

Les bases du système linux pour le calcul scientifique

G. Cougoulat / F. Audra

<https://pole-calcul-formation.gricad-pages.univ-grenoble-alpes.fr/ced/>

Séance 2 :

Notions de base sur les systèmes Linux

- ▶ Les éditeurs
- ▶ L'environnement et les variables
- ▶ Initialisation du shell
- ▶ Les Flux et la redirection
- ▶ Mon premier script shell

Séance 2 : Notions de base sur les systèmes Linux

Les éditeurs

► Le système GNU Linux possède de très nombreux éditeurs de textes.

○ **Vi** : (**Vim** dans sa version plus user friendly), présent par défaut sur un grand nombre de système Linux. Possède un grand nombre de fonctionnalités, qui peuvent être étendue par des plugins. C'est un éditeur modal (modes : normal, insertion, ligne de commande, etc...)

Vim possède un tutoriel que l'on lance avec la commande : ***vimtutor***

○ **Emacs** : éditeur très connu et très complet qui, comme Vim, peut étendre ses fonctionnalités via des plugins. Nécessite un certain temps d'apprentissage.

○ **Nano** : très basique, possède uniquement les fonctions indispensables. Très facile d'accès.

○ **Gedit, reedit, Kedit, Kate, etc...** Chaque distribution Linux ou environnement de bureau (Gnome, Kde, Xfce, etc...) propose son propre éditeur de texte

► Il est fortement conseillé de maîtriser au moins un des deux principaux éditeurs.

Séance 2 : Notions de base sur les systèmes Linux

L'environnement et les variables

- ▶ L'environnement est un ensemble de règles qui définissent le comportement du système pour un utilisateur. Ses règles sont fixées au moyen de « **variables d'environnement** ».

Exemple : la variable \$HOME définit l'emplacement du dossier personnel d'un utilisateur.

- ▶ **Portée des variables d'environnement** : elles sont valides pour le shell courant ainsi que tout ses shells enfants.
- ▶ La commande **env** permet de lister les variables d'environnement du processus shell courant.
- ▶ Les variables d'environnement sont déclarées de cette façon :

```
$ export $HOME=/home/mon_login  
$ export $PATH=/home/mon_login/bin:$PATH
```

- ▶ Il existe également des variables qui ont une portée limitée au processus courant. On peut lister l'ensemble des variables avec la commande **set**. Les variables se déclarent de cette façon:

```
$ ma_variable ="son contenu"
```

- ▶ La commande **unset** permet d'effacer cette déclaration.

Séance 2 : Notions de base sur les systèmes Linux

L'environnement et les variables

► Ci-dessous quelques variables d'environnement utiles :

\$HOME : Chemin FQDN vers le répertoire personnel de l'utilisateur

\$USER : Nom de l'utilisateur

\$SHELL : Nom du shell courant

\$PATH : Chemin d'accès aux programmes, par ordre de priorité

\$LD_LIBRARY_PATH : Chemin d'accès aux bibliothèques dynamiques

\$MANPATH : Chemin vers les pages du manuel

\$LANG : Langue

\$RANDOM : fonction du shell qui génère un entier compris entre 0 et 32767.

Séance 2 : Notions de base sur les systèmes Linux

Initialisation du shell

- ▶ Les variables d'environnement sont définies lors de différentes phases, par plusieurs fichiers de configuration :

Lors d'un shell de connexion bash (login shell) :

- Au niveau système (fichiers édités par l'administrateur) : /etc/profile ou /etc/bash.profile
- Au niveau utilisateur : /home/\$USER/.bash_profile ou /home/\$USER/.profile

Lors d'un shell bash interactif :

- Au niveau du système : /etc/bash.bashrc
- Au niveau utilisateur : /home/\$USER/.bashrc

- ▶ Bonne pratique : il est fortement recommandé de ne pas modifier ces fichiers spécifiquement à un projet ou un code. Il est préférable d'utiliser des fichiers d'environnement pour surcharger son environnement grâce à la commande source :

```
$ source ./mon_environnement.sh
```

- ▶ On peut également forcer la relecture de son fichier ~/.profile pour appliquer les modifications au shell

```
$ . ~/.profile
```

Séance 2 : Notions de base sur les systèmes Linux

Les flux et la redirection

► Schéma de principe :



► Les opérateurs de redirection de flux :

- > : rediriger stdout vers un nouveau fichier (le fichier sera écrasé s'il existe déjà)
- 2> : idem mais pour stderr.

