Séminaire technique
Shell - ssh

Pauline POMMERET

Encadrée par Pierre-Elliott BÉCUE

27 novembre 2012

Qu'est ce qu'un shell ?

Une *interface système*, ou *shell* est une couche logicielle qui fournit l'interface utilisateur d'un système d'exploitation. Il correspond à la couche la plus externe du système d'exploitation.
Le shell du système d'exploitation peut exister sous deux formes :

- interface en ligne de commande dite *CLI*, *Command Line Interface*, où l'utilisateur lance des instructions sous forme de texte ;
- interface graphique dite *GUI*, *Graphical User Interface*, où l'utilisateur sa souris et qui a pour mérite d'être intuitif.

Je vais parler du *shell* Unix.

Les différentes espèces de shell Unix

Il existe de nombreux shell :

- Shell de Stephen BOURNE
  - BOURNE shell (`/bin/sh`) : ancien shell par défaut, souvent shell par défaut pour `root` ;
  - BOURNE-Again shell (`/bin/bash`) : interprète par défaut (par défaut pour Mac OS X, Cygwin) ;
- *C shell* (`/bin/csh`) : évolution du shell `sh` avec une syntaxe plus proche du C ;
- KORN shell (`/usr/bin/ksh`) : compatible avec `bash`, incluant des fonctionnalités proche du `csh` ;
- Z shell (`/usr/bin/zsh`) : sorte de BOURNE shell étendu reprenant les fonctionnalités les plus pratiques de `bash`, `ksh` et `csh`, par défaut au CR@NS.

2 cas possibles : shell interactif ou non interactif

Shell interactif

```
TANT QUE <durée de la connexion>
   lire-ligne-au-clavier(ligne)
   traiter-ligne(ligne)
   afficher-résultats()
   afficher-prompt()
FIN TANT QUE
```

Shell non-interactif

```
traiter-ligne(ligne)
   POUR <premier champ de la ligne> DANS
      *<nom de fichier ascii>:
        TANT QUE <fichier ascii> NON VIDE
        traiter-ligne(<ligne courante du
fichier ascii>)
        FIN TANT QUE
      *<nom de binaire executable>:
        charger-et-lancer(<fichier binaire>)
      *<commande shell>:
        executer-commande()
      *AUTRE:
        afficher-message("command not
found")
   FIN TANT QUE
```

Principes généraux

Lors de son lancement, le shell ouvre 3 canaux :

- 0, l'entrée standard, `stdin`, qui est par défaut le clavier ;
- 1, la sortie standard, `stdout`, qui est par défaut l'écran ;
- 2, la sortie erreur standard, `stderr`, qui est par défaut l'écran.

Le shell lit ce qui se trouve sur son entrée standard.

Caractères de contrôle clavier essentiels

<tab>

<tab> permet de faire de la « tab-complétion » c'est-à-dire de compléter par exemple les noms de commandes, de fichiers, les chemins. SUPER UTILE ! !

^C

^C interrompt un processus attaché au terminal (SIGINT, signal 11)

^D

^D renvoie un caractère de fin de fichier (caractère ASCII 026), si le shell lit, il termine

Caractères de contrôle clavier essentiels

^Z

^Z suspend un processus en premier plan c'est-à-dire qu'il est mis en pause. Il reprend grâce à `fg`.

