ZANTE Guillaume

CHATIRON Thibault

SRT2

Printemps 2013

Rapport de projet LO14

Architecture et administration des systèmes

Contenu

Introduction : 2

1) Contenu du projet 2

2) Sauvegarde dans un fichier var/log/tacheron 3

3) Commande tacherontab 5

4) La commande tacheron 8

a) Autorisation des utilisateurs 8

b) Vérification de la syntaxe 9

c) Gestion des secondes 14

d) Comparaison des dates 14

e) Fonction principale 16

Conclusion 19

Introduction :

Dans le cadre de l'UV LO14, qui introduit les bases de l'architecture et de l'administration des systèmes, nous avons été amenés à réaliser un projet de programmation informatique. L'UV est centré sur l'utilisation de Linux et de l'apprentissage du bash. Pour réaliser notre projet nous avons donc utilisé le système d'exploitation linux et ses capacités.
Le but de ce projet est de réaliser une commande sous la forme d'une commande Linux permettant de planifier l'exécution de tâches par le système. Cette nouvelle commande est appelée « tacheron ». La commande doit être capable d'exécuter toutes les taches allant d'une commande simple à l'exécution de scripts complexes. L'exécution de ces scripts et commandes doivent être mémorisé grâce à un message de compte rendu contenu dans un fichier texte.

1) Contenu du projet

Afin de nous répartir le travail et pour pouvoir nous fixer des objectifs nous avons cherché dans un premier temps à repérer les grandes parties du travail que nous devrons fournir. Nous avons donc divisé le projet en trois parties distinctes :

- La commande tâcheron à proprement parler, qui permet de lire le fichier /etc/tacherontab contenant les informations sur les fichiers à exécuter ainsi que le moment où ils devront l'être. Les fichiers à exécuter devrons être placés dans le répertoire /etc/tacheron/. Le fichier tacherontab doit répondre à certaines contraintes d'écriture afin de pouvoir être exécuté.
- La sauvegarde des comptes rendus de l'exécution de la commande dans le fichier : /var/log/tacheron. Chaque action ajoute des lignes supplémentaires à la fin du fichier pour savoir si l'exécution s'est effectuée correctement.
- La création d'une commande tacherontab, dont la syntaxe est la suivante *tacherontab [-u user] {-l | -r | -e}.* Cette commande permet la programmation du service tacheron en fonction de l'utilisateur que l'on a défini:
    - *–l* permet d'afficher le fichier tacherontab de l'utilisateur désigné.
    - *–r* permet d'effacer le fichier tacherontab de l'utilisateur désigné.
    - *–e* permet d'éditer ou de créer un fichier tahcrontab[user] de l'utilisateur

2) Sauvegarde dans un fichier var/log/tacheron

Dans un premier temps, nous avons eu l'idée de créer une fonction plus générale qui permettra de sauvegarder le résultat de l'exécution d'une tache dans un fichier à l'emplacement var/log/tacheron. Nous avons donc créé une fonction qui prend en entrée plusieurs paramètres, dans un premier temps le nom de la tache exécuté ou un message relatif à son exécution ainsi que comme paramètre une indication sur le bon fonctionnement de l'exécution. La fonction peut s'écrire : update [commande] {--succeed--, --error--}. Avant, on vérifie si les paramètres d'entrée sont conformes.

Grace à une boucle itérative on compare chaque paramètre d'entrée à la recherche de ces indications :

```
for i in $info
do
     tab[$i]=$i
# verification si l'execution s'est déroulé normalement
```

On vérifie si le paramètre de contrôle est –error--, si c'est le cas, alors on sort de la boucle et l'on enregistre un message d'erreur qui sera écrit dans le fichier var/log/tacheron.

```
if [ "${tab[$num]}" = '--error--' ] ;  then
     message_error="ERREUR: Le fichier n'a pu etre execute"
     unset -v tab[$i]
     flag=1
     break
```

