

TP1 n°3 : Shell GNU/Linux

○ Introduction :

Le **Bash** est un **shell**, c'est-à-dire un **interpréteur de commandes**, écrit pour le projet **GNU** en 1989.

Bash est l'acronyme de *Bourne-Again SHell*, un calembour sur le shell Bourne sh, qui a été historiquement le premier shell d'**Unix**. Le shell Bourne original fut écrit par **Stephen Bourne**. Bash a été principalement écrit par **Brian Fox** et **Chet Ramey**. La syntaxe de Bash est compatible avec sh et inclut des développements issus de csh, ksh et zsh...

Bash est le shell par défaut de la plupart des systèmes GNU/Linux, il est distribué sous la licence libre GNU GPL. Il fonctionne sur la plupart des systèmes d'exploitation de type Unix, et a également été porté sous Windows par **Cygwin**, puis sous windows 10 build 14316 par Microsoft.

Notes sur WSL (1 ou 2) : <https://docs.microsoft.com/fr-fr/windows/wsl/install-win10>

(installer le « man » : `sudo su -; apt-get update; apt-get install man-db; apt-get install aptitude; aptitude install "non du paquet"`)

Si un langage de programmation permet de dire à un processeur ce qu'il doit faire, **un interpréteur de commandes permet de dire à un système d'exploitation, un service ou à une application ce qu'ils doivent faire**. Cette nuance permet aux anglo-saxons de distinguer les notions de **programming** lorsqu'il s'agit d'écrire un code pour créer une application et de **scripting** lorsqu'il s'agit de programmer un système d'exploitation, un service ou une application. Le jeu d'instructions de Bash est donc spécialisé afin d'utiliser des systèmes Unix.

- <https://www.gnu.org/software/bash/>
- https://fr.wikipedia.org/wiki/Bourne-Again_shell
- https://fr.wikibooks.org/wiki/Programmation_Bash

○ Préambule :

1. **Ouvrir 2 terminaux :**

- 1 pour l'exécution des commandes et scripts,
- 1 pour les manuels...

2. **A l'URL : <https://http21.netSPACE.mc/uca/iut/rt/>**

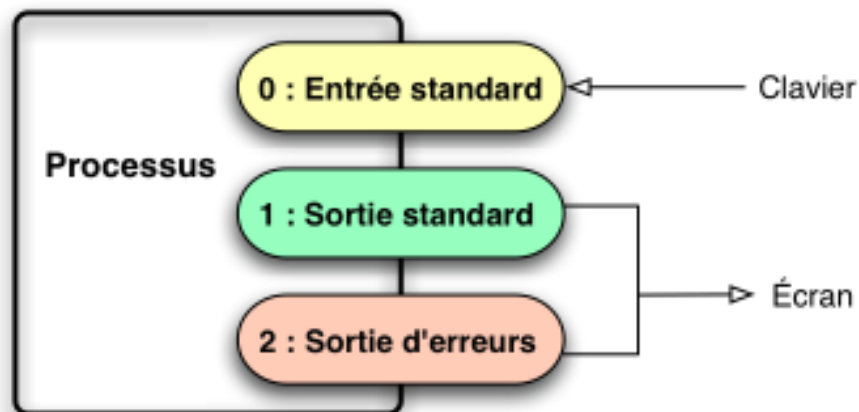
- télécharger la dernière version de **TP3-Shell_GNU_Linux.pdf**
- télécharger les *.sh contenus dans le dossier "shells/"

3. **Ouvrir un éditeur de texte, comme par exemple :**

- xedit (le plus ancien en mode graphique),
- emacs (un ancêtre pour le développement),
- Geany,
// Lancer Geany dans un terminal :
geany *.sh &
ou
emacs *.sh &
- ...