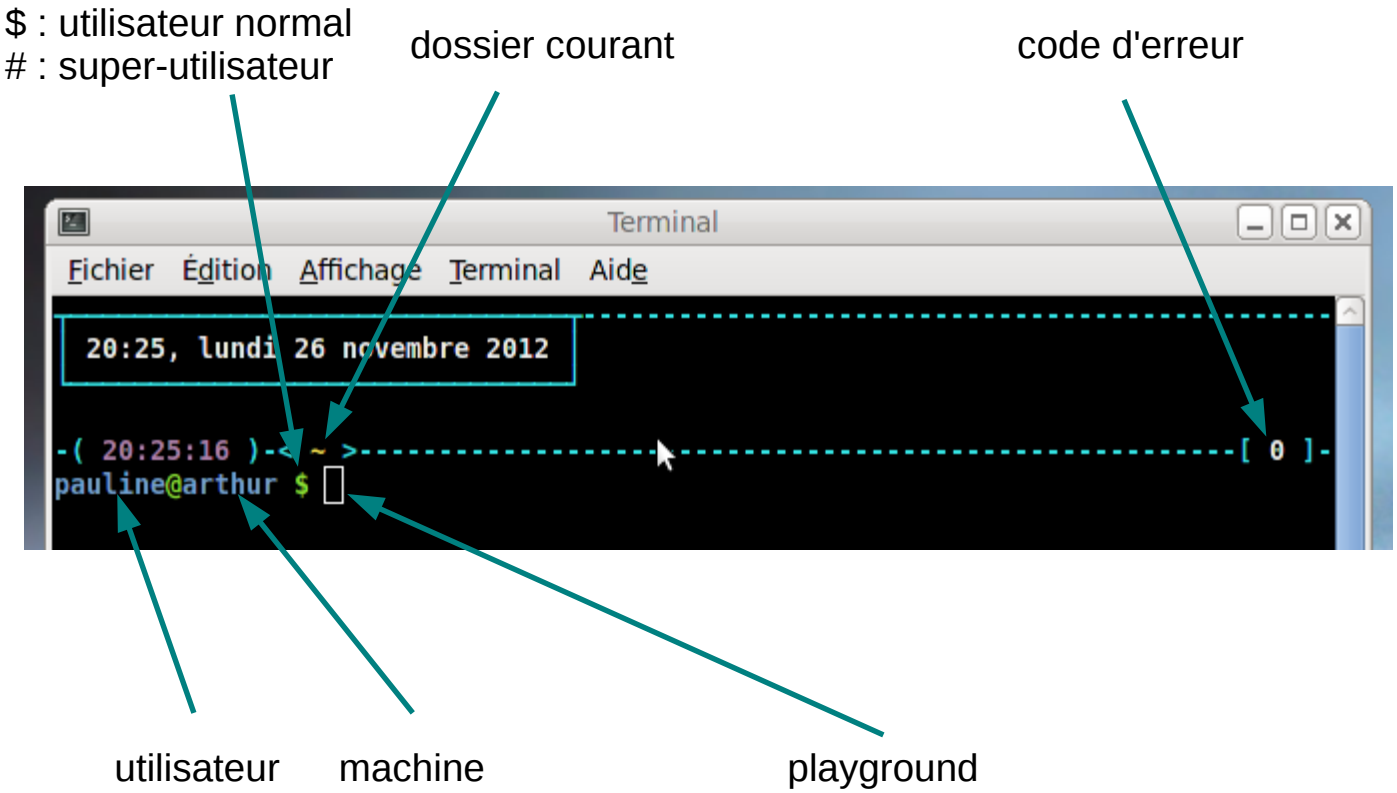
^A

^A permet de revenir au tout début de la ligne écrite.

^W

^W permet l'effacement du mot qui précède la position du curseur.

Présentation du terminal



Format d'une commande shell

- Une commande simple est une séquence de mots séparés par un séparateur blannc.
- Le premier mot désigne le nom de la commmande à exécuter, les mots suivants sont passés en arguments de la commande.
- La valeur retournée par la commande est celle de son exit.

Recherche par le shell d'une commande

Il y a deux cas possibles :

- si la commande est intégrée au shell, il l'exécute lui même,
- si la commande n'est pas intégrée au shell, le shell va la chercher dans le `PATH`.

le `PATH`

La variable `<PATH>` contient la liste des répertoires dans lesquels vont être recherchés les fichiers exécutables.
Si une commande n'est pas dans le `PATH`, il faut écrire tout le chemin jusqu'à la commande.

