

IF110 - Le shell `bash`



Laurent Réveillère

Université de Bordeaux

laurent.reveillere@u-bordeaux.fr
<http://www.reveillere.fr/>

D'après le cours d'introduction aux systèmes d'exploitation de Télécom SudParis



Plan

- Terminal et shell
- Le langage `bash`
- Les variables
- Les structures algorithmiques
- Arguments d'une commande
- Commandes imbriquées

Terminal

- Porte d'entrée d'un ordinateur



- Un terminal offre :
 - un canal pour entrer des données (clavier, souris, écran tactile...)
 - un canal pour afficher des données (écran, imprimante, haut-parleur...)

IF110

3

Terminal

Un ordinateur n'a pas toujours un terminal intégré



*Bien que ce soit souvent le cas
(smartphone, tablette, ordinateur portable...)*

IF110

4

Un terminal peut être virtualisé

- Un terminal virtuel émule le comportement d'un terminal physique dans un autre terminal (virtuel ou physique)

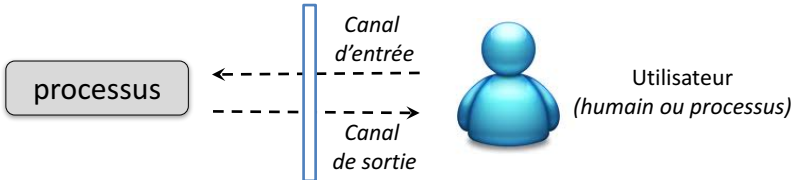



Terminaux virtuels

IF110 5

Processus et terminal

- Un processus communique avec l'utilisateur via un terminal
- On dit que le processus est attaché à un (et un seul) terminal



processus ← Canal d'entrée

→ Canal de sortie Utilisateur (humain ou processus)

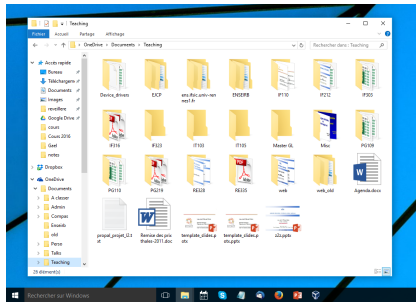
Terminal d'attache du processus

- Remarque:** lorsqu'un terminal est fermé, tous les processus attachés au terminal sont détruits

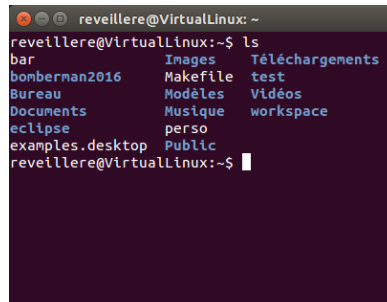
IF110 6

Le shell

Le shell est un programme permettant d'interagir avec les services fournis par un système d'exploitation



Shell en mode graphique
(Bureau windows, X-windows...)



Shell en mode texte
(bash, tcsh, zsh, ksh, cmd.exe...)

IF110

7

Plan

- ❑ Terminal et shell
- ❑ Le langage bash
- ❑ Les variables
- ❑ Les structures algorithmiques
- ❑ Arguments d'une commande
- ❑ Commandes imbriquées

IF110

8



Bourne-Again Shell (bash)

- ❑ Dans ce cours, nous étudions **bash**
 - shell en **mode texte**
 - En mode texte car permet d'écrire des scripts !
- ❑ Attaché à un terminal virtuel en mode texte

```
revellere@VirtualLinux: ~  
revellere@VirtualLinux:~$
```

IF110

9



Remarque importante

- ❑ Dans la suite du cours, nous utiliserons souvent le terme « shell » pour désigner le « Bourne-Again shell »
- ❑ Mais n'oubliez pas que `bash` n'est qu'un shell parmi de nombreux autres shells
 - `bash`, `tcsh`, `zsh`, `ksh`, `cmd.exe`...