○ **Script en Shell Unix :**

- ◆ Les descripteurs de flux pour chaque processus :



- ◆ Les redirections :

- sortie standard (STDOUT) :
//
ls 1> ls.out
ou en général
ls > ls.out

```

# cat ls.out
ou pour paginer
# more ls.out
// Ajoute à la fin du fichier s'il existe ou le crée s'il
n'existe pas
# ls ../ >> ls.out
- erreur standard (STDERR) :
# ls -azerty 2> ls.err
- standard + erreur :
// On renvoi la sortie d'erreur (STDERR) la où pointe
STDOUT (i.e. descripteur 1)
# ls -lsa /* > ls.err.out 2>&1
# ls -lsa /* >> ls.err.out 2>&1
ou
# ls -lsa /* >& ls.err.out
# ls -lsa /* >>& ls.err.out
- // Fichier ou un flux vers l'entré standard (STDIN)
# cat < ls.err
- // Entrée de type "here document" :
# cat << _EOS_ > cat.out
mon
texte
est
celui ci !
_EOS_
# more cat.out
# cat << _EOS_ | mail -s "Mon sujet"
mon email...
_EOS_

```

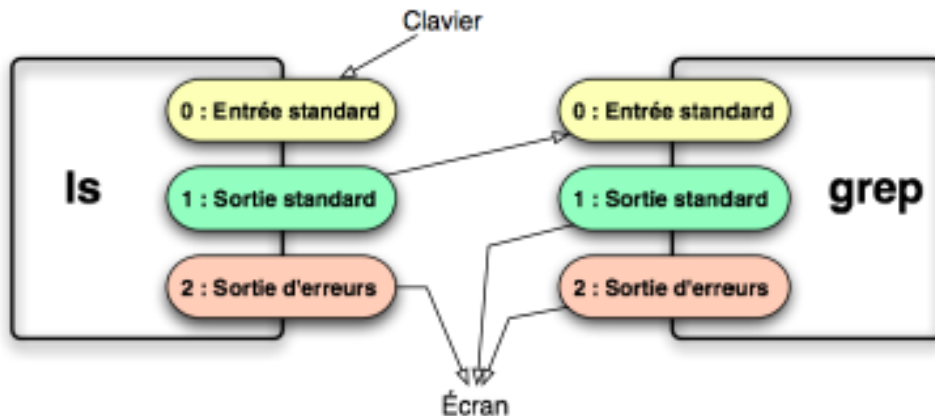
- ◆ le pipe (tube) | permet de **rediriger la sortie standard d'une commande vers l'entrée standard d'une autre commande** :


```

// Affiche tout les dossiers ou fichiers qui finissent par "n"
à la racine du système
# ls / | grep "n$"
ce qui équivaut à
# ls / > ls.out
# grep "n$" ls.out
// Affiche tout les dossiers ou fichiers qui commence par
"b" à la racine du système

```

c



◆ Enchaînement de scripts :

– En parallèle :

```
# ./script1.sh & ./script2.sh & ./script3.sh & ./script4.sh &
```

– Successivement :

```
# ./script1.sh; ./script2.sh; ./script3.sh; ./script4.sh
```

– **Successivement (tant qu'ils se terminent correctement => exit 0) :**

```
# ./script1.sh && ./script2.sh && ./script3.sh && ./script4.sh
```

– **Successivement (tant qu'ils se terminent mal => exit != 0) :**

```
# ./script3.sh || ./script2.sh || ./script1.sh || ./script4.sh
```

◆ Tuer un processus avec son PID (Processus ID) :

```
# kill -9 PID
```

○ **Les variables de l'environnements :**

◆ // Visualisation des variables d'environnement

```
# env
```

```
// Chemins d'accès aux exécutables...
```

```
# echo $PATH
```

```
// Dossier racine de l'utilisateur
```

```
# echo $HOME
```

```
// Dossier courant
```

```
# echo $PWD
```

```
// Paramètres de la connexion SSH
```

```
# echo SSH_CONNECTION
```

// etc...

- ◆ Définir une variable d'environnement :

// Assignment d'une variable

MA_VARIABLE="Ma Valeur"

// Affichage de la valeur de cette variable

echo \$MA_VARIABLE

ou

echo \${MA_VARIABLE}

MA_VARIABLE="Ma Valeur"; ./my_env.sh

>> MA_VARIABLE is not defined !