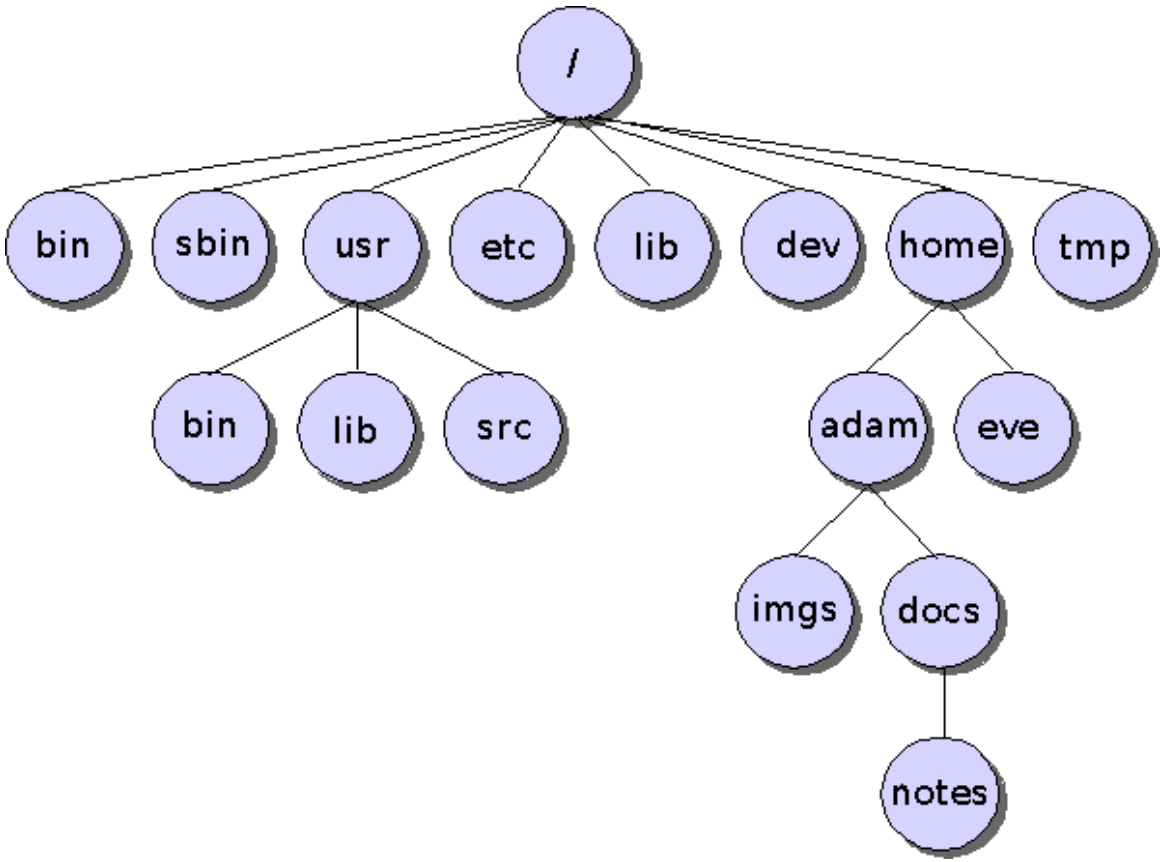
Pour visualiser le `PATH` :

```
echo $PATH
```

Pour ajouter un dossier au `PATH` :

```
export PATH=$PATH:/plif/plaf/plouf
```

Système de fichiers



Chemin relatif et chemin absolu

Le chemin absolu est la succession des répertoires à parcourir depuis la **racine** pour accéder au fichier spécifié.
Par exemple :

```
/home/pauline/Crans/Seminaires/shell-ssh.aux
```

Le chemin relatif est la successio des répertoires à parcourir depuis le **répertoire courant** pour accéder au fichier spécifié.
Par exemple :

```
../../L3-S1/Bioinfo/scripts/trx/trx.py
```

`..` est le répertoire parent

`cd`, *change directory*

définition

`cd` est une commande qui permet de naviguer dans l'arborescence des fichiers, connaissant l'emplacement du dossier que l'on cherche.

```
cd <chemin absolu ou relatif>
```

Que ce soit pour le chemin absolu ou pour le chemin relatif, il faut connaître l'arborescence des fichiers. Il ne faut surtout pas oublier d'utiliser la « tab-complétion », ça fait gagner beaucoup de temps.

`pwd`, *print working directory*

définition

`pwd` est une commande qui permet d'afficher le dossier courant, d'afficher où l'on est.

