plus « fonctionnelle » comme ceci :



Une fois qu'on a l'ensemble de ces informations et représentations, il n'y a plus qu'à développer !
Nous allons avoir besoin « d'outils » (instructions) pour rédiger le code : énumérons les !

De quoi avons nous besoin pour écrire notre script ?

- Déclarer une variable
- Stocker dans un tableau la liste des « Pi » avec leur adresse IP
- Savoir « charger » (lire et exécuter) des instructions présentes dans un autre script (le fichier contenant la déclaration du tableau)
- Savoir connaître le nombre de paramètres passés lors de l'appel du scripte
- Savoir tester une condition et exécuter un groupe d'instructions si elle est vrai, un autre groupe sinon.
- Savoir afficher des informations dans la console
- Savoir définir un commentaire (instructions non exécutées)
- Savoir parcourir tous les éléments d'un tableau
- Savoir tester si le contenu d'une variable est le même que celui d'une chaîne de caractères
- Savoir tester si le contenu d'une variable est égal, inférieur ou supérieur à une valeur numérique
- Savoir tester si une chaîne est composée de nombres séparés par une virgule
- Savoir extraire des nombres séparés par des virgules et les « parcourir »
- Savoir définir une fonction (groupe d'instructions) avec un paramètre ou sans qui renvoi une valeur numérique comme résultat
- Savoir exécuter une commande système
- Connaître la commande qui permet d'exécuter à distance une autre commande

Avec toutes ces données, on est en mesure d'écrire notre script.

Voici comment en bash sont définies les instructions :

- Déclarer une variable nommée mavariable

```
mavariable=valeur
```

- Stocker dans un tableau (à deux dimensions) la liste des « Pi » avec leur nom et adresse IP, tableau stocké dans une variable nommée rb

```
declare -A rb
rb[1,0]="Accueil"
rb[1,1]="192.168.1.220"
rb[2,0]="Hall Batiment A"
rb[2,1]="192.168.1.221"
rb[3,0]="Hall Batiment B"
rb[3,1]="192.168.2.220"
rb[4,0]="Salle des profs Batiment C"
rb[4,1]="192.168.8.220"
rb[5,0]="Salle des profs Batiment B"
rb[5,1]="192.168.3.220"
rb[6,0]="Self"
rb[6,1]="192.168.7.220"
rb[7,0]="Laboratoires"
rb[7,1]="192.168.7.221"
rb[8,0]="Batiment Pedagogique"
rb[8,1]="192.168.5.220"
```

- Savoir « charger » (lire et exécuter) des instructions présentes dans un autre script (le fichier contenant la déclaration du tableau)

```
source autre_script.sh
```

- Savoir connaître le nombre de paramètres passés lors de l'appel du script

```
$#
```

- Savoir tester une condition et exécuter un groupe d'instructions si elle est vrai, un autre groupe sinon.

```
if [ cond ]
```

```
then
	instruction 1
	instruction 2
	instruction 3
else
	instruction 4
	instruction 5
	instruction 6
fi
```

- Savoir afficher des informations dans la console

```
echo "info à afficher"
```

- Savoir définir un commentaire (instructions non exécutées)

```
# Commentaire qui ne sera pas exécuté
```

- Savoir parcourir tous les éléments d'un tableau

```
for ((i=1; i<=nb; i++)) do
	# instruction exécutée sur un élément d'un tableau
	echo "$i : ${montableau[$i,0]} (${montableau[$i,1]})"
done
```

- Savoir tester si le contenu d'une variable nommée mavariable est le même que celui d'une chaîne de caractères

```
if [ "$mavariable" == "chaîne" ]
then
	…
fi
```

- Savoir tester si le contenu d'une variable nommée mavariable est égal, inférieur ou supérieur à une valeur numérique

```
if [ $mavariable -eq 1 -o $mavariable -lt 1 -o $mavariable -gt 1 ]
then
	…
fi
```

- Savoir tester si une chaîne est composée de nombres séparés par une virgule (ici utilisation d'une expression régulière)

```
re='^[0-9,]+$'
if [[ $piList =~ $re ]]
then
	…
fi
```

- Savoir extraire des nombres séparés par des virgules stockés dans une variable nommée mavariable et les « parcourir »

```
for i in `echo $mavariable | tr "," " "`
do
	echo $i # Utiliser la valeur de chaque nombre
done
```

- Savoir définir une fonction (groupe d'instructions) avec un paramètre ou sans qui renvoi une valeur numérique comme résultat

```
mafonction () {
	if [ "$1" == "tous" ]
	then
		return 1
	else
		return 0
	fi
}
```

- Savoir exécuter une commande système

```
exec "commande système"
```

- Connaître la commande qui permet d'exécuter à distance une autre commande

```
ssh -c
```

Il nous reste plus qu'à assembler les pièces du puzzle en fonction de notre algorithme qu'on a définit préalablement...