IF110

10



Bash

- ❑ **Interpréteur de commandes**
 - Lit des commandes (à partir du terminal ou d'un fichier)
 - Exécute les commandes
 - Écrit les résultats sur son terminal d'attache

- ❑ **Bash définit un langage, appelé le langage bash**
 - Structures algorithmiques classiques (if, while, for, etc.)
 - Variables

- ❑ **Accès rapide aux mécanismes offerts par le noyau du système d'exploitation (tube, fichiers, redirections, ...)**

IF110

11



Un texte bash

- ❑ Un **texte** est formé de **mots bash**

- ❑ Un **mot bash** est
 - Formé de **caractères** séparés par des **blancs** (blanc : espace, tabulation, retour à la ligne)
Exemple : Coucou=42!* est un unique mot
 - Exceptions :
 - » ; & && | | | () ` sont des mots ne nécessitant pas de blancs
 - » Si une chaîne de caractères est entourée de de "" ou ' ', bash considère un unique mot

bash est sensible à la casse (c.-à-d., minuscule ≠ majuscule)

IF110

12



Texte bash

□ Un **texte** est formé de **mots**

```
Ici nous avons 5 mots
" En bash, ceci est un unique "mot" y compris mot milieu"
Voici, trois, mots
" zip "@é$èçà°-_"^$%ù£,..:+= ' est un autre unique mot'
Nous|avons;NEUF&&mots&ici
```

IF110

13



Un texte bash

□ Un **texte** est formé de **mots**

Attention :

Ce n'est pas parce qu'on écrit des mots que ces mots ont un sens pour bash

Exemple : `echo yop!3:bip` est constitué de deux mots mais n'est pas compréhensible par bash

IF110

14



Invocation d'une commande `bash`

□ Invocation d'une commande :

`cmd arg1 arg2...`

➤ *tout est optionnel sauf `cmd`*

➤ Lance la commande `cmd` avec les arguments `arg1`, `arg2`

```
$
```

IF110

15



Invocation d'une commande `bash`

□ Invocation d'une commande :

`cmd arg1 arg2...`

➤ *tout est optionnel sauf `cmd`*

➤ Lance la commande `cmd` avec les arguments `arg1`, `arg2`

```
$ echo Salut tout le monde
```

IF110

16



Invocation d'une commande `bash`

□ Invocation d'une commande :

`cmd arg1 arg2...`

➤ *tout est optionnel sauf `cmd`*

➤ Lance la commande `cmd` avec les arguments `arg1`, `arg2`

```
$ echo Salut tout le monde
```

IF110

17



Invocation d'une commande `bash`

□ Invocation d'une commande :

`cmd arg1 arg2...`

➤ *tout est optionnel sauf `cmd`*

➤ Lance la commande `cmd` avec les arguments `arg1`, `arg2`

```
$ echo Salut tout le monde
Salut tout le monde
```

IF110

18

Invocation d'une commande `bash`

- Invocation d'une commande :
 - `cmd arg1 arg2...`
 - *tout est optionnel sauf `cmd`*
 - Lance la commande `cmd` avec les arguments `arg1`, `arg2`

```
$ echo "Salut  tout le  monde"
```

IF110 19

Invocation d'une commande `bash`

- Invocation d'une commande :
 - `cmd arg1 arg2...`
 - *tout est optionnel sauf `cmd`*
 - Lance la commande `cmd` avec les arguments `arg1`, `arg2`

```
$ echo "Salut  tout le  monde"
```

IF110 20



Invocation d'une commande `bash`

□ Invocation d'une commande :

`cmd arg1 arg2...`

➤ *tout est optionnel sauf `cmd`*

➤ Lance la commande `cmd` avec les arguments `arg1`, `arg2`

```
$ echo "Salut tout le monde"
Salut tout le monde
```



La première commande à connaître

□ `man 1 cmd`

➤ `man` pour manuel : donne de l'aide

➤ `1` (optionnel) indique la section d'aide de la commande

» `1` : commandes

➤ `cmd` est la commande dont on veut consulter le manuel

```
$ man ls
```

Première commande à connaître

- ❑ `man 1 cmd`
 - `man` pour manuel : donne de l'aide
 - `1` (optionnel) indique la section d'aide de la commande
 - » `1` : commandes
 - `cmd` est la commande dont on veut consulter le manuel

```

reveillere@VirtualLinux: ~
ls(1)                                User Commands                                L(1)
NAME
  ls - list directory contents

SYNOPSIS
  ls [OPTION]... [FILE]...

DESCRIPTION
  List information about the FILES (the current directory by
  default). Sort entries alphabetically if none of -cftuvSUX
  nor --sort is specified.

Mandatory arguments to long options are mandatory for short
Manual page ls(1) line 1 (press h for help or q to quit)