Cette commande est très pratique lorsque l'on ne dispose pas de `.bashrc` ou `.zshrc` « user-friendly » qui renouvelle l'affichage de la localisation à chaque retour de prompt.

`ls`, *lists segments*

`ls`

`ls` est une commande qui permet d'afficher le contenu d'un répertoire.

les options utiles

- `ls -a` affiche tous les fichiers et dossiers du répertoires (mêmes les cachés)
- `ls -l` affiche la liste des fichiers et des dossiers, avec leurs dates de dernière modification, leurs tailles, les utilisateurs propriétaires, groupe propriétaire et les droits.
- `ls -lh` même chose que précédemment, avec les tailles en format *human readable*.

`mv`, *move*

`mv`

`mv` peut servir à déplacer ou renommer des fichiers.

Exemples :

- `mv test plop` renomme le fichier/répertoire « test » en « plop »
- `mv plop /home/pauline/Test` déplace « plop » dans « Test »

les options

- `mv -i` *interactive* demande pour chaque fichier/répertoire s'il peut déplacer le fichier/répertoire
- `mv -u` *update* demande à `mv` de ne pas supprimer le fichier si la date de modification est la même ou plus récente que son remplaçant

`cp`, *copy*

définition

`cp` permet de copier un fichier ou un répertoire.

- `cp fichier1 fichier2` copie `fichier1` en un `fichier2`;
- `cp fichier répertoire` copie le fichier dans le répertoire.

les options

- `cp -i` avertit de l'existence d'un fichier du même nom et demande s'il peut remplacer son contenu.
- `cp -r` permet de copier de manière récursive l'ensemble d'un répertoire et de ses sous-répertoires.
- `cp -p` préserve toutes les informations comme le propriétaire, le groupe et la date de création.

touch

touch

`touch` sert à modifier le *timestamps* d'un fichier.

`touch test` va créer le fichier test dans le dossier courant, s'il n'existe pas encore.

les options

- `touch -t STAMP` utilise `STAMP` au lieu du temps présent.
- `touch -r plop -B 5 test` fait paraître le fichier test 5 secondes plus vieux que le fichier plop.
- `touch -r plop -F 5 test` fait paraître le fichier test 5 secondes plus jeune que le fichier plop.
- `touch -m` modifie la date de dernière modification.

`mkdir`, *make directory*

`mkdir`

`mkdir` permet de créer un répertoire.

```
mkdir <nom du répertoire>
```

les options

- `mkdir -p` permet de créer une suite de répertoires :

```
pwd
    /home/pauline
mkdir -p L3-S1/Biol336/Immuno/Diapos
```

`rm`, *remove*

`rm`

`rm` permet de supprimer un fichier.

```
rm <nom du fichier>
```

les options

- `rm -i` permet de demander à l'utilisateur s'il veut vraiment effacer le fichier.
- `rm -d` permet de supprimer un répertoire qu'il soit plein ou nous (dangereux).
- `rm -r` permet de supprimer un répertoire et ses sous-répertoires (très dangereux).
- `rm -f` permet de supprimer les fichiers protégés en écriture et les répertoires sans demander de confirmation (vraiment très dangereux)

`cat`, *concatenate*

`cat`

`cat` permet de concatener des fichiers ou de lire un fichier.

- `cat <nom du fichier 1> <nom du fichier 2>` permet de concaténer les deux fichiers.
- `cat <nom du fichier>` permet de l'afficher dans la sortie standard

les options

- `cat -n` permet de numéroter les lignes dans la sortie standard.

`less`, *less*

`less`

`less` lit au fur et à mesure le fichier qu'on lui donne et permet la navigation en amont et en aval.

les options

- `less /pattern <entrée>` permet de rechercher le *pattern* dans le fichier, en ayant son contexte.
- `less /pattern! <entrée>` permet de rechercher les lignes ne contenant pas *pattern*.
- `^D` permet d'avancer de N lignes (par défaut, la moitié de la taille de l'écran).
- `^B` permet de reculer de N lignes (par défaut, la moitié de la taille de l'écran).