- >> : rediriger stdout à la fin d'un fichier (le fichier sera créé s'il n'existe pas)
- 2>> : idem mais avec stderr

- < : lire stdin depuis un fichier
- << : lire depuis le clavier progressivement jusqu'à un indicateur de fin de fichier

○ Rediriger le flux des erreurs standard (2>) vers la sortie standard : **2>&1**

○ Rediriger tous les flux de sorties vers un fichier (/dev/null par exemple) : **> /dev/null 2>&1**

○ Chainer des commandes avec l'opérateur | :

le pipe permet de rediriger la sortie standard d'une commande vers l'entrée de la commande suivante.

Séance 2 : Notions de base sur les systèmes Linux

Les flux et la redirection

► Rediriger les sorties :

```
$ cut -d , -f 1 fichier_inexistant.csv > sortie.txt  
cut: fichier_inexistant.csv: Aucun fichier ou répertoire de ce type
```

```
$ cut -d , -f 1 fichier_inexistant.csv >> sortie.txt 2> erreurs.log
```

► Rediriger l'entrée standard :

Depuis un fichier :

```
$ sort < mes_donnees.csv
```

Depuis le clavier :

```
$ cat << FIN  
> Bonjour le monde  
> FIN  
Bonjour le monde
```


Séance 2 : Notions de base sur les systèmes Linux

Les flux et la redirection

► Eliminer un flux : redirection vers /dev/null

(Il est possible d'effectuer plusieurs redirection sur une seule commande).

```
$ cut -d , -f 1 fichier_inexistant.csv > sortie.txt 2> /dev/null
```

► Rediriger un flux vers un autre :

Redirection du flux d'erreur (stderr) vers la sortie standard (stdout).

```
$ cut -d , -f 1 fichier_inexistant.csv > sortie.txt 2>&1
```

► Tube de communication :

Redirection de la sortie d'une commande vers l'entrée de la commande suivante :

```
$ cat mes_donnees.csv | sort
```

► Regroupement de commandes :

```
$ ls -l ; cat mes_donnees.csv ;
```

Séance 2 : Notions de base sur les systèmes Linux

Mon premier script shell

► Définition :

Un script est un enchainement de commandes, regroupées dans un fichier en vue d'une exécution automatique.

► Créer un script :

Un script peut s'écrire dans un éditeur de texte basique. Par convention, la première ligne d'un script contient le « **shebang** » qui indique le shell à invoquer pour interpréter les commandes suivantes.

```
#!/bin/bash
```

► Exécuter un script :

○ En lançant un shell fils et en passant notre script en paramètre :

```
$ bash mon_script.sh
```

○ En ajoutant les droits d'exécution à notre fichier de script :

```
$ chmod +x ./mon_script.sh  
$ ./mon_script.sh
```

Séance 2 : Notions de base sur les systèmes Linux

Mon premier script shell

► Variables spéciales :

Lorsqu'on écrit un script shell on peut avoir besoin de données inhérentes au script. Il est possible d'y avoir accès grâce aux variables spéciales suivantes :

```
$0 : contient le nom du script
 $# : indique le nombre d'arguments passés au script

 $? : le code de retour de la dernière commande exécutée
 $$ : PID du processus en cours

 $n : correspond au nième argument d'une commande.
 $* ou @$ : renvoi la liste des arguments passés au script lors de son exécution.
```

► Codes de retour :

Tout processus renvoie un code de retour (entier positif ou négatif entre **0 et +-255**) qui nous renseigne sur l'exécution de ce programme.