On vérifie si le paramètre de contrôle est –succeed--, si c'est le cas, alors on sort de la boucle et l'on enregistre un message d'erreur qui sera écrit dans le fichier var/log/tacheron. On mémorise dans tab seulement le nom de la fonction, on supprime le caractère de contrôle grâce à la commande unset.

```
elif [ "${tab[$num]}" = '--succeed--' ] ; then
     message_error="Tache    accomplie!    Aucune    erreur    a
l'execution"
     unset -v tab[$i]
     flag=1
     break
fi
done
```

On vérifie si les paramètres de contrôle ne sont pas présent, si c'est le cas, alors on met fin à la fonction et l'on affiche un message d'erreur sur la sortie standard de la machine, le fichier var/log/tacheron ne sera donc pas modifié.

```
If [ "$flag" = "1"  ] ; then
     echo "parametres d'entrée invalides"
     flag=0
     exit
fi
```

Si les paramètres d'entrée sont corrects et que le programme c'est correctement exécuté alors on mémorise le résultat de la commande dans le fichier. On mémorise dans un premier temps la date actuelle, qui sera à peu de chose près la même date que la date d'exécution de la commande sauf si la commande met beaucoup de temps à s'exécuter. Cela dépend donc des capacités de la machine. On mémorise ensuite le nom du fichier, ou de la commande qui a été exécutée. Dans une troisième ligne, on mémorise le message d'erreur qui indique si la commande c'est exécuté correctement.

```
# Ecriture dans le fichier /var/log/tacheron.txt, historique
d'utilisation des taches planifiées.

echo "date d'execution:" `date` >> var/log/tacheron.txt
echo "nom du fichier: ${tab[*]}"  >> var/log/tacheron.txt
echo -e "$message_error\n" >> var/log/tacheron.txt

fi
```

Cependant, dans notre programme tacheron, nous avons décidé de ne pas utiliser cette fonction car nous avons jugé que l'utilisation de la fonction risquait de ralentir encore un peu plus le processus et de rendre moins performant notre commande. De plus comme la date que l'on affiche dans le fichier var/log/tacheron.txt est celle de l'exécution de la fonction update et non celle de la fonction tacheron, nous avons décidé de simplifier les choses en ajoutant simplement trois echo ci-dessus en modifiant la date : en prenant celle d'exécution de la commande, le nom du fichier ou de la commande est le dernier champ du fichier tacherontab[user]. Pour le message d'erreur, on fait une simple vérification de la sortie standard pour savoir s'il n'y a pas eu d'erreur lors de l'exécution du programme.

3) Commande tacherontab

Dans un premier temps, on cherche l'utilisateur pour cela, on vérifie si le mot –u est le premier mot de la chaine de caractère

```
premier_mot= `cut -d ' ' -f1 $1`
deuxieme_mot= `cut -d ' ' -f2 $1`

if [ "premier_mot" = "-u" ] ; then
                user=$deuxieme_mot
fi
```

On mémorise dans deux variables les deux premiers champs d'entrée, si le premier champ d'entrée est « -u » alors le second champ sera l'utilisateur. Cependant, pour être sûr que l'on prend bien le bon paramètre pour l'utilisateur, on a décidé de procéder autrement. On stock les paramètres d'entrée dans une variable param. Puis on isole le premier et le dernier terme pour vérifier que le premier est bien –u et que le dernier est soit –e, -l ou –r.

```
dernier_mot=${param##* }
```

Permet d'isoler le dernier mot de la chaine de caractère d'entrée. On prend tous les caractères(*) en partant de la fin (##) jusqu'au premier espace. L'avantage ici par rapport à l'utilisation d'un cut c'est que l'on a pas besoin de connaitre la longueur de la chaine de caractère pour prendre le dernier mot.

```
verif_validity=${param%% *}
```

Permet d'isoler le premier mot de la chaine de caractère d'entrée. On prend tous les caractères(*) en commençant par le début (%%) jusqu'au premier espace.

Ensuite on vérifie si on a entré un utilisateur en paramètre

```
if [ "$verif_validity"  = "-u" ]
                then
                     user=${param:3}
#Permet de supprimer les 3 premiers caractères de la chaine de caractère param

                     user=${user%%-*}
#Conserve tous les caractères de la chaine param sans le « -u », jusqu'au symbole « - »

Fi
```

Ensuite on vérifie le dernier mot de la chaine qui est stocké dans la variable dernier_mot afin de savoir quelle sera la commande à exécuter sur le fichier tacherontab[user].