`nano`, *nano*

`nano`

`nano` est un éditeur de texte, natif sur Ubuntu et Debian.

`nano <nom du fichier>` permet d'éditer le fichier en question.

les raccourcis

- `^O` permet d'écrire le fichier *i.e.* de sauvegarder.
- `^X` permet de fermer le fichier.
- `^K` permet de couper les lignes.
- `^U` permet de coller les lignes.

Ce qui est pratique avec `nano`, c'est qu'il y a toujours une anti-sèche...

`find`, *find*

`find`

`find` permet de trouver un fichier portant un nom donné.

`find / -name <nom du fichier> -print` permet de retrouver le fichier en question.

les options

- `find / -name 'plop*' -print` permet de chercher tous les fichiers dont le nom commence par « plop ».
- `find / -name bin -type d -print` permet de chercher tous les **répertoires** dont le nom est « bin ».
- `find / -name 'plop*' -print -ok rm` permet de supprimer tous les fichiers dont le nom commence par « plop ».

`grep`, *global regular expression print*

`grep`

`grep` cherche une expression rationnelle dans un fichier.

`grep <pattern> <nom du fichier>` permet de rechercher le « pattern » dans le fichier considéré.

les options

- `grep -n` permet d'afficher la ligne à laquelle le motif a été retrouvé dans le fichier.
- `grep -l bla /home/pauline/inutile/*` permet de donner les noms des fichiers où le motif « bla » est présent.
- `grep -c plop /home/pommeret/irclogs/Crans/2012/11/*` permet de compter le nombre de plop dans mes lofs du mois de novembre.

`chmod`, *change mode*

`chmod`

`chmod` permet de spécifier les droits qu'ont les utilisateurs sur un fichier.

$$- \underbrace{rwx}_{u} \mid \underbrace{rwx}_{g} \mid \underbrace{rwx}_{o}$$

u : propriétaire, g : groupe, o : autres utilisateurs

les options

- `chmod o +/- x/w/r` permet d'ajouter/enlever les droits d'exécution/écriture/lecture à « o ».
- `chmod 764` permet de donner tous les droits au propriétaire, le droit de lecture et d'écriture au groupe et le droit de lecture aux autres.

`chown`, *change owner*

`chown`

`chown` permet de définir le propriétaire et le groupe d'un fichier (nécessite d'être `root`).

```
ls -al
   -rw-r-r- pauline pauline 27 nov. 00:53 Doctor-who
chown pommeret.pauline Doctor-who
ls -al
   -rw-r-r- pommeret pauline 27 nov. 00:53 Doctor-who
```

les options

- `chown -R` permet de changer les permissions d'un répertoire et de ses sous-répertoires.

`sudo`, *substitute user do*

`sudo`

`sudo` permet d'exécuter des commandes qui ne peuvent être lancées qu'en étant `root`).

- `sudo nano /etc/apt/sources.list` pour executer une commande ponctuelle en tant que `root`.
- `sudo su` pour devenir `root` avec son propre mot de passe si l'on est dans le *sudoer file*.

le *sudoer file*

Le fichier de configuration de `sudo` est `/etc/sudoers`

```
# User privilege specification
root    ALL=(ALL) ALL
pauline ALL=(ALL) ALL

# Allow members of group sudo to execute any command
# (Note that later entries override this, so you might need to move
# it further down)
%sudo ALL=(ALL) ALL
```

`adduser`, *add user*

`adduser`

`adduser` permet d'ajouter un utilisateur à une machine.

```
adduser gribouille
```

- crée un répertoire `/home/gribouille`,
- ajoute l'utilisateur dans le fichier de configuration `/etc/passwd`,
- il reste à l'ajouter éventuellement au *sudoer file*.