Nous avons défini que la liste des Raspberry Pi serait dans un fichier « externe » pour pouvoir être réutilisée par d'autres scripts ultérieurement.
Créons un premier script **raspi.inc.sh** qui contiendra la définition de l'ensemble des « Pi » dans un tableau en bash (on le nomme .inc.sh pour indiquer qu'il s'agit d'un script qui doit être chargé par un autre script...) ainsi que certaines informations comme l'identifiant et le mot de passe :

```
#!/bin/bash
# Déclaration du tableau des Raspberry Pi; le tableau est placé dans une variable nommée
rb
declare -A rb
# Déclaration du nombre de Raspberry Pi dans une variable nommée nbRP
nbRP=8
# Déclaration de l'identifiant de connexion au raspberry pi
loginRP=pi
# Déclaration du mot de passe de connexion au raspberry pi
passwdRP=lemotdepasse

# Initialisation du tableau à 2 dimensions
rb[1,0]="Accueil"
rb[1,1]="192.168.1.220"
rb[2,0]="Hall Batiment A"
rb[2,1]="192.168.1.221"
rb[3,0]="Hall Batiment B"
rb[3,1]="192.168.2.220"
rb[4,0]="Salle des profs Batiment C"
rb[4,1]="192.168.8.220"
rb[5,0]="Salle des profs Batiment B"
rb[5,1]="192.168.3.220"
rb[6,0]="Self"
rb[6,1]="192.168.7.220"
rb[7,0]="Laboratoires"
rb[7,1]="192.168.7.221"
rb[8,0]="Batiment Pedagogique"
rb[8,1]="192.168.5.220"
```

Nous avons donc un tableau à 2 dimensions :

- première dimension : index du Raspberry Pi (son rang, de 1 à 8 dans notre cas)
- deuxième dimension : les informations du Raspberry Pi, premier index son libellé (rang 0) et deuxième index l'adresse IP

On peut voir cette représentation sous forme « graphique » comme ceci :

1		2		3		4		5		6		7		8	
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Accueil	192.168.1.220	Hall Batiment A	192.168.1.221	…	…	…	…	…	…	…	…	…	…	…	…

Nous avons ensuite défini un algorithme pour que notre programme réalise nos actions souhaitées.
Créons alors le script principal **pilotepi.sh** qui contiendra les instructions de l'algorithme :

```
#!/bin/bash

#####################################################################
# Script pilotepi.sh                                                #
# Auteur : Serge COUDÉ                                              #
# Version 1.0                                                       #
# Date : septembre 2017                                             #
#                                                                   #
```

```
# Ce script est diffusé sous la licence Cecill v2                    #
# url : http://                                                      #
#                                                                    #
# Version 1.0 : initialisation du script                             #
#                                                                    #
######################################################################

# Chargement des informations sur les raspberry pi
source raspi.inc.sh

# Définition d'une fonction affichant le "mode d'emploi" du script
function usage {
    # Affichage d'une ligne avec le nom du script "dynamique" ($0)
    echo "Usage : $0 [n|n1,n2,n3|tous]"
    echo
    echo "Liste des raspberry pi disponibles :"
    # Boucle pour afficher l'ensemble des raspberry pi définis dans le tableau chargé
préalablement
    # La boucle utilise une variable i qui va être initialisée avec la valeur 1 et
ensuite à chaque
    # itération incrementée de 1 jusqu'à atteindre la valeur contenue dans la variable
nbRP
    for ((i=1; i<=nbRP; i++)) do
        # Affichage d'une ligne avec le "rang" puis le nom et ensuite l'adresse IP du
raspberry pi
         echo "     $i :" ${rb[$i,0]} "(${rb[$i,1]})"
    done # Fin de la boucle
    exit 0 # On termine le script avec un code retour valant 0 car le script s'est bien
déroulé
}

# Définition de la fonction qui va redémarrer à distance le navigateur du raspberry pi
# Cette fonction va être appelée par la fonction piRefresh
# Cette fonction a besoin du "rang" du raspberry pi en argument
# Elle renverra une fois terminée une valeur 0 (bien déroulée) ou 1 (problème survenu)
function navRefresh() {
    rang=$1 # On met la valeur du premier argument de la fonction dans une variable avec
un nom plus explicite
    # On vérifie que le rang est bien un rang existant
    if [ -z "${rb[$rang,0]}" ] # Si la "cellule" dans le tableau est vide
    then
        echo "    [ERREUR] Le raspberry pi de rang $rang n'existe pas..."
        return 1 # On renvoi 1 car une erreur s'est produite
    fi
    echo "    Redémarrage du navigateur sur le raspberry pi $rang"
    # Affichage des informations du raspberry pi
    echo "        Nom : ${rb[$rang,0]}"
    echo "        IP : ${rb[$rang,1]}"
    # Excution du script restart-webbrowser.sh à distance sur le raspberry pi avec
      l'utilisateur pi et son mot de passe "compteraspberrypi"
    sshpass -p compteraspberrypi ssh -X ${rb[$rang,0]} -l pi './restart-webbrowser.sh'
    return 0
}

# Définition de la fonction qui va appeler la fonction navRefresh sur un, ou plusieurs
raspberry pi
# Cette fonction va être appelée par les instructions "principales" du script
# Cette fonction a besoin du "rang" de chaque raspberry pi séparé par une virgule
# Elle renverra une fois terminée une valeur 0 (bien déroulée) ou 1 (problème survenu)
function piRefresh() {
    idxList=$1 # On met la valeur du premier argument de la fonction dans une variable
avec un nom plus explicite, ici la liste des rangs (indexes) de raspberry pi
    result=0 # Définition d'une variable avec une valeur initiale à 0 qui permettra de
savoir
             # si l'ensemble des raspberry pi souhaités ont bien eu leur navigateur
redémarré
```

```
    # On "découpe" le paramètre en fonction du séparateur virgule et on "boucle" sur
l'ensemble des items à l'aide d'une variable i
    for i in `echo $idxList | tr "," " "` # "découpage à l'aide de l'instruction tr (en
fait, on remplace chaque virgule par un espace, et l'instruction for applique les
instructions contenues entre le do et done à chaque élément)
    do
        # Appel de la fonction qui va réaliser la demande de redémarrage du navigateur à
distance
        navRefresh $i
        res=$? # récupération de la valeur renvoyée par la fonction navRefresh appelée
(0 ou 1); cette valeur est placée dans une variable nommée res
        if [ $res -eq 1 ] # Si la fonction appelée a rencontré un problème
        then
            result=1 # On "enregistre" le fait que la fonction a rencontré un problème
        fi
    done # Fin de la boucle
    return $result # La fonction renvoi soit 0 (tout s'est bien passé) ou 1 (au moins un
problème est survenu)
}


# On teste si le script est appelé avec 0, 1 ou plusieurs paramètres
# Dans le cas d'aucun paramètre ou plus de 1, alors on affiche l'usage
if [ $# -eq 0 -o $# -gt 1 ]
then
    usage
else # Il y a un paramètre donné lors de l'appel du script
    piList="" # Définition d'une variable piList qui va contenir les rangs de tous les
raspberry pi dont on veut redémarrer le navigateur
    if [ "$1" == "tous" ] # Le paramètre vaut "tous" alors on défini la liste des rangs
de tous les raspberry pi
    then
        echo "Début du redémarrage de l'ensemble des raspberry pi"
        # La boucle utilise une variable i qui va être initialisée avec la valeur 1 et
ensuite à chaque
        # itération incrementée de 1 jusqu'à atteindre la valeur contenue dans la
variable nbRP
        # On va créer une liste de tous les rangs des raspberry pi séparés par une
virgule
        for ((i=1; i<=nbRP; i++)) do
            if [ $i -eq 1 ] # S'il s'agit du premier, on ne met pas de virgule
            then
                piList="$i"
            else # Sinon on place une virgule avant d'insérer le rang
                piList="$piList,$i"
            fi
        done # Fin de la boucle
    else # On doit avoir un groupe d'au moins 1 raspberry pi sous forme de liste de
rangs séparés par une virgule
        piList=$1 # On met le paramètre passé au script dans une variable nommée de
façon plus explicite
        # On vérifie que le paramètre est bien une liste de nombres séparés par des
virgules
        re='^[0-9,]+$' # Utilisation d'une expression régulière pour effectuer le test
        if ! [[ $piList =~ $re ]] # Si le paramètre ne correspond pas à l'expression
régulière alors
        then
            # On affiche un message d'erreur
            echo "    [ERREUR] : le paramètre n'est pas une liste de n° de pi..."
            # Le script se termine avec un code retour signalant une erreur
            exit 1
        fi
        echo "Début du redémarrage d'un groupe de raspberry pi"
    fi
    # On appelle la fonction piRefresh avec la liste des rangs
```

```
    piRefresh $piList
    result=$? # On récupère la valeur du retour de la fonction piRefresh appelée
    if [ $result -eq 1 ] # Au moins 1 raspberry pi n'a pas eu son navigateur
redémarré...
    then
         echo "    [ERREUR] Au moins 1 raspberry pi n'a pas eu son navigateur
redémarré..."
         exit 1 # Un problème est survenu, on termine le script avec la valeur 1
    else
        # Tout s'est bien passé, on adapte le message de fin en fonction du paramètre
passé
        if [ "$1" == "tous" ]
        then
            echo "Fin du redémarrage de l'ensemble des raspberry pi"
        else
            echo "Fin du redémarrage d'un groupe de raspberry pi"
        fi
        exit 0 # Tout s'est bien passé, le script se termine avec la valeur 0
    fi
fi
```