```

IF110 23

Caractères spéciaux de bash

- ❑ **Caractères spéciaux**

```
\ ' ` " > < $ # * ~ ? ; ( ) { }
```

 - `'` est appelé quote ou apostrophe alors que ``` est appelé antiquote (accent grave)
 - `\` est appelé backslash ou contre-oblique
 - Explication de chacun donnée dans la suite du cours
- ❑ **Désactiver l'interprétation des caractères spéciaux**
 - `\` désactive l'interprétation spéciale du caractère suivant
 - `'...'` ⇒ désactive l'interprétation dans toute la chaîne
 - `"..."` ⇒ seuls sont interprétés les caractères `$ \ ``

IF110 24

Script bash

❑ Programme `bash` = texte `bash` dans un fichier texte

- Interprétable par `bash` au lancement par l'utilisateur
- Modifiable par un éditeur de texte (p. ex. `emacs`, `vi`, mais pas `word` !)
- Un programme `bash` doit être rendu exécutable avec :
`chmod u+x mon_script.sh`
(notion vue dans le cours sur le système de fichiers)
- Par convention, les noms de script sont suffixés par l'extension « `.sh` »
» p. ex., `mon_script.sh`

❑ Invocation du script nommé `mon_script.sh` avec

- `./mon_script.sh` ./ indique que le script se trouve dans le répertoire courant (notion vue plus loin)
- Avec ses arguments :
`./mon_script.sh arg1 arg2`

IF110

25

Structure d'un script bash

❑ Première ligne : `#! /bin/bash`

- `#!` : indique au système que ce fichier est un ensemble de commandes à exécuter par l'interpréteur dont le chemin suit
» par exemple : `/bin/sh`, `/usr/bin/perl`, `/bin/awk`, etc.
- `/bin/bash` lance `bash`

❑ Puis séquence structurée de commandes shell

```
#! /bin/bash  
  
commande1  
commande2  
...  
mon_script.sh
```

❑ Sortie implicite du script à la fin du fichier

- Sortie explicite avec la commande `exit`

IF110

26



Plan

- ❑ Terminal et shell
- ❑ Le langage bash
- ❑ Les variables
- ❑ Les structures algorithmiques
- ❑ Arguments d'une commande
- ❑ Commandes imbriquées

IF110

27



Variables bash

- ❑ Déclaration/affectation avec =
`ma_var=valeur`
- ❑ Consultation en préfixant du caractère \$
`$ma_var`
- ❑ Saisie interactive
`read var1 var2 ... varn`
 - Lecture d'une ligne saisie par l'utilisateur (jusqu'au retour chariot)
 - Le premier mot va dans `var1`
 - Le second dans `var2`
 - Tous les mots restants vont dans `varn`

IF110

28

Variables bash

- Attention :
- Pas de blanc dans `ma_var=valeur`
- Pas de blanc dans `$ma_var`
- (dans les deux cas, bash interprète de façon spéciale un unique mot)

Leur
_var
varn
tour

- Le second dans `var2`
- Tous les mots restants vont dans `varn`

IF110 29

Variables bash - exemple

```
$
```

IF110 30



Variables bash - exemple

```
$ a=42  
$
```

IF110

31



Variables bash - exemple

```
$ a=42  
$ echo $a  
42  
$
```

IF110

32



Variables bash - exemple

```
$ a=42
$ echo $a
42
$ s='Bonjour, monde!!!'
$
```

IF110

33



Variables bash - exemple

```
$ a=42
$ echo $a
42
$ s='Bonjour, monde!!!'
$ echo $s
Bonjour, monde!!!
$
```

IF110

34



Variables bash - exemple

```
$ a=42
$ echo $a
42
$ s='Bonjour, monde!!!'
$ echo $s
Bonjour, monde!!!
$ read x
Ceci est une phrase ← Saisi par l'utilisateur
$
```

IF110

35



Variables bash - exemple

```
$ a=42
$ echo $a
42
$ s='Bonjour, monde!!!'
$ echo $s
Bonjour, monde!!!
$ read x
Ceci est une phrase
$ echo $x
Ceci est une phrase
$
```

IF110

36



Variables bash - exemple

```
$ a=42
$ echo $a
42
$ s='Bonjour, monde!!!'
$ echo $s
Bonjour, monde!!!
$ read x
Ceci est une phrase
$ echo $x
Ceci est une phrase
$ read x y
Ceci est une phrase ← Saisi par l'utilisateur
$
```

IF110

37



Variables bash - exemple

```
$ a=42
$ echo $a
42
$ s='Bonjour, monde!!!'
$ echo $s
Bonjour, monde!!!
$ read x
Ceci est une phrase
$ echo $x
Ceci est une phrase
$ read x y
Ceci est une phrase
$ echo $x ← Premier mot
Ceci
$
```