`passwd`, *password*

définition

`passws` permet d'attribuer un mot de passe à un utilisateur ou de changer de mot de passe.

```
passwd gribouille
```

le `/etc/passwd`

Les mots de passe ne sont pas dans `/etc/passwd` mais dans `/etc/shadow`. `/etc/passwd` ressemble à ça :

```
saned:x:109:117::/home/saned:/bin/false
hplip:x:110:7:HPLIP system user,,,:/var/run/hplip:/bin/false
pauline:x:1000:1000:Pauline,,,:/home/pauline:/bin/bash
cl-builder:x:111:118::/usr/share/common-lisp/:/bin/false
```

`ssh`, *secure shell*

`ssh`

`ssh` est à la fois une commande et un protocole de communication sécurisée (les trames sont chiffrées) entre un client et un serveur distant. Le `ssh` permet de se connecter à une machine distante et d'y travailler.

```
ssh user@server.domain
```

les options

- `ssh -X` permet de transférer un accès au serveur X local à l'hôte distant (*X11forwarding*).
- `ssh -L port-local:Hostname:port-distant user@server.domain` permet de transférer les requêtes vers `Hostname` (tunnel).

`man`, *manual*

`man`

`man` est une commande qui permet d'ouvrir une « page de manuel ». Toutes les commandes possèdent une page de manuel qui explique leur fonctionnement et détaillent leurs options.

```
man <command>
```

les raccourcis

- `q` permet de quitter la page du manuel.
- ↑↓ permettent de naviguer dans les pages du manuel d'une commande.

Redirections

définition

Une redirection renvoie une entrée/sortie d'un fichier vers un autre fichier.

- `command > plop` renvoie la sortie standard de la commande vers `plop`, écrase le fichier `plop` s'il existe.
- `command » plop` ajoute la sortie standard de la commande au fichier `plop`.
- `command < plop` redirige l'entrée standard de la commande depuis le fichier `plop`.
- `2>/dev/null` redirige la sortie standard d'erreur vers `/dev/null` (« les égouts fg).

`pipe`, *pipeline*

`pipe`

`pipe` est une sorte de tuyau qui permet de renvoyer la sortie standard d'une commande vers l'entrée standard d'une autre.

- `ls -alh | grep 'pattern'` permet de retrouver toutes les lignes de `ls` contenant "pattern".

```
-( 17:32:21 )-< ~/L3/BV >-------------------------------------------------[ 0 ]-
pauline@arthur $ ls -alh | grep AB
-rw-r--r-- 1 pauline pauline  234  6 nov.  18:01 ABC_développement_floral.aux
-rw-r--r-- 1 pauline pauline 9,2K  6 nov.  18:01 ABC_développement_floral.log
-rw-r--r-- 1 pauline pauline 160K  6 nov.  18:01 ABC_développement_floral.pdf
-rw-r--r-- 1 pauline pauline 1,1K 26 nov.  09:15 ABC_développement_floral.tex
```

- `cat <fichier> | sendmail <email>`

Les métacaractères d'expansion

définition

Les métacaractères d'expansion sont des caractères, dont la signification peut dépendre de l'emplacement où l'on se trouve, qui permettent l'expansion des noms de fichiers.

- `*` remplace n'importe quelle chaîne de caractère ;
- `?` remplace n'importe quel caractère ;
- `[...]` permet de chercher une identité entre un caratère entre crochets et les mots donnés en argument : pour `pr[aio]*`, prie, proie, prisme et prison praire conviennent ;
- `[.-.]` permet de chercher une identité entre un caractère contenu entre les deux points et les mots donnés en argument : pour `pr[a-i]*`, prie, prisme, prison et praire conviennent ;
- `{..,..}` permet de chercher successivement deux chaînes de caractère : pour `pr{is,oi}*`, proie, prisme et prison conviennent ;
- ~ remplace par le `$HOME`.

||, &&, ;

||

`<command1> || <command2>` execute la première commande et la seconde, si et seulement si la première retourne un code de sortie différent de `0`.

&&

`<command1> && <command2>` execute la première commande et la seconde. L'exécution séquentielle s'arrête dès qu'une commande renvoie un code de sortie différent de `0`.

;

`<command1>; <command2>; <command3>` execute les commandes indépendamment les unes des autres.

Affectation de variables

Pour affecter une variable :

```
mavariable=contenu
```

où le contenu est au choix :

- une chaîne de caractères entre guillemets/apostrophes,
- un entier,
- la sortie d'une commande.