Ce script est volontairement TRES documenté pour expliquer l'ensemble des instructions, il est à visé pédagogique. Dans l'optique d'un script « classique » en entreprise, il n'y a pas besoin de commenter autant que cela, mais c'est néanmoins nécessaire voire obligatoire.
Exemple du même script en version « entreprise » :

```
#!/bin/bash

#####################################################################
# Script pilotepi.sh                                                #
# Auteur : Serge COUDÉ                                              #
# Version 1.0                                                       #
# Date : septembre 2017                                             #
#                                                                   #
# Ce script est diffusé sous la licence Cecill v2                   #
# url : http://                                                     #
#                                                                   #
# Version 1.0 : initialisation du script                            #
#                                                                   #
#####################################################################

# Chargement des informations sur les raspberry pi
source raspi.inc.sh

# Définition d'une fonction affichant le "mode d'emploi" du script
function usage {
    echo "Usage : $0 [n|n1,n2,n3|tous]"
    echo
    echo "Liste des raspberry pi disponibles :"
    for ((i=1; i<=nbRP; i++)) do
         echo "     $i :" ${rb[$i,0]} "(${rb[$i,1]})"
    done
    exit 0 # On termine le script avec un code retour valant 0 car le script s'est bien
déroulé
}

# Définition de la fonction qui va redémarrer à distance le navigateur du raspberry pi
# Cette fonction va être appelée par la fonction piRefresh
# Cette fonction a besoin du "rang" du raspberry pi en argument
# Elle renverra une fois terminée une valeur 0 (bien déroulée) ou 1 (problème survenu)
function navRefresh() {
    rang=$1
    # On vérifie que le rang est bien un rang existant
    if [ -z "${rb[$rang,0]}" ]
    then
        echo "    [ERREUR] Le raspberry pi de rang $rang n'existe pas..."
```

```
        return 1
    fi
    echo "    Redémarrage du navigateur sur le raspberry pi $rang"
    # Affichage des informations du raspberry pi
    echo "        Nom : ${rb[$rang,0]}"
    echo "        IP : ${rb[$rang,1]}"
    # Excution du script restart-webbrowser.sh à distance sur le raspberry pi avec
      l'utilisateur pi et son mot de passe "compteraspberrypi"
    sshpass -p compteraspberrypi ssh -X ${rb[$rang,0]} -l pi './restart-webbrowser.sh'
    return 0
}

# Définition de la fonction qui va appeler la fonction navRefresh sur un, ou plusieurs
raspberry pi
# Cette fonction va être appelée par les instructions "principales" du script
# Cette fonction a besoin du "rang" de chaque raspberry pi séparé par une virgule
# Elle renverra une fois terminée une valeur 0 (bien déroulée) ou 1 (problème survenu)
function piRefresh() {
    idxList=$1
    result=0

    # On "découpe" le paramètre en fonction du séparateur virgule et on "boucle" sur
l'ensemble des items à l'aide d'une variable i
    for i in `echo $idxList | tr "," " "`
    do
        navRefresh $i
        res=$?
        if [ $res -eq 1 ] # Si la fonction appelée a rencontré un problème
        then
            result=1
        fi
    done
    return $result # La fonction renvoi soit 0 (tout s'est bien passé) ou 1 (au moins un
problème est survenu)
}


# On teste si le script est appelé avec 0, 1 ou plusieurs paramètres
# Dans le cas d'aucun paramètre ou plus de 1, alors on affiche l'usage
if [ $# -eq 0 -o $# -gt 1 ]
then
    usage
else # Il y a un paramètre donné lors de l'appel du script
    piList=""
    if [ "$1" == "tous" ]
    then
        echo "Début du redémarrage de l'ensemble des raspberry pi"
        for ((i=1; i<=nbRP; i++)) do
            if [ $i -eq 1 ] # S'il s'agit du premier, on ne met pas de virgule
            then
                piList="$i"
            else # Sinon on place une virgule avant d'insérer le rang
                piList="$piList,$i"
            fi
        done
    else # On doit avoir un groupe d'au moins 1 raspberry pi sous forme de liste de
rangs séparés par une virgule
        piList=$1
        # On vérifie que le paramètre est bien une liste de nombres séparés par des
virgules
        re='^[0-9,]+$' # Utilisation d'une expression régulière pour effectuer le test
        if ! [[ $piList =~ $re ]] # Si le paramètre ne correspond pas à l'expression
régulière
        then
            echo "    [ERREUR] : le paramètre n'est pas une liste de n° de pi..."
            # Le script se termine avec un code retour signalant une erreur
```

```
            exit 1
        fi
        echo "Début du redémarrage d'un groupe de raspberry pi"
    fi
    # On appelle la fonction piRefresh avec la liste des rangs
    piRefresh $piList
    result=$?
    if [ $result -eq 1 ] # Au moins 1 raspberry pi n'a pas eu son navigateur
redémarré...
    then
         echo "    [ERREUR] Au moins 1 raspberry pi n'a pas eu son navigateur
redémarré..."
         exit 1 # Un problème est survenu, on termine le script avec la valeur 1
    else
        # Tout s'est bien passé, on adapte le message de fin en fonction du paramètre
passé
        if [ "$1" == "tous" ]
        then
            echo "Fin du redémarrage de l'ensemble des raspberry pi"
        else
            echo "Fin du redémarrage d'un groupe de raspberry pi"
        fi
        exit 0 # Tout s'est bien passé, le script se termine avec la valeur 0
    fi
fi
```

