# ARCHITECTURE LINUX

Romuald GRIGNON
Robin PENEA

Coding Factory by ITESCIA
CCI Paris Ile-de-France

Année scolaire 2019-2020

# Présentation du projet

- Le but de ce projet est de combiner différents éléments ou fonctionnalités utilisables dans un système d'exploitation Linux à savoir :
  - Parcourir et gérer l'arborescence de fichiers, et gérer les informations du système de fichiers
  - Lancer des exécutables et gérer les informations des processus du système d'exploitation
  - Utiliser des opérations de redirection de flux et les opérations de filtrage de données
  - Utiliser de la programmation C et du scripting shell pour interagir avec les commandes et les flux
- Le thème utilisé ici est la domotique
- Nous allons, entre autres, créer un exécutable C qui va simuler le comportement d'un capteur d'une maison (température, ouverture de porte, présence dans une pièce, état d'une lampe, ...) qui envoie régulièrement des données.
- Parfois les données envoyées seront des données d'erreur pour simuler le fait qu'un capteur a détecté un défaut

# Présentation du projet

- Dans ce projet, nous allons créer 2 scripts shell qui vont permettre de séquencer, mais surtout d'automatiser une succession d'actions.
- Le premier script « generation.sh » va s'occuper de lancer nos exécutables C et de rediriger leurs sorties vers des fichiers de logs
- Le deuxième script « supervision.sh » va s'occuper de vérifier le comportement du premier script, et notamment s'assurer que les fichiers de logs remplis par « generation,sh » ne sont pas trop gros, et si c'est le cas, les compresser pour les archiver. Ce deuxième script s'assurera également de stopper et de relancer la generation
- D'un point de vue fonctionnel, le premier script est ici pour récupérer les informations qui proviennent des capteurs de la maison, et le deuxième script est là pour superviser le travail du premier
- L'ensemble des stories qui va suivre permettra d'aborder les différents éléments cités précédemment

# Présentation du projet

- Les données de sortie normale d'un capteur :
  - sensorID : identifiant du capteur, chaine de caractères arbitraire
  - sensorName : nom du capteur
  - value : valeur entiere
  - minValue : valeur entiere plus petite que value
  - meanValue : valeur à virgule comprise entre minValue et maxValue
  - maxValue : valeur entiere plus grande que toutes les autres valeurs
- Les données de sortie d'erreur d'un capteur :
  - sensorID : identifiant du capteur, chaine de caractères arbitraire
  - errorCode : une chaine de caractères arbitraire pour résumer le type de l'erreur
  - errorDetail : un message indiquant le détail de l'erreur

# LES STORIES

# Story #1 : Script Shell

- En tant qu'étudiant de la Coding Factory,
- Je veux créer un script shell capable de lancer un exécutable
- Afin de de m'initier au concept de script système
- Critères d'acceptation :
  - Le script shell se nomme « generation.sh »
  - Le script appelle l'exécutable « genTick » qui est fourni par le PO
  - Le script prend en argument un entier et le passe à l'exécutable
  - Sur le terminal apparaît un message OK ou ERROR à intervalle régulier, l'intervalle étant le nombre entier de milli-secondes fourni en argument
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #2 : Executable C

- En tant qu'étudiant de la Coding Factory,
- Je veux créer un programme C
- Afin de m'initier au développement système
- Critères d'acceptation :
  - Codage d'un programme C « genSensorData » qui compile et s'exécute sans erreurs, crash, …
  - Le programme C va simplement afficher un message de bienvenue au démarrage puis se stopper
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #3 : Flux (stdin, stdout)

- En tant qu'étudiant de la Coding Factory,
- Je veux modifier mon script et mon programme C
- Afin de rediriger les flux d'entrée/sortie de l'un à l'autre
- Critères d'acceptation :
  - Modifiez « genSensorData » pour qu'il lise l'entrée Standard et la recopie telle qu'elle sur la sortie standard
  - Modifiez « generation.sh » pour qu'il chaîne « genTick » et « genSensorData »
  - Votre script affichera toujours un message ERROR ou OK à intervalle régulier comme la story #1 mais ici la sortie sera fournie par « genSensorData »
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #4 : Simulateur de capteur domotique