Pour afficher le fichier tacherontab de l'utilisateur dans la sortie, avec le paramètre « -l », c'est-à-dire sur le terminal :

```
          if [ "$dernier_mot" = "-l" ]
                    then
                    echo "affichage du fichier tacherontab de
l'utilisateur" $user

# Changement de répertoire pour trouver le repertoire de
l'utilisteur concerné
                    cd /home
                    cd $user
                    cd Bureau/

# Affichage du fichier tacherontab
                    read `find etc/tacheron/tacherontab$user`
```

Pour vider le fichier tacherontab de l'utilisateur, avec le paramètre « -r » :

```
               elif [ "$dernier_mot" = "-r" ]
                    then
                    echo "efface le fichier tacherontab de
l'utilisateur" $user

# Changement de répertoire pour trouver le repertoire de
l'utilisateur concerné
                    cd /home
                    cd $user
                    cd Bureau/

# Suppression et création du fichier
                    cd `find etc/tacheron/`
                    rm tacherontab
                    touch tacherontab$user
```

Pour nous simplifier la tâche nous avons décidé de supprimer le fichier et de le recréer de façon vierge.

Pour éditer ou créer le fichier tacherontab de l'utilisateur, avec le paramètre « -e » :

```
               elif [ "$dernier_mot" = "-e" ]
                    then
                    echo "creation ou édition"

# Changement de répertoire pour trouver le repertoire de
l'utilisteur concerné
                    cd /home
```

```
                    cd $user
                    cd Bureau/

# Trouver le fichier /etc/tacheron/tachrontab$user
                    cd `find etc/tacheron/`
                    pwd
                    fichier=tacherontab$user

                    echo $fichier
```

Si on trouve le fichier, si il existe:

```
                    if [ -f $fichier ] ;
                         then
                         #echo "sa marche"
                         mkdir -p ../../tmp
                         cp $fichier ../../tmp/$fichier
                         vi ../../tmp/$fichier
                         cp ../../tmp/$fichier $fichier
                         echo "le fichier $fichier a été édité"
```

Si le fichier existe pas, on le créé :

```
                    else
                         echo "le fichier $fichier n'existe
pas"

                         mkdir -p ../../tmp
                         vi ../../tmp/$fichier
                         cp ../../tmp/$fichier $fichier
                         echo "le fichier $fichier a été créé
et édité"
                    fi
```

Nous avons eu quelques problèmes avec la commande tacherontab, comme nous avons changé de méthode de traitement des paramètres d'entrée, la commande ne fonctionne plus avec le fichier tacherontab par défaut.
Nous avons aussi commis un oubli de vérification, en effet si l'utilisateur n'existe pas dans le système, un message d'erreur s'affichera bien, car le répertoire que le programme va chercher dans l'arborescence le répertoire correspondant à etc/tacheron de l'utilisateur grâce à la commande « find », mais cependant il aurait été préférable de vérifier l'existence de l'utilisateur avant pour gagner du temps de traitement.

4) La commande tacheron

a) Autorisation des utilisateurs

- *L'usage de la commande « tacheron » est en principe réservé à root. On peut toutefois autoriser certains utilisateurs. Pour cela, on en dresse la liste sur des lignes successives dans le fichier /etc/tacheron.allow, et de façon symétrique, on peut mettre dans /etc/tacheron.deny la liste des utilisateurs non autorisés.*

Code

```
# création des fichiers allow et deny s'il n'existe pas

if [ ! -f etc/tacheron.allow ]
then touch etc/tacheron.allow
fi

if [ ! -f etc/tacheron.deny ]
then touch etc/tacheron.deny
fi

# on va vérifier si le user est autorisé

if [ $(whoami) = "root" ]; #l'utilisateur est le root
then
echo "Lancement du programme - root"
verif_syntax
else if awk 'BEGIN { print "Verification des Users dans etc/tacheron.allow";}
        $1 == $USER {print "User autorisé : "$0 }
        END   { print "Fin" }' etc/tacheron.allow
        then
             echo "lancement du programme"
             verif_syntax
else if awk 'BEGIN { print "Verification des Users dans etc/tacheron.deny";}
        $1 == $USER {print "User non autorisé : "$0 }
        END   { print "Fin" }' etc/tacheron.deny
        then
              echo "Vous n'êtes pas autorisé"
     fi
    fi
fi
```