// Pour que MA_VARIABLE soit connu de tous les environnements fils de l'environnement de création de MA_VARIABLE, il faut faire un "export" :

export MA_VARIABLE

./ma_variable.sh

> MA_VARIABLE value is : Ma Valeur

- ◆ Persistance des variables définies par l'utilisateur :

cat ~/.bashrc

ou

cat ~/.profile

- ◆ Les « alias » (raccourcis de commandes) :

- // Liste des alias :

alias

- // Par exemple pour "piéger" certaines commandes :

alias rm="rm -i"

alias cp="cp -i"

alias mv="mv -i"

- Persistance des "alias" dans le fichier

"~/.bash_aliases"

- **Principales commandes :**

- // Visualisation "par page" d'un fichier

more /etc/passwd

ls -lsar /etc | more

- // Concaténation de plusieurs fichiers vers la sortie standard

cat /etc/passwd /etc/fstab

cat /etc/passwd /etc/fstab > passwd-fstab

- // Concaténation "inversée" de plusieurs fichiers vers la

- sortie standard
- # tac /etc/passwd
- // Extrait la dernière partie d'un fichier
- # tail -n 2 /etc/passwd
- // Extrait le début d'un flux
- # head -n 10 /etc/passwd
- // Recherche de dossier(s) ou fichier(s) avec des critères
- # man find
- // Recherche à la racine du système, sur une profondeur de 3 niveaux maxi, des dossiers qui ont une permission 700 :
- # find / -maxdepth 3 -type d -perm 700
- # Ne pas voir les erreurs
- # find / -maxdepth 3 -type d -perm 700 2> /dev/null
- // Recherche dans le dossier /etc, sur une profondeur de 2 niveaux maxi, de tout les fichiers.
- // On exécute une commande "ls -lsa" sur les résultats, symbolisés par {} :
- // Syntaxe si "beaucoup" de résultats (autant de commandes que de résultats)
- # find /etc -maxdepth 2 -type f -exec ls -lsa {} \;
- // Syntaxe avec pas "beaucoup" de résultats (une seule commande)
- # find /etc -maxdepth 2 -type f -exec ls -lsa {} +
- // Recherche dans le dossier /etc, sur une profondeur de 2 niveaux maxi, de tout les fichiers qui ont une permission 644. On exécute une commande "ls -lsa" sur les résultats.
- # find /etc -maxdepth 2 -type f -perm 644 -exec ls -lsa {} \;
- // Recherche de "motif(s)" dans un fichier texte ou dans un flux d'entrée
- # grep "root" /etc/passwd
- // avec une expression régulière
- # egrep "(root|www-data)" /etc/passwd
- # ls / | grep "n\$"
- // Sélectionner des "portions" de chaque ligne d'un fichier
- // par exemple les descriptions des utilisateurs :
- # cut -d: -f5 /etc/passwd
- // Transformation sur un flux de chaînes de caractères
- https://fr.wikipedia.org/wiki/Stream_Editor
- // Efface et affiche toute les lignes de /etc/passwd sauf

- celle finissant par "login"
- # sed -e '/login\$/!d' /etc/passwd
- # cut -d: -f7 /etc/passwd | sed -e 's/sbin/bin/i'
- # cut -d: -f7 /etc/passwd | sed -e 's:/usr/sbin/nologin:/bin/false:i'
- // Langage de traitement de lignes
<https://fr.wikipedia.org/wiki/Awk>
// Affiche la liste des utilisateurs
awk 'BEGIN{FS=":"; print "\nUsers list :\n"}{printf " - %s\n", \$1}END{print "\nThats All !!!\n"}' /etc/passwd | more
- ...

○ **Programmation en Shell Unix d'un script "Hello" :**

```
# ./hello.sh Jean-Michel
> Hello It's : Jean-Michel :)
```

- la première ligne d'un script Shell commence par :

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

ou pour d'autres langages interprétés (e.g. PHP, Python, Perl, Ruby, etc.) :

```
#!/usr/bin/php
#!/usr/bin/python
#!/usr/bin/perl
#!/usr/bin/ruby
```

- nano ou vi hello.sh

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

```
# Get first argument
firstname=$1
```

```
# Test on firstname and exit on error
[ -z "${firstname}" ] && echo ">> Firstname is empty !"
&& exit 1
```

```
# Ou avec un if...the...fi :
#if [ -z "${firstname}" ]; then
```

```
# echo ">> Firstname is empty !" && exit 1
#fi
```

```
# Display the message
echo "> Hello It's : $firstname !"
```

```
# Ou encore
#echo "> Hello It's : ${firstname} !"
```

```
# Exit without error
exit 0
```