Pour afficher la valeur de la variable :

```
echo $mavariable
```

test conditionnel

test conditionnel

Un test conditionnel se préoccupe du code de sortie des commandes qu'il a en argument. Si la sortie est non nulle (`False`) il saute à l'étape suivante.
`elif` et `else` sont optionnels.

```
if <commande0>;
then <liste de commandeA>;
elif <commande1>;
then <liste de commandeB>;
else
<liste de commandeC>;
fi;
```

boucles `for`

boucles `for`

Une boucle `for` sert à executer un nombre défini à l'avance de fois ce qu'il y a dans le bloc.

```
for i in $(commande);
do <liste de commandeA>;
done;
```

boucles `while`

boucle `while`

Une boucle `while` sert à executer un nombre non défini à l'avance un bloc.

```
while <commande>;
do <liste de commande>;
done;
```

attention

Il est évident que cela peut être un peu dangeureux : si la commande ne renvoit jamais d'erreur, il ne reste plus que `Ĉ` pour l'arrêter. Mais cela peut être l'objectif...

fonction

définition

Une fonction est un objet qui prend des arguments et qui retourne une sortie.

```
inhome() {
if test $PWD = "/home/$USER";
then echo true;
else
echo false;
fi; }
```

Quand on aime pas le shell

Quand on connait pas du tout et/ou que l'on trouve ça moche (par exemple, pour les additions), il suffit de faire un script en autre chose, par exemple en python :

le script python

```
#!/usr/bin/env python
# -*- coding: utf8 -*-
import sys
if __name__ == '__main__'
  print sys.argv[1]
```

le script shell

```
#!/bin/bash
python /home/pauline/scripts/plop.py $@
```

Quand on aime

On peut faire des choses utiles. Par exemple, simplifier la vie de ceux qui veulent faire de l'addition d'entiers en shell :

script

```
#!/bin/bash
echo $(( $1 + $2 ))
```

Ensuite, il ne reste plus qu'à faire : `./plus 2 3` et ça donne `5`. C'est plus facile que : `$(( 2 + 3 ))`.

cron, *chronos*

`cron`

`cron` sert à executer périodiquement des commandes ou des scripts. C'est un programme avec de nombreux de fichiers de configuration (plusieurs pour la machine et un par utilisateur). Pour éditer son propre fichier de configuration : `crontab -e`.

```
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
# 30 *  * * *   /home/pauline/scripts/plop coucou (toutes les Xh30)
# 0 2  * * *   command (tous les jours à 2h)
# */5 *  * * *  commande (toutes les 5 minutes)
```

extrait de `crontab -e`

cron, *chronos*

`cron` pour la machine

Les fichiers de configuration de `cron` sont dans `/etc` et leur nom est `cron*`.

```
-( 19:59:52 )-< /etc/cron.daily >----------[ 0 ]-
pauline@arthur $ ls -al ./
total 84
drwxr-xr-x   2 root root  4096  5 nov.   22:18 ./
drwxr-xr-x 136 root root 12288 27 nov.   17:19 ../
-rwxr-xr-x   1 root root   311  2 nov.    2009 0anacron*
-rwxr-xr-x   1 root root 14799 15 avril  2011 apt*
-rwxr-xr-x   1 root root   314 11 août   2011 aptitude*
-rwxr-xr-x   1 root root   502 17 juin   2010 bsdmainutils*
-rwxr-xr-x   1 root root   256 10 nov.   2011 dpkg*
-rwxr-xr-x   1 root root  4109 12 mai    2011 exim4-base*
-rwxr-xr-x   1 root root    89 18 avril  2010 logrotate*
-rwxr-xr-x   1 root root  1335  3 janv.  2011 man-db*
-rwxr-xr-x   1 root root   606  3 nov.   2009 mlocate*
-rwxr-xr-x   1 root root   249 15 févr.  2011 passwd*
-rw-r--r--   1 root root   102 19 déc.   2010 .placeholder
-rwxr-xr-x   1 root root  2143  9 oct.   2010 popularity-contest*
-rwxr-xr-x   1 root root  3594 19 déc.   2010 standard*
```

contenu de `cron.daily`