- En tant qu'étudiant de la Coding Factory,
- Je veux modifier mon programme C
- Afin de générer des données comme le ferait un capteur domotique
- Critères d'acceptation :
  - Modifiez « genSensorData » pour qu'il génère une ligne de données nominales et une ligne de données d'erreur, telles que décrites en début de document
  - Il générera une ligne nominale quand il reçoit sur l'entrée standard « OK »
  - Il générera une ligne d'erreur quand il reçoit sur l'entrée standard « ERROR »
  - Votre script « generation.sh » affichera maintenant une ligne nominale ou d'erreur à intervalle régulier : nous avons donc notre simulateur de capteur fonctionnel
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #4bis : Simulateur de capteur domotique

- En tant qu'étudiant de la Coding Factory,
- Je veux modifier mon programme C
- Afin de générer des données comme le ferait un capteur domotique
- Critères d'acceptation :
  - Modifiez « genSensorData » pour qu'il gère sur son entrée standard des caractères intempestifs qiu seraient reçus.
    - Ex : si il reçoit ERR#R il ne doit pas générer quoi que ce soit en sortie car cette chaine n'est pas correcte.
  - Vous utiliserez l'exécutable fourni genTickv2 dans votre application pour cela (cette version 2 envoie des caractères intempestifs de manière aléatoire
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #5 : Fichiers de logs

- En tant qu'étudiant de la Coding Factory,
- Je veux modifier mon script de generation
- Afin de rediriger les flux de données vers de fichiers
- Critères d'acceptation :
  - Modifiez « generation.sh » pour qu'il prenne en argument un dossier et 2 noms de fichiers
  - Modifiez le pour qu'il envoie le flux de stdout vers le premier fichier (fichier de log nominal) et le flux de stderr vers le 2eme fichier (fichier de logs d'erreurs)
  - Si les fichiers existent déjà, on ajoute les logs à la suite
  - Maintenant votre script n'affiche plus rien mais quand il est en cours d'exécution, les deux fichiers passés en arguments sont remplis en continu
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #6 : Filtrage des données

- En tant qu'étudiant de la Coding Factory,
- Je veux modifier mon script de generation
- Afin de filtrer les données à stocker dans les fichiers
- Critères d'acceptation :
  - Modifiez « generation.sh »pour qu'il filtre les données nominales (stdout) avant qu'elles arrivent dans le fichier de log nominal. Il s'agira de ne garder QUE les données suivantes :
    - SensorID
    - SensorName
    - Value
    - MaxValue
  - Le fichier de log nominal sera maintenant rempli avec les mêmes données qu'avant mais filtrées
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #7 : Utilisateur Unix

- En tant qu'étudiant de la Coding Factory,
- Je veux modifier mon script de génération
- Afin de stocker les fichiers de logs dans le dossier de l'utilisateur courant
- Critères d'acceptation :
  - Le script « generation.sh » va récupérer l'utilisateur courant et afficher son identifiant
  - Le script va vérifier que l'utilisateur est un utilisateur bien précis (en dur dans le script). Si ce n'est pas le cas, un message d'erreur s'affichera et le script s'arrêtera
  - Le dossier passé en paramètre au script est maintenant un sous-dossier du dossier racine de cet utilisateur
  - Votre script se comporte donc comme à la story précédente mais cette fois-ci les fichiers de logs sont situés dans un sous-dossier de la racine de l'utilisateur plutot que le répertoire courant
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #8 : Supervision (taille des fichiers)