- **Mode exécution sur hello.sh (uniquement pour moi, ici) :**

```
# chmod u+x hello.sh
# ./hello.sh
```

- **Programmation en Shell Unix d'un script qui effectue une opération arithmétique en deux nombres entiers (prendre hello.sh comme squelette) :**

https://fr.wikibooks.org/wiki/Programmation_Bash/Regex

```
# ./my_script.sh -a1 5 -a2 2 -o +
=> 5 + 2 = 7
# ./my_script.sh -o '*' -a2 5 -a1 2
=> 2 * 5 = 10
# ./my_script.sh -a2 3 -o / -a1 6
=> 6 / 3 = 2
// En option !
# ./my_script.sh
```

>> No input a1 parameters...

Usage : arithmetic operation between integer

```
my_script.sh -o {*,+,-,/} -a1 2 -a2 3
```

-o : arithmetic operator
-a1 : left operand

-a2 : right operand

```
# ./my_script.sh -o '*' -a1 5 -a2 2
```

- **Les arguments** sont ici dans l'ordre :

- \$1 ou \${1} : -o
- \$2 ou \${2} : *
- \$3 ou \${3} : -a1
- \$4 ou \${4} : 5
- \$5 ou \${5} : -a2
- \$6 ou \${6} : 2

- **La commande "shift" :**

// Exemple : arguments.sh

"shift" permet quant à elle de décaler les arguments du script. La valeur du 1er argument (i.e. \$1) est remplacée par la valeur du 2nd argument (i.e. \$2), etc.

Elle permet, en particulier, l'analyse des arguments d'un script ou d'une fonction...

On peut indiquer en argument de "shift" le nombre de pas (position) dont il faut décaler les arguments :

// Décalage de 2

shift 2

ou

shift; shift

- **Variables spéciales :**

- \$0 a pour valeur le nom du script ;
- \$# a pour valeur le nombre d'arguments passés au script ;
- \$@ contient la liste de tous les arguments du script.
- \$* contient la liste de tous les arguments du script (décomposée).
- **\$? renvoi le code de sortie de la dernière commande**

- **Les fonctions :**

// Exemple : arguments.sh

- déclaration :

```
myFonction() {
```

```
    a1=$1
```

```
    a2=$2
```

```

        ...
        an=$n
        instructions
    }
- appel :
  myFonction arg1 arg2 ... argn

```

- **Les tests :**

// Exemple : controls.sh

```

# man test
# test 4 -gt 2 && echo "vrai"
# [ 4 -lt 2 ] || echo "faux"

```

- **Les structures contrôles :**

// exemples : controls.sh

```

- if [ condition(s) ]; then
    instruction(s)
  elif [ condition(s) ]; then
    instruction(s)
  else
    instruction(s)
  fi

- case $a in
    '*') instruction(s) ;;
    +) instruction(s) ;;
    -|%) instruction(s) ;;
    /) instruction(s) ;;
    *) instruction(s) par défaut ;;
  esac

```

// exemples : loops.sh

```

- for variable in liste_valeur; do
    instruction(s)
  done

- for variable in {begin..end..increment}; do
    instruction(s)
  done

- for (( var=min; var <= max; var++ )); do

```

```
        instruction(s)
done
```

```
- while [ condition ]; do
        instruction(s)
done
```

- **Les opérations arithmétiques entières :**
// exemples : arguments.sh et loops.sh

```
result=$((votre opération))
```

```
// 4 * 2
# echo $((4 * 2))
8
# o='*'; a1=4; a2=6; result=$((a1 $o a2)); echo $result
24
```

- **Les opérations arithmétiques "décimales" avec "bc" :**
\$((echo "votre opération" | bc))
echo \$(echo "0.25 * 4" | bc)
a1=4.5; a2=2.5; result=\$(echo "\$a1 / \$a2" | bc); echo
\$result
o='/'; a1=4.5; a2=2.5; result=\$(echo "\$a1 \$o \$a2" | bc -
l); echo \$result