IF110

38



Variables bash - exemple

```
$ a=42
$ echo $a
42
$ s='Bonjour, monde!!!'
$ echo $s
Bonjour, monde!!!
$ read x
Ceci est une phrase
$ echo $x
Ceci est une phrase
$ read x y
Ceci est une phrase
$ echo $x ← Premier mot
Ceci
$ echo $y ← Tous les mots qui suivent
est une phrase
```

IF110

39



Plan

- Terminal et shell
- Le langage bash
- Les variables
- Les structures algorithmiques
- Arguments d'une commande
- Commandes imbriquées

IF110

40



Schéma algorithmique séquentiel

- Suite de commandes les unes après les autres
 - Sur des lignes séparées
 - Sur une même ligne en utilisant le caractère point virgule (;) pour séparateur

IF110

41



Schéma alternatif (if)

- Schéma alternatif simple
 - Si alors ... sinon (si alors ... sinon ...)
 - `elif` et `else` sont optionnels

```
if cond; then
  cmds
elif cond; then
  cmds
else
  cmds
fi
```

IF110

42



Conditions de test

□ Tests sur des valeurs numériques

- [n1 **-eq** n2] vrai si n1 est égal à n2
- [n1 **-ne** n2] vrai si n1 est différent de n2
- [n1 **-gt** n2] vrai si n1 supérieur strictement à n2
- [n1 **-ge** n2] vrai si n1 supérieur ou égal à n2
- [n1 **-lt** n2] vrai si n1 inférieur strictement à n2
- [n1 **-le** n2] vrai si n1 est inférieur ou égal à n2

□ Tests sur des chaînes de caractères

- [mot1 **=** mot2] vrai si mot1 est égale à mot2
- [mot1 **!=** mot2] vrai si mot1 n'est pas égale à mot2
- [**-z** mot] vrai si mot est le mot vide
- [**-n** mot] vrai si mot n'est pas le mot vide

IF110

43



Conditions de test

□ Tests sur des valeurs numériques

- [n1 **-eq** n2]

Attention :

Les blancs sont essentiels !

□

- [mot1 **!=** mot2] : vrai si mot1 n'est pas égale à mot2
- [**-z** mot] : vrai si mot est le mot vide
- [**-n** mot] : vrai si mot n'est pas le mot vide

IF110

44



Remarque sur les conditions

- ❑ `[cond]` est un raccourci pour la commande `test cond`
- ❑ `test` est une commande renvoyant vrai (valeur 0) ou faux (valeur différente de 0) en fonction de l'expression qui suit

```
if [ $x -eq 42 ]; then
    echo coucou
fi
```

≈

```
if test $x -eq 42; then
    echo coucou
fi
```

IF110

45



Schéma alternatif simple (if)

- ❑ Schéma alternatif simple
 - Si alors ... sinon (si alors ... sinon ...)
 - `elif` et `else` sont optionnels

```
if cond; then
    cmds
elif cond; then
    cmds
else
    cmds
fi
```

```
x=1
y=2
if [ $x -eq $y ]; then
    echo "$x = $y"
elif [ $x -ge $y ]; then
    echo "$x > $y"
else
    echo "$x < $y"
fi
```

IF110

46

Schéma alternatif multiple (case)

□ Schéma alternatif multiple

- Si mot vaut motif1 ...
Sinon si mot vaut motif2 ...
Sinon ...
- Motif : chaîne de caractères pouvant utiliser des méta-caractères (voir plus loin)
- *) correspond au cas par défaut

```
case mot in
  motif1)
    ...;;
  motif2)
    ...;;
  *)
    ...;;
esac
```

IF110

47

Schéma alternatif multiple (case)

□ Schéma alternatif multiple

- Si mot vaut motif1 ...
Sinon si mot vaut motif2 ...
Sinon ...

```
case mot in
  motif1)
    ...;;
  motif2)
    ...;;
  *)
    ...;;
esac
```

```
lang="fr"
case $lang in
  "fr")
    echo "Bonjour";;
  "it")
    echo "Ciao";;
  *)
    echo "Hello";;
esac
```

IF110

48



Schémas itératifs (boucle `while`)

□ `while`

- Tant que ... faire ...
- Mot clé `break` pour sortir de la boucle

```
while cond; do  
  cmds  
done
```



Schémas itératifs (boucle `while`)