On créé les fichiers tacheron.allow et tacheron.deny s'ils n'existent pas. La suite de la vérification s'appuiera sur un balayage dans un premier temps du fichier .allow et ensuite, de du fichier .deny. Le principe étant d'autoriser un utilisateur qui est dans les deux fichiers.

Inconvénient si on applique cette politique sur un système constitué de nombreuses personnes. En effet, si une totalité de 100 personnes on ne veut qu'en interdire l'accès à 2, il y a peu d'intérêt de créer le fichier .allow. Il suffirait de regarder ceux qui sont interdits et autoriser l'accès aux autres.

Si l'utilisateur est autorisé (ou root), on lance la fonction de vérification de la syntaxe du fichier /etc/tacheron/tacheron$user

b) Vérification de la syntaxe

- *Chaque ligne du fichier tacherontab contient 7 champs. Les 6 premiers champs déterminent les moments d'exécution de la tâche décrite au 7ème champ.*
- *Les 6 premiers, séparés par des espaces, appelés champs temporels, décrivent la périodicité : 15 secondes (0-3), minutes (0-59), heures (0-23), jour du mois (1-31), mois de l'année (1-12), jour de la semaine (0-6, 0=dimanche)*
- *Le 7ème est la commande à exécuter, ce peut être naturellement un script.*
- *Un champ temporel peut contenir :*
  - une valeur précise et valide pour le champ (par exemple 15 sur le champ minute)
  - une liste de valeurs valides, séparées par des virgules (1,3,5 dans le champ mois : janvier, mars, mai)
  - un intervalle valide (1-5 dans le champ jour : du lundi au vendredi)
  *- * pour signifier toutes les valeurs possibles du champ (* dans le champ minute : toutes les minutes)*
  *- */5 (dans le champ minutes : tous les 5 minutes), 0-23/3 (dans le champ heures : toutes les 3 heures)*
  - un ou plusieurs « ~nombre » peuvent être ajoutés afin de désactiver certaines valeurs dans l'intervalle. Par exemple, « 5-8~6~7 » est équivalent à « 5,8 ».