Pour bien comprendre le découpage du script, on peut le reprendre et mettre en parallèle notre algorithme, cette fois-ci sans commentaire pour une meilleure lisibilité dans ce cas précis :

```
#!/bin/bash

source raspi.inc.sh
```

Affichage de l'usage
Et des « Pi »

```
function usage {
    echo "Usage : $0 [n|n1,n2,n3|tous]"
    echo
    echo "Liste des raspberry pi disponibles :"
    for ((i=1; i<=nbRP; i++)) do
         echo "     $i :" ${rb[$i,0]} "(${rb[$i,1]})"
    done
    exit 0
}
```

Exécuter à distance
le script de redémarrage

```
function navRefresh() {
    rang=$1
    if [ -z "${rb[$rang,0]}" ]
    then
        echo "    [ERREUR] Le raspberry pi de rang $rang n'existe pas..."
        return 1
    fi
    echo "    Redémarrage du navigateur sur le raspberry pi $rang"
    echo "        Nom : ${rb[$rang,0]}"
    echo "        IP : ${rb[$rang,1]}"
    sshpass -p compteraspberrypi ssh -X ${rb[$rang,0]} -l pi './restart-webbrowser.sh'
    return 0
}
```

```
function piRefresh() {
    idxList=$1
    result=0
```

Extraire l'ensemble des n° des « Pi » et pour chacun

```
    for i in `echo $idxList | tr "," " "`
    do
```

Exécuter à distance le script de redémarrage

```
        navRefresh $i
        res=$?
        if [ $res -eq 1 ]
        then
            result=1
        fi
    done
    return $result
}
```

1 argument ?

```
if [ $# -eq 0 -o $# -gt 1 ]
then
    usage
```

Affichage de l'usage Et des « Pi »

```
else
    piList=""
```

Vaut-il « Tous » ?

```
    if [ "$1" == "tous" ]
    then
        echo "Début du redémarrage de l'ensemble des raspberry pi"
```

Pour l'ensemble de tous les « Pi »

```
       for ((i=1; i<=nbRP; i++)) do
            if [ $i -eq 1 ]
            then
                piList="$i"
            else
                piList="$piList,$i"
            fi
        done
    else
        piList=$1
```

Est-il une liste de nombres ?