- En tant qu'étudiant de la Coding Factory,
- Je veux créer un script de supervision
- Afin de contrôler la taille des fichiers de logs
- Critères d'acceptation :
  - Crééz un autre script « supervision.sh » qui va prendre la meme liste de paramètres que le script « generation.sh » + une taille de fichier en octets
  - Ce script va vérifier qui est l'utilisateur courant et aller récupérer la taille des fichiers de logs
  - Afficher un message si le dossier de logs de l'utilisateur n'existe pas ou si le script « generation.sh » n'est pas lancé
  - Ce script va ensuite tester si la taille de ces fichiers ne dépasse pas la taille maximale passée en paramètre et afficher un message contenant les tailles des fichiers + la mention si ils dépassent ou non la limite
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #9 : Supervision (archivage)

- En tant qu'étudiant de la Coding Factory,
- Je veux modifier mon script de supervision
- Afin de compresser les fichiers et les archiver
- Critères d'acceptation :
  - Modifiez le script « supervision.sh » afin qu'il stoppe l'exécution du script « generation.sh » lancé par le même utilisateur si la taille des fichiers de logs dépasse la limite
  - Une fois le script « generation.sh » stoppé, votre script va compresser les fichiers de logs dans le même repertoire. Utilisez le datetime courant dans le nom de cette archive
  - La compression des fichiers ne se fera QUE si le script « generation.sh » était lancé.
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #10 : Supervision (tri et comptage)

- En tant qu'étudiant de la Coding Factory,
- Je veux modifier mon script de supervision
- Afin de trier et compter le nombre d'enregistrements dans mes fichiers
- Critères d'acceptation :
  - Modifiez « supervision.sh » pour qu'il ajoute dans l'archive un fichier qui récapitule des informations sur les logs. Ce fichier contiendra :
    - Une ligne qui indique le nombre d'enregistrements dans le fichier infos.log
    - Une ligne qui indique le nombre d'enregistrements dans le fichier errors.log
  - Modifiez « supervision.sh » pour qu'il modifie infos.log et errors.log avant de les sauvegarder et qu'il trie par numéro de Sensor les enregistrements de chacun des fichiers
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #11 : Signaux Unix

- En tant qu'étudiant de la Coding Factory,
- Je veux modifier mon code (script ou programme C)
- Afin de capturer le signal d'interruption et afficher un message de sortie
- Critères d'acceptation :
  - Modifiez votre script « generation.sh » ou votre exécutable C « genSensorData » (au choix) afin qu'il capture le signal SIGINT
  - Faîtes en sorte d'afficher un dernier message sur la sortie standard à la réception de ce signal
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #12 : Supervision (processus) (1/2)

- En tant qu'étudiant de la Coding Factory,
- Je veux modifier mon script de supervision
- Afin de pouvoir terminer et relancer le script de generation
- Critères d'acceptation :
  - Modifiez votre script « supervision.sh » pour que ce soit lui qui lance le script « generation.sh » la toute première fois
  - Après avoir créé l'archive des logs, vous pourrez relancer le processus « generation.sh » afin qu'il regénère des fichiers de logs
  - A ce stade, vous aurez 2 scripts qui tournent en boucle, « generation.sh » et « supervision.sh », le premier générant des fichiers de logs, le deuxieme stoppant le premier quand les fichiers sont trop gros, créant une archive et le relançant ensuite
  - Cet enchainement d'action doit donc se poursuivre en boucle
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)

# Story #13 : Supervision (processus) (2/2)

- En tant qu'étudiant de la Coding Factory,
- Je veux _modifier mon script de supervision
- Afin de _mettre en pause et reprendre l'exécution du script de generation
- Critères d'acceptation :
  - Vous allez créer une variante des stories #11 et #12 en modifiant « supervision.sh » afin qu'il envoie le signal SIGSTOP au lieu de SIGINT. Ainsi le processus de génération ne sera pas détruit.
  - Après archivage des fichiers de log, votre script enverra le signal SIGCONT afin que le processus de génération reprenne là où il s'était arrêté
  - Tous les éléments de code ainsi que les commentaires sont en anglais
  - Documentation du projet (guide utilisateur) incluant un diagramme fonctionnel de l'application
  - Archivage sous Git du livrable de la story (code + diagramme)