Code

```
# On vérifie si les taches de "tacherontab" sont sous la bonne forme

function verif_syntax {

function verif_jour {

if [ $(echo $jour | grep "^[0-9]\{1,2\}$") ] && [ $jour -le 6 ]; #nombre simple et inférieur à 6
then
commande
elif [ $( echo $jour | grep "^\([0-9]\{1,2\},\)*[0-9]\{1,2\}$") ]; #liste séparée par des virgules
then
commande
elif [ $( echo $jour | grep "^\([0-9]\{1,2\}-\)*[0-9]\{1,2\}$") ]; #intervale -
then
commande
elif awk '{if(( $6 == "x" ))}' etc/tacherontab #si seconde = *
then
commande
elif [ $(echo $jour | grep "^\x\/[0-9]\{1,2\}$") ]; # sous la forme */X avec X un nb
then
commande
elif [ $( echo $jour | grep "^\([0-9]\{1,2\}-[0-9]\{1,2\}\)\/[0-9]\{1,2\}$") ]; # X-Y/Z
then
commande
elif [ $( echo $jour | grep "^\([0-9]\{1,2\}-[0-9]\{1,2\}\)\{1\}\(~[0-9]\{1,2\}\)*$") ]; #X-Y~Z liste avec exception
then
```

```
commande
else
echo "Erreur syntaxique jour"
fi
}

function verif_mois {

if [ $(echo $mois | grep "^[0-9]\{1,2\}$") ] && [ $mois -le 12 ]; #nombre simple et inférieur à 12
then
verif_jour
elif [ $( echo $mois | grep "^\([0-9]\{1,2\},\)*[0-9]\{1,2\}$") ]; #liste séparée par des virgules
then
verif_jour
elif [ $( echo $mois | grep "^\([0-9]\{1,2\}-\)*[0-9]\{1,2\}$") ]; #intervale -
then
verif_jour
elif awk '{if(( $5 == "x" ))}' etc/tacherontab #si seconde = *
then
verif_jour
elif [ $(echo $mois | grep "^\x\/[0-9]\{1,2\}$") ]; # sous la forme */X avec X un nb
then
verif_jour
elif [ $( echo $mois | grep "^\([0-9]\{1,2\}-[0-9]\{1,2\}\)\/[0-9]\{1,2\}$") ]; # X-Y/Z
then
verif_jour
elif [ $( echo $mois | grep "^\([0-9]\{1,2\}-[0-9]\{1,2\}\)\{1\}\(~[0-9]\{1,2\}\)*$") ]; #X-Y~Z liste avec
exception
then
verif_jour
else
echo "erreur syntaxique mois"
fi

}

function verif_num_jour {

if [ $(echo $num_jour | grep "^[0-9]\{1,2\}$") ] && [ $num_jour -le 31 ]; #nombre simple et inférieur
à 31
then
verif_mois
elif [ $( echo $num_jour | grep "^\([0-9]\{1,2\},\)*[0-9]\{1,2\}$") ]; #liste séparée par des virgules
then
verif_mois
elif [ $( echo $num_jour | grep "^\([0-9]\{1,2\}-\)*[0-9]\{1,2\}$") ]; #intervale -
then
verif_mois
elif awk '{if(( $4 == "x" ))}' etc/tacherontab #si seconde = *
then
verif_mois
```

```
elif [ $(echo $num_jour | grep "^\x\/[0-9]\{1,2\}$") ]; # sous la forme */X avec X un nb
then
verif_mois
elif [ $( echo $num_jour | grep "^\([0-9]\{1,2\}-[0-9]\{1,2\}\)\/[0-9]\{1,2\}$") ]; # X-Y/Z
then
verif_mois
elif [ $( echo $num_jour | grep "^\([0-9]\{1,2\}-[0-9]\{1,2\}\)\{1\}\(~[0-9]\{1,2\}\)*$") ]; #X-Y~Z liste
avec exception
then
verif_mois
else
echo "erreur synatxique num jour"
fi

}

function verif_heure {

if [ $(echo $heure | grep "^[0-9]\{1,2\}$") ] && [ $heure -le 23 ]; #nombre simple et inférieur à 23
then
verif_num_jour
elif [ $( echo $heure | grep "^\([0-9]\{1,2\},\)*[0-9]\{1,2\}$") ]; #liste séparée par des virgules
then
verif_num_jour
elif [ $( echo $heure | grep "^\([0-9]\{1,2\}-\)*[0-9]\{1,2\}$") ]; #intervale -
then
verif_num_jour
elif awk '{if(( $3 == "x" ))}' etc/tacherontab #si seconde = *
then
verif_num_jour
elif [ $(echo $heure | grep "^\x\/[0-9]\{1,2\}$") ]; # sous la forme */X avec X un nb
then
verif_num_jour
elif [ $( echo $heure | grep "^\([0-9]\{1,2\}-[0-9]\{1,2\}\)\/[0-9]\{1,2\}$") ]; # X-Y/Z
then
verif_num_jour
elif [ $( echo $heure | grep "^\([0-9]\{1,2\}-[0-9]\{1,2\}\)\{1\}\(~[0-9]\{1,2\}\)*$") ]; #X-Y~Z liste avec
exception
then
verif_num_jour
else
echo "erruer syntaxique heure"
fi

}

function verif_minute {

if [ $(echo $minute | grep "^[0-9]\{1,2\}$") ] && [ $minute -le 59 ]; #nombre simple et inférieur à 60
then
verif_heure
```

```
elif [ $( echo $minute | grep "^\([0-9]\{1,2\},\)*[0-9]\{1,2\}$") ]; #liste séparée par des virgules
then
verif_heure
elif [ $( echo $minute | grep "^\([0-9]\{1,2\}-\)*[0-9]\{1,2\}$") ]; #intervale -
then
verif_heure
elif awk '{if(( $2 == "x" ))}' etc/tacherontab #si seconde = *
then
verif_heure
elif [ $(echo $minute | grep "^\x\/[0-9]\{1,2\}$") ]; # sous la forme */X avec X un nb
then
verif_heure
elif [ $( echo $minute | grep "^\([0-9]\{1,2\}-[0-9]\{1,2\}\)\/[0-9]\{1,2\}$") ]; # X-Y/Z
then
verif_heure
elif [ $( echo $minute | grep "^\([0-9]\{1,2\}-[0-9]\{1,2\}\)\{1\}\(~[0-9]\{1,2\}\)*$") ]; #X-Y~Z liste
avec exception
then
verif_heure
else
echo "erreur syntaxique minute"
fi

}

function verif_seconde {
while read C1 C2 C3 C4 C5 C6
do
seconde=$C1
minute=$C2
heure=$C3
num_jour=$C4
mois=$C5
jour=$C6

if [ $(echo $seconde | grep "^[0-9]\{1,2\}$") ] && [ $seconde -le 3 ]; #nombre simple et inférieur à 3
then
verif_minute
elif [ $( echo $seconde | grep "^\([0-9]\{1,2\},\)*[0-9]\{1,2\}$") ]; #liste séparée par des virgules
then
verif_minute
elif [ $( echo $seconde | grep "^\([0-9]\{1,2\}-\)*[0-9]\{1,2\}$") ]; #intervale -
then
verif_minute
elif awk '{if(( $1 == "x" ))}' etc/tacherontab #si seconde = *
then
verif_minute
elif [ $(echo $seconde | grep "^\x\/[0-9]\{1,2\}$") ]; # sous la forme */X avec X un nb
then
verif_minute
 elif [ $( echo $seconde | grep "^\([0-9]\{1,2\}-[0-9]\{1,2\}\)\/[0-9]\{1,2\}$") ]; # X-Y/Z
```

```
then
verif_minute
elif [ $( echo $seconde | grep "^\([0-9]\{1,2\}-[0-9]\{1,2\}\)\{1\}\(~[0-9]\{1,2\}\)*$") ]; #X-Y~Z liste avec exception
then
verif_minute
else
echo "erreur suntaqiue seconde"
fi

done <etc/tacheron/tacherontab$user
}

verif_seconde

}
```