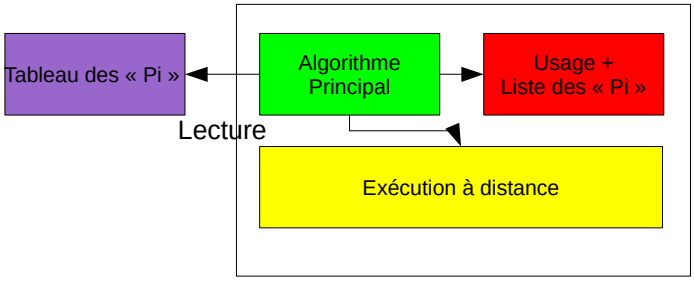
Est-il un nombres ?

```
       re='^[0-9,]+$'
        if ! [[ $piList =~ $re ]]
        then
            echo "    [ERREUR] : le paramètre n'est pas une liste de n° de pi..."
            exit 1
        fi
        echo "Début du redémarrage d'un groupe de raspberry pi"
    fi
    piRefresh $piList
```

```
    result=$?
    if [ $result -eq 1 ]
    then
         echo "    [ERREUR] Au moins 1 raspberry pi n'a pas eu son navigateur
redémarré..."
         exit 1
    else
        if [ "$1" == "tous" ]
        then
            echo "Fin du redémarrage de l'ensemble des raspberry pi"
        else
            echo "Fin du redémarrage d'un groupe de raspberry pi"
        fi
        exit 0
    fi
fi
```

On constate que l'on retrouve globalement les différentes actions décrites dans l'algorithme (et heureusement !) mais aussi qu'il y a présence de plein d'instructions servant à gérer les erreurs. Ces dernières sont très importantes pour avoir un script « robuste », c'est à dire que ne se « plante » pas en cas de la survenue d'une mauvaise saisie ou d'une instruction qui ne s'exécute pas bien. Si le script « plante », alors il se peut qu'il se retrouve dans un état « instable » ce qui peut ouvrir des failles de sécurité pour la machine l'exécutant...

On peut aussi faire « apparaître » le découpage fonctionnel que l'on avait envisagé :




```
#!/bin/bash

source raspi.inc.sh

function usage {
    echo "Usage : $0 [n|n1,n2,n3|tous]"
    echo
    echo "Liste des raspberry pi disponibles :"
    for ((i=1; i<=nbRP; i++)) do
         echo "     $i :" ${rb[$i,0]} "(${rb[$i,1]})"
    done
    exit 0
}

function navRefresh() {
    rang=$1
    if [ -z "${rb[$rang,0]}" ]
    then
        echo "    [ERREUR] Le raspberry pi de rang $rang n'existe pas..."
        return 1
    fi
    echo "    Redémarrage du navigateur sur le raspberry pi $rang"
    echo "        Nom : ${rb[$rang,0]}"
    echo "        IP : ${rb[$rang,1]}"
    sshpass -p compteraspberrypi ssh -X ${rb[$rang,0]} -l pi './restart-webbrowser.sh'
```

```
    return 0
}

function piRefresh() {
    idxList=$1
    result=0
    for i in `echo $idxList | tr "," " "`
    do
        navRefresh $i
        res=$?
        if [ $res -eq 1 ]
        then
            result=1
        fi
    done
    return $result
}

if [ $# -eq 0 -o $# -gt 1 ]
then
    usage
else
    piList=""

    if [ "$1" == "tous" ]
    then
        echo "Début du redémarrage de l'ensemble des raspberry pi"

       for ((i=1; i<=nbRP; i++)) do
            if [ $i -eq 1 ]
            then
                piList="$i"
            else
                piList="$piList,$i"
            fi
        done
    else
        piList=$1
        re='^[0-9,]+$'
        if ! [[ $piList =~ $re ]]
        then
            echo "    [ERREUR] : le paramètre n'est pas une liste de n° de pi..."
            exit 1
        fi
        echo "Début du redémarrage d'un groupe de raspberry pi"
    fi
    piRefresh $piList
    result=$?
    if [ $result -eq 1 ]
    then
         echo "    [ERREUR] Au moins 1 raspberry pi n'a pas eu son navigateur
redémarré..."
         exit 1
    else
        if [ "$1" == "tous" ]
        then
            echo "Fin du redémarrage de l'ensemble des raspberry pi"
        else
            echo "Fin du redémarrage d'un groupe de raspberry pi"
        fi
        exit 0
    fi
fi
```

Notre second script présent sur le raspberry pi sera écrit de la façon suivante :

```
#!/bin/sh

#########################################################################
# Script restart-webbrowser.sh                                          #
# Auteur : Serge COUDÉ                                                  #
# Version 1.0                                                           #
# Date : septembre 2017                                                 #
#                                                                       #
# Ce script est diffusé sous la licence Cecill v2                       #
# url : http://                                                         #
#                                                                       #
# Version 1.0 : initialisation du script                                #
#                                                                       #
#########################################################################

# Accès à l'écran principal d'affichage
export DISPLAY=":0"
# Récupération de l'id de la fenêtre du navigateur
WID=$(xdotool search --onlyvisible --class iceweasel|head -1)
# Mise en avant plan de la fenêtre du navigateur
xdotool windowactivate ${WID}
# Envoyer la simulation de commande de touches ctrl+F5
xdotool key ctrl+F5
exit 0
```

Améliorations possibles

Il est toujours possible d'améliorer ses scripts. Rien que le fait de l'utiliser nous fait voir qu'il y a des manques ou des failles qu'on n'a pas pris en compte à la base (malheureusement!).