□ `while`

- Tant que ... faire ...
- Mot clé `break` pour sortir de la boucle

```
while cond; do  
  cmds  
done
```

```
x=10  
while [ $x -ge 0 ]; do  
  read x  
  echo $x  
done
```



Schémas itératifs (boucle `for`)

□ `for`

- Pour chaque ... dans ... faire ...
- `var` correspond à la variable d'itération
- `liste` : ensemble sur lequel `var` itère

```
for var in liste; do
  cmds
done
```

IF110

51



Schémas itératifs (boucle `for`)

□ `for`


- Pour chaque ... dans ... faire ...
- `var` correspond à la variable d'itération
- `liste` : ensemble sur lequel `var` itère

```
for var in liste; do
  cmds
done
```

```
for var in 1 2 3 4; do
  echo $var
done
```

IF110


52



Plan

- ❑ Terminal et shell
- ❑ Le langage bash
- ❑ Les variables
- ❑ Les structures algorithmiques
- ❑ Arguments d'une commande
- ❑ Commandes imbriquées

IF110 53



Arguments d'une commande

- ❑ `mon_script.sh arg1 arg2 arg3 arg4 ...`
 ⇒ chaque mot est stocké dans une variable numérotée

<code>mon_script.sh</code>	<code>arg1</code>	<code>arg2</code>	<code>arg3</code>	<code>arg4</code>	...
<code>"\$0"</code>	<code>"\$1"</code>	<code>"\$2"</code>	<code>"\$3"</code>	<code>"\$4"</code>	...

- `"$0"` toujours le nom de la commande
- `"$1"` ... `"$9"` : les paramètres de la commande
- `$#` : nombre de paramètres de la commande
- `"$@"` : liste des paramètres : `"arg1" "arg2" "arg3" "arg4" ...`
- `shift` : décale d'un cran la liste des paramètres

IF110 54



Arguments d'une commande

```
#!/bin/bash
for i in "$@"; do
  echo $i
done
```

mon_echo.sh

```
$
```

IF110

55



Arguments d'une commande

```
#!/bin/bash
for i in "$@"; do
  echo $i
done
```

mon_echo.sh

```
$. /mon_echo.sh
$
```

IF110

56



Arguments d'une commande

```
#!/bin/bash
for i in "$@"; do
  echo $i
done
```

mon_echo.sh

```
$/mon_echo.sh
$/mon_echo.sh toto titi
toto
titi
$
```

IF110

57



Arguments d'une commande

```
#!/bin/bash
for i in "$@"; do
  echo $i
done
```

mon_echo.sh

```
$/mon_echo.sh
$/mon_echo.sh toto titi
toto
titi
$/mon_echo "fin de" la demo
fin de
la
demo
$
```

IF110

58



Plan

- ❑ Terminal et shell
- ❑ Le langage bash
- ❑ Les variables
- ❑ Les structures algorithmiques
- ❑ Arguments d'une commande
- ❑ Commandes imbriquées

IF110

59



Imbrication de commandes

- ❑ Pour récupérer le texte écrit sur le terminal par une commande dans une chaîne de caractères
 - \$ (cmd)
 - Attention à ne pas confondre avec \$cmd qui permet l'accès à la valeur de la variable cmd

IF110

60



Imbrication de commandes

- ❑ Pour récupérer le texte écrit sur le terminal par une commande dans une chaîne de caractères
 - \$ (cmd)
 - Attention à ne pas confondre avec \$cmd qui permet l'accès à la valeur de la variable cmd

```
$ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$
```

IF110

61



Imbrication de commandes

- ❑ Pour récupérer le texte écrit sur le terminal par une commande dans une chaîne de caractères
 - \$ (cmd)
 - Attention à ne pas confondre avec \$cmd qui permet l'accès à la valeur de la variable cmd

```
$ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$ echo "Nous sommes le $(date). "
Nous sommes le lundi 27 juillet 2015, 12:47:06
(UTC+0200).
$
```

IF110

62



Imbrication de commandes

- ❑ Pour récupérer le texte écrit sur le terminal par une commande dans une chaîne de caractères

- \$ (cmd)

- Attention à ne pas confondre `$(date)` et `date`.

- l'accès à la valeur

Attention, récupère la variable `date` et non le résultat de la commande `date`.

```
$ date
lundi 27 juillet 2015, 12:47:06 (UTC+0200)
$ echo "Nous sommes le $(date). "
Nous sommes le lundi 27 juillet 2015, 12:47:06
(UTC+0200).
$ echo "Nous sommes le $date."
Nous sommes le .
$
```

IF110

63



Conclusion

- ❑