On utilisera ici que des grep.

On balaie le fichier /etc/tacheron/tacherontab$user. On associe à chaque champ le domaine temporel qui lui est dédié (champ 1 pour les secondes, champ 2 pour les minutes…).

Si la syntaxe est bonne pour le premier élément temporel, c'est-à-dire les secondes, on passe à la vérification de la syntaxe de l'élément suivant (minutes) et ainsi de suite.

La première ligne conditionnelle permet de vérifier la périodicité : 15 secondes (0-3), minutes (0-59), heures (0-23), jour du mois (1-31), mois de l'année (1-12), jour de la semaine (0-6, 0=dimanche).
La seconde ligne conditionnelle permet de vérifier si le champ comporte une liste séparée par des virgules.
La troisième ligne permet de vérifier si le champ est intervalle constitué d'un « - »
La quatrième permet de vérifier si le champ est une « * ».
Ici, on a mis « x » car nous avons rencontré des problèmes lors de l'interprétation du caractère précédent.
La cinquième ligne permet de vérifier si le champ est sous la forme */X avec X un nombre pour créer une périodicité (ex : */3 dans le champ 3 représente toutes les 3 heures)
La sixième ligne est similaire à la cinquième condition à l'exception près que ce qui précède le « / » est un intervalle constitué de « - »
La septième ligne quant à elle, gère les exceptions. Un ou plusieurs « ~nombre » peuvent être ajoutés afin de désactiver certaines valeurs dans l'intervalle.

Si la syntaxe n'est pas bonne, on envoie un message d'erreur.