On peut par exemple constater qu'on est obligé d'avoir un script installé sur chaque raspberry pi pour rafraîchir la page du navigateur. Ca va pour 8 écrans, mais imaginez pour 20 ou 30. A chaque modification du script « local », il faudrait la répercuter sur 20 ou 30... Il faudrait voir à ce que le script pilotepi.sh puisse exécuter lui même à distance les instructions du script « local ». Et si ce n'est pas possible, une autre solution serait d'écrire un script « local » générique qui irait récupérer les instructions sur mon ordinateur pour les exécuter sur le raspberry pi. Ainsi, je n'aurais qu'une modification à réaliser et l'ensemble des raspberry pi en profiteraient...

Et si on utilisait un autre langage pour réaliser la même chose ?

Nous avons créé un script en bash, nous pouvons réaliser les mêmes actions dans un autre langage. Par exemple, un langage en vogue depuis quelques années est le langage python. Nous allons voir que la méthode est la même, seules les instructions vont être un peu différentes.

Reprenons les différents besoins cités précédemment :

Voici comment en python sont définies les instructions :

- Déclarer une variable nommée mavariable

```
mavariable = valeur
```

- Stocker dans un tableau (à deux dimensions) la liste des « Pi » avec leur nom et adresse IP, tableau stocké dans une variable nommée montableau

```
import pandas as pd

rb = pd.DataFrame(
```

```
    columns=["nom","ip"],
    index=[1,2,3,4,5,6,7,8],
    data=[["Accueil", "192.168.1.220"],
          ["Hall Batiment A", "192.168.1.221"],
          ["Hall Batiment B", "192.168.2.220"],
          ["Salle des profs Batiment C", "192.168.8.220"],
          ["Salle des profs Batiment B", "192.168.3.220"],
          ["Self", "192.168.7.220"],
          ["Laboratoires", "192.168.7.221"],
          ["Batiment Pedagogique", "192.168.5.220"]])
```

- Savoir « charger » (lire et exécuter) des instructions présentes dans un autre script (le fichier contenant la déclaration du tableau)

```
import <autrescript> as nomfacileautiliser """ autrescript
```

- Savoir connaître le nombre de paramètres passés lors de l'appel du script

```
import sys
nbargs = len(sys.argv)-1
```

- Savoir tester une condition et exécuter un groupe d'instructions si elle est vrai, un autre groupe sinon.

```
if nbargs == 1:
      print "Il y a 1 argument"
      print "Le programme peut s'executer"
else:
      print "Il y a 0 ou plus de 1 arguments"
      print "Le programme ne pas pas s'executer"
```

- Savoir afficher des informations dans la console

```
print mavariable
```

- Savoir définir un commentaire (instructions non exécutées)

```
""" ceci est un commentaire qui ne sera pas execute"""
```

- Savoir parcourir tous les éléments d'un tableau

dans notre cas, pas besoin : on affiche directement le contenu du tableau qui se présente bien à l'affichage

- Savoir tester si le contenu d'une variable nommée mavariable est le même que celui d'une chaîne de caractères

```
if mavariable == "tous":
      print "oui"
else
      print "non"
```

- Savoir tester si le contenu d'une variable nommée mavariable est égal, inférieur ou supérieur à une valeur numérique

```
if mavariable == 0 or mavariable > 1 or mavariable < 3
```

- Savoir tester si une chaîne est composée de nombres séparés par une virgule

```
import re
if re.search("^[0-9,]+$", "1,7,8"):
      print "c'est ok"
else
      print "ce n'est pas bon !"
```

- Savoir extraire des nombres séparés par des virgules stockés dans une variable nommée mavariable et les « parcourir »

```
mavariable="1,8,9"
for i in mavariable.split(","):
      print i
```

- Savoir définir une fonction (groupe d'instructions) avec un paramètre ou sans qui renvoie une valeur numérique comme résultat

```
def mafonction(liste):
    """ Definition de la fonction qui fait quelque chose

        Parameters
        ----------
        liste : string
        parametre passe a la fonction et qu'on peut utiliser dedans
```

```
        Returns
        ----- --
        int
        Renvoie 0 (si bien deroulee) ou 1 (si un probleme est survenu)
    """
    if liste == "tous" :
       return 0
    return 1
```

- Savoir exécuter une commande système

```
import os
os.system('commande système')
```

- Connaître la commande qui permet d'exécuter à distance une autre commande (la librairie paramiko doit être préalablement installée !)

```
import paramiko

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(hostname='<adresseip>', username='<utilisateur>', password='<motdepasse>')
stdin, stdout, stderr = ssh.exec_command('commande à exécuter')
for line in stdout.read().splitlines():
    print(line)
ssh.close()
```

On applique la même méthode que pour le bash en assemblant les différentes parties pour reproduire l'algorithme :