Par convention, **0** indique que tout c'est déroulé correctement, toute autre valeur indique qu'il y a eu une erreur lors de l'exécution ou dans la syntaxe.

Séance 2 : Notions de base sur les systèmes Linux

Mon premier script shell

► Exemples :

```
$ ls -l /File_not_found
ls: cannot access /File_not_found: No such file or directory
$ echo $?
2

$ ls -l / > /dev/null
$ echo $?
0
```

► Codes de retour d'un script shell :

Dans un script shell, il peut être utile de fixer le code de retour pour signaler une erreur particulière à l'utilisateur. On utilise la commande **exit *n*** pour cela.

Dans le cas contraire, la variable **\$?** contient seulement le code retour de la dernière commande exécutée par le script.

Séance 2 : Notions de base sur les systèmes Linux

Mon premier script shell

► Les structures de contrôles : Tests

Plusieurs syntaxes pour effectuer des tests :

- `test expression` est équivalente à la syntaxe `[expression]`.
- Attention aux espaces autour de l'opérateur de comparaison.

```
$ # Numérique
$ test $nombre1 = nombre2
$ [ $nombre1 = $nombre2 ]

$ # Chaîne de caractères
$ [ "$nombre1" = "$nombre2" ]
```

```
$ # Tests avancés
$ # pour utiliser les opérateurs
$ # (>, <<, &&, ||, =~, !~, ...)
$ [[ $nombre1 > $nombre2 ]]
```

Séance 2 : Notions de base sur les systèmes Linux

Mon premier script shell

► Les opérateurs de tests :

	symbole	littérale	Signification
Nombres	=	-eq	égalité
	> , <	-gt , -lt	supérieur , inférieur (stricte)
	!=	-ne	différence
	=> , <=	-ge , -le	supérieur ou égale, inférieur ou égale
Chaines	== , !=		égale , différent
		-z , -n	vide , non vide
Logique	&&	-a	ET logique
		-o	OU logique

► **Attention** : Le test **-n** nécessite absolument que la chaîne de caractères soit entre guillemets à l'intérieur des crochets de test. (Une chaîne non initialisée n'est pas considérée comme vide mais comme **null**).

Placez toujours vos chaînes de caractères à tester entre guillemets.

Séance 2 : Notions de base sur les systèmes Linux

Mon premier script shell

► Les structures de contrôles : Condition

if ... then ... else

```
if [ $var = 2 ]
then
    echo "Egal à 2."
else
    echo "Différent de 2"
fi
```

Séance 2 : Notions de base sur les systèmes Linux

Mon premier script shell

► Les structures de contrôles : Boucle *for*

```
#!/bin/bash

# Ligne de commentaire
set -e # Indique au shell de s'arrêter en cas d'erreur

for ((i=0;i<3;i++))
do
    echo "A la $i"
done

exit 0 # Expliciter le code de sortie
```

```
for i in {1..3}
do
    echo "A la $i"
done
```


Séance 2 : Notions de base sur les systèmes Linux

Mon premier script shell

► Les structures de contrôles : Boucle *while* et *until*

```
#!/bin/bash

a_trouver=$((($RANDOM % 100) + 1))

echo "Entrez un nombre compris entre 1 et 100"
read i

while [ "$i" -ne "$a_trouver" ]; do
    if [ "$i" -lt "$a_trouver" ]; then
        echo "trop petit, entrez un nombre compris entre 1 et 100"
    else
        echo "trop grand, entrez un nombre compris entre 1 et 100"
    fi
    read i
done
echo "bravo, le nombre etait en effet $a_trouver »"
```

► La boucle *until* à le sens opposé à la boucle *while* : elle est exécutée tant que la condition est **fausse**.

Séance 2 : Notions de base sur les systèmes Linux

Mon premier script shell

► Caractères de protection :

Simple quotes vs double quotes :

```
$ export TEST="ma variable"  
$  
$ echo "$TEST"  
ma variable  
$  
$ echo '$TEST'  
$
```

L'antislash :

```
$ export TEST="ma variable"  
$  
$ echo "\$TEST"  
$
```