A la fin de la dernière vérification (celle du jour de la semaine), on lance la fonction commande.

c) Gestion des secondes

Code

```
#

function verifSeconde {

seconde_actuelle=$(date +'%S')

if [ "$seconde_actuelle" -le 14 ];then
  seconde_tacheron=0
elif [ "$seconde_actuelle" -le 29 ];then
  seconde_tacheron=1
elif [ "$seconde_actuelle" -le 44 ];then
  seconde_tacheron=2
else
  seconde_tacheron=3
fi
}
```

Le champ 1 de tacherontab est celui des secondes. La contrainte était de mettre ce champ soit à 0,1,2 ou 3 et non, pas des secondes directement. On a donc partagé les 60 secondes en 4 permettant d'avoir une périodicité de 15 secondes.

La première condition gère les premières 15 secondes, la deuxième, les 15 deuxièmes et ainsi de suite.

Cette fonction sera utilisée dans la fonction principale commande.

d) Comparaison des dates

Code

```
#permet de vérifier la date actulle avec la date située dans tacherontab

function dateOK() {

 date_actuelle="$1"
 date_tacheron="$2"
 return=1

 if [ "$date_tacheron" = "x" ]; then # contient une étoile
   return=0

 elif [ $(echo "$date_tacheron" | grep "^$date_actuelle$") ]; then # nombre simple
   return=0

 elif [ $(echo "$date_tacheron" | grep "$date_actuelle" | grep -v [-~/]) ]; then  #virgule
```

```
  IFS=',' read -a array <<< "$date_tacheron"
  for element in "${array[@]}"
   do
   if [[ "$element" -eq "$date_actuelle" ]];then
    return 0;
   fi;
  done

elif [ $(echo "$date_tacheron" | grep "-" | grep -v [,~/]) ]; then  # intervalles
  champs1=$(echo "$date_tacheron" | cut -d'-' -f1)
  champs2=$(echo "$date_tacheron" | cut -d'-' -f2)
  if [[ "$champs1" -le "$date_actuelle" ]] && [[ "$champs2" -ge "$date_actuelle" ]]; then
    return 0
  fi

  elif [ $(echo "$date_tacheron" | grep "~") ]; then  # "~"

     IFS='~' read -a array <<< "$date_tacheron"
     for element in "${array[@]}"
      do
            if [[ "$element" -eq "$date_actuelle" ]];then
                  return 1
                  fi

      done
          return 0;

elif [ $(echo "$date_tacheron" | grep "/") ];then # contient des slashs
   champs1=$(echo "$date_tacheron" | cut -d'/' -f1)
   champs2=$(echo "$date_tacheron" | cut -d'/' -f2)

   if [ "$champs1" = "x" ];then #commeance par *
      resultat=$(expr "$date_actuelle" % "$champs2")
      if [ "$resultat" -eq 0 ]; then
            return 0
      else
            return 1
      fi

   elif [ $(echo "$champs1" | grep "-" | grep -v [,~/]) ]; then #commence par intervalle
     c1=$(echo "$champs1" | cut -d'-' -f1)
     c2=$(echo "$champs1" | cut -d'-' -f2)
      if [ "$c1" -le "$date_actuelle" ] && [ "$c2" -ge "$date_actuelle" ]; then
            resultat=$(expr "$date_actuelle" % "$champs2")
            if [ "$resultat" -eq 0 ]; then
                  return 0
            else
                  return 1
            fi
      fi
```

```
   fi
 fi

 return $return

}
```

Cette fonction permet de vérifier si la date courante est la même que celle située dans le fichier tacherontab.
Cette fonction a deux arguments. Le premier étant la date courante et la deuxième, la date de la planification.
La première condition permet de vérifier si le champ est une « * »
La deuxième condition permet de comparer des nombres simples (sans intervalles…)
La troisième condition gère les listes constituées de « , ». On met dans un « tableau » chaque nombre située dans la liste. Et on vérifie pour chaque nombre si celui-ci est identique au nombre correspondant à la date courante.
La quatrième condition gère les intervalles. Si le champ du fichier tacherontab est constitué d'un « - » alors on compare la date courante avec les champs extrémums. Ainsi, si le nombre situé avant le – est inférieur ou égal à la date actuelle et le nombre situé après, est supérieur (ou égal) à la date courante, alors on se situe dans l'intervalle. >return 0. On peut donc lancer le programme.
La cinquième condition gère les exceptions. On utilise la même méthode que précédemment avec un array mais ici, si le nombre situé dans le tableau est le même que celui correspondant avec la date courante alors on ne lance pas la planification (car cela correspond à l'exception).
La sixième condition gère les intervalles ou * avec « / ». Si on est dans l'intervalle, et que le reste de la division est égale à 0, soit une division de la date courante avec le champ alors, on lance le planification.