Définition du fichier contenant les informations des Raspberry Pi ***raspiconf.py :***

```
"""
Module config des raspberry pi
"""
__authors__ = ("Sege COUDE")
__version__ = "1.0"
__copyright__ = " copyleft "
__date__ = "04/07/2019"

import pandas as pd

rb = pd.DataFrame(
    columns=["nom","ip"],
    index=[1,2,3,4,5,6,7,8],
    data=[["Accueil", "192.168.1.220"],
          ["Hall Batiment A", "192.168.1.221"],
          ["Hall Batiment B", "192.168.2.220"],
          ["Salle des profs Batiment C", "192.168.8.220"],
          ["Salle des profs Batiment B", "192.168.3.220"],
          ["Self", "192.168.7.220"],
          ["Laboratoires", "192.168.7.221"],
          ["Batiment Pedagogique", "192.168.5.220"]])
```

Définition du script principal ***pilotepi.py*** :

```
#!/usr/bin/python

"""
Script pilotepi.sh
Ce script est diffuse sous la licence Cecill v2
url : http://
Version 1.0 : initialisation du script
"""
```

```
__authors__ = ("Sege COUDE")
__version__ = "1.0"
__copyright__ = " copyleft "
__date__ = "04/07/2019"


# Chargement des informations sur les raspberry pi (chargement du script raspiconf.py)
import raspiconf as conf

# Chargement des librairies utiles au script
import sys
import numpy as np
import re
import paramiko

def usage():
    """ fonction affichant le "mode d'emploi" du script.
    """
    # Affichage d'une ligne avec le nom du script "dynamique" (argv[0])
    print "Usage : " + sys.argv[0] + " [n|n1,n2,n3|tous]"
    print
    print "Liste des raspberry pi disponibles :"
    # L'affichage standard d'un objet de type dataFrame correspond a ce que nous
attendons
    print conf.rb
    sys.exit(0) # On termine le script avec un code retour valant 0 car le script s'est
bien deroule

def navRefresh(rang):
    """ Definition de la fonction qui va redemarrer a distance le navigateur du
raspberry pi

    Cette fonction va etre appelee par la fonction pirefresh

    Parameters
    ----------
    rang : int
    Rang du raspberry pi.
    Returns
    ----- --
    int
    Renvoie 0 (si bien deroulee) ou 1 (si un probleme est survenu)
    """
    # On verifie que le rang est bien un rang existant
    try:
        if not conf.rb.loc[rang].empty:
            print "    Redemarrage du navigateur sur le raspberry pi " + str(rang)
            # Affichage des informations du raspberry pi
            print "        Nom : " + conf.rb.loc[rang,"nom"]
            print "        IP : " + conf.rb.loc[rang,"ip"]
            """ Execution du script restart-webbrowser.sh a distance sur le raspberry pi
avec l'utilisateur pi et son mot de passe "compteraspberrypi"
            """
            ssh = paramiko.SSHClient()
            ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            ssh.connect(hostname=conf.rb.loc[rang,"ip"], username='pi',
password='compterasberrypi')
            stdin, stdout, stderr = ssh.exec_command('./restart-webbrowser.sh')
            for line in stdout.read().splitlines():
                print "            res : " + line
            ssh.close()
            return 0
    except:
        # Si la "cellule" dans le tableau est vide
        print "    [ERREUR] Le raspberry pi de rang " + str(rang) + " n'existe pas..."
```

```
        return 1

def piRefresh(idxList):
    """ Definition de la fonction qui va appeler la fonction navrefresh sur un, ou
plusieurs raspberry pi.

        Cette fonction va etre appelee par les instructions "principales" du script

        Parameters
        ----------
        idxList : string
        Liste des "rangs" des raspberry pi separes par des virgules.
        Returns
        ----- --
        int
        Renvoie 0 (si bien deroulee) ou 1 (si un probleme est survenu)
    """
    result = 0 # Definition d'une variable avec une valeur initiale a 0 qui permettra de
savoir
               # si l'ensemble des raspberry pi souhaites ont bien eu leur navigateur
redemarre
    # On "decoupe" le parametre en fonction du separateur virgule et on "boucle" sur
l'ensemble des items a l'aide d'une variable i
    for i in idxList.split(","):
        # Appel de la fonction qui va realiser la demande de redemarrage du navigateur a
distance
        res = navRefresh(int(i))
        # recuperation de la valeur renvoyee par la fonction navrefresh appelee (0 ou
1); cette valeur est placee dans une variable nommee res
        if res == 1: # Si la fonction appelee a rencontre un probleme
            result=1 # On "enregistre" le fait que la fonction a rencontre un probleme
    return result # La fonction renvoi soit 0 (tout s'est bien passe) ou 1 (au moins un
probleme est survenu)

# On teste si le script est appele avec 0, 1 ou plusieurs parametres
# Dans le cas d'aucun parametre ou plus de 1, alors on affiche l'usage
nbargs = len(sys.argv)-1
if nbargs == 0 or nbargs > 1:
    usage()
else: # Il y a un parametre donne lors de l'appel du script
    piList = "" # Definition d'une variable piList qui va contenir les rangs de tous les
raspberry pi dont on veut redemarrer le navigateur
    if sys.argv[1] == "tous": # Le parametre vaut "tous" alors on defini la liste des
rangs de tous les raspberry pi
        print "Debut du redemarrage de l'ensemble des raspberry pi"
        # On va creer une liste de tous les rangs des raspberry pi separes par une
virgule
        # La boucle utilise une variable i qui va prendre les valeurs de 1 a la
dimension du tableau des raspberry pi
        for i in np.arange(1,conf.rb.shape[0]):
            if i == 1: # S'il s'agit du premier, on ne met pas de virgule
                piList = str(i)
            else: # Sinon on place une virgule avant d'inserer le rang
                piList = piList + "," + str(i)
    else: # On doit avoir un groupe d'au moins 1 raspberry pi sous forme de liste de
rangs separes par une virgule
        piList = sys.argv[1] # On met le parametre passe au script dans une variable
nommee de facon plus explicite
        # On verifie que le parametre est bien une liste de nombres separes par des
virgules
        er="^[0-9,]+$" # Utilisation d'une expression reguliere pour effectuer le test
        if not re.search(er, piList):
            # le parametre n'est pas une succession de nombres separes par des virgules,
on affiche un message d'erreur
            print "    [ERREUR] : le parametre n'est pas une liste de numeros de pi
separes par des virgules..."
```

```
            # Le script se termine avec un code retour signalant une erreur
            sys.exit(1)
        # le parametre est bien une succession de nombres separes par des virgules, on
continu !
        print "Debut du redemarrage d'un groupe de raspberry pi"
    # On appelle la fonction pirefresh avec la liste des rangs
    result = piRefresh(piList)
    # On recupere la valeur du retour de la fonction pirefresh appelee
    if result == 1: # Au moins 1 raspberry pi n'a pas eu son navigateur redemarre...
         print "    [ERREUR] Au moins 1 raspberry pi n'a pas eu son navigateur
redemarre..."
         sys.exit(1) # Un probleme est survenu, on termine le script avec la valeur 1
    else:
        # Tout s'est bien passe, on adapte le message de fin en fonction du parametre
passe
        if sys.argv[1] == "tous":
            print "Fin du redemarrage de l'ensemble des raspberry pi"
        else:
            print "Fin du redemarrage d'un groupe de raspberry pi"
        sys.exit(0) # Tout s'est bien passe, le script se termine avec la valeur 0
```