Cette fonction est utilisée dans la fonction principale commande.

e) Fonction principale

Code

```
#fonction principale

function commande {

while [ 1 ]
do

while read line
do
minute_actuelle=$(date +'%M')
heure_actuelle=$(date +'%H')
```

```
num_jour_actuel=$(date +'%d')
mois_actuel=$(date +'%m')
jour_actuel=$(date +'%u')

seconde=$(echo $line | awk '{split($0,a," "); print a[1]}')
minute=$(echo $line | awk '{split($0,a," "); print a[2]}')
heure=$(echo $line | awk '{split($0,a," "); print a[3]}')
num_jour=$(echo $line | awk '{split($0,a," "); print a[4]}')
mois=$(echo $line | awk '{split($0,a," "); print a[5]}')
jour=$(echo $line | awk '{split($0,a," "); print a[6]}')
cmd=$(echo $line | cut -d" " -f7-)

if dateOK "$mois_actuel" "$mois" ; then
        if dateOK "$num_jour_actuel" "$num_jour" ; then
                if dateOK "$jour_actuel" "$jour" ; then
                        if dateOK "$heure_actuelle" "$heure" ; then
                                if dateOK "$minute_actuelle" "$minute"; then
                                  verifSeconde
                                        if dateOK "$seconde_tacheron" "$seconde" ; then
`$cmd`

erreur=`echo $?`
if [ "${erreur}" -eq 0 ]; then
message_error="Tache accomplie! Aucune erreur a l'execution"


# Ecriture dans le fichier /var/log/tacheron.txt, historique d'utilisation des taches planifiées.

echo "date d'execution:" `date` >> var/log/tacheron.txt
echo "nom du fichier: $cmd"  >> var/log/tacheron.txt
echo -e "$message_error\n" >> var/log/tacheron.txt

else
message_error="ERREUR: Le fichier n'a pu etre execute"


# Ecriture dans le fichier /var/log/tacheron.txt, historique d'utilisation des taches planifiées.

echo "date d'execution:" `date` >> var/log/tacheron.txt
echo "nom du fichier: $cmd"  >> var/log/tacheron.txt
echo -e "$message_error\n" >> var/log/tacheron.txt

fi
                                        fi
                                fi
                        fi
                fi
        fi
fi

done <etc/tacheron/tacherontab$user
```

```
done

}
```

Cette fonction tournera en boucle.

On balaie le fichier etc/tacheron/tacherontab$user et on vérifie à chaque fois si la date courante et la date du fichier tacherontab est la même pour chaque champ. Attention, on vérifie le champ des secondes après avoir lancer la fonction verifSeconde permettant de gérer la périodicité des 15 secondes.

Si les dates correspondent alors on peut lancer la commande (cmd correspondant au champ 7 du fichier tacherontab). Et on enregistre dans /var/log (déjà expliqué avant).

Conclusion

Ce projet a été très enrichissant dans la manière d'utiliser au mieux la programmation Linux et d'un planificateur de taches plus particulièrement.

Il nous a permis de voir comment utiliser au mieux les différentes et nombreuses commandes que Linux possède telles que grep et awk.

Il a soulevé bien des questions quant à la bonne utilisation et la gestion des caractères tel que « * » ou même, les expressions régulières. On a rencontré des problèmes au sujet de la gestion des utilisateurs avec la commande tacherontab.

On a donc trouvé un autre moyen en changeant le « * » par un « x ». Les différents tests nous ont permis de voir que le planificateur fonctionnait bien dans son ensemble avec les limites précédemment citées.