A vous de retrouver les différentes parties de l'algorithme créé au début, ainsi que le découpage en modules !
Vous constaterez que si les instructions peuvent être différentes, globalement la structure du programme reste assez semblable à la première !

Améliorations possibles

On constate que la définition des Raspberry Pi est effectuée dans un script externe et écrite dans le langage utilisé. Il peut être intéressant de créer plutôt un fichier texte au format csv par exemple (avec des séparateurs de champ, etc.) et de développer une fonction qui lira le fichier texte et chargera les bonnes valeurs en mémoire. Ainsi, ce fichier peut être réutilisé par d'autres programmes écrits dans d'autres langages sans avoir à recoder sa définition...

Les bonnes pratiques à adopter dans le développement de scripts

Fort de ces deux exemples, on peut dégager des façons de programmer qui sont considérées comme des bonnes pratiques.

Un script doit pouvoir afficher son « manuel » d'utilisation facilement. Soit lorsqu'on l'appelle sans paramètre, soit avec un paramètre comme -? ou -h.

Un script devrait pouvoir être exécuté en mode « silencieux », c'est à dire sans interaction de l'utilisateur durant son exécution. Il prendra alors une liste d'arguments au moment de son appel et ensuite les utilisera au fur et à mesure de ses besoins.

Il est obligatoire de commenter/documenter son script. Même si techniquement ce n'est pas nécessaire, c'est un besoin pour qui reprendra éventuellement le code afin de ne pas passer du temps à comprendre comment il a été réalisé et pensé.

Un script doit toujours prendre en compte toutes les saisies données (que cela de la part de l'utilisateur, d'un fichier de configuration, des paramètres passés au moment de son lancement, de données reçues en temps réel, etc.) Tous les « input » devraient être au préalable vérifiés avant leur prise en compte par le script. Chaque instruction du script pouvant être mal exécutée devrait être vérifiée après ou pendant leur exécution pour être prise en charge par la gestion des erreur le cas

échéant. En bash par exemple, on vérifiera chaque fois le code retour renvoyé par l'appel d'une fonction ou d'un autre programme et effectuer le traitement qui permettra au script au mieux de continuer à s'exécuter, au pire de s'arrêter proprement. En python, on utilisera le mécanisme des « try except » qui permet d'encapsuler la gestion des erreurs de façon efficace.

Un script doit pouvoir fournir à tout moment un « état » de son exécution : les fameux logs. Au minimum il doit afficher dans la console les informations d'exécution, mais il est souhaitable qu'il stocke toutes les informations dans un fichier qui sera défini soit par défaut soit passé en argument ou déclaré dans un fichier de configuration. Ce fichier contiendra le nom de la machine sur laquelle le script s'exécute, la date et l'heure de l'enregistrement pour chaque information, le niveau de la trace (trace, debug, info, warning, error, critical) et l'information enregistrée, un enregistrement par ligne. Il sera alors aisé par un système de supervision (comme Nagios par exemple) de lire à fréquence régulière ce fichier et d'y détecter des termes permettant d'alerter les superviseurs...

Aller plus loin

Si on va plus loin, on peut, pour une meilleure maintenabilité du code (reprise du code par autrui ou évolution dans le temps), utiliser des « patrons de conceptions » (design pattern) standards que tout développeur/scripteur doit/devrait connaître. On peut aussi améliorer la lisibilité du code et son efficacité en utilisant lorsque c'est possible des structures informatiques comme les classes (en Programmation Orientée Objet ou POO). Cette POO permet de travailler de manière très efficace en écrivant du code optimisé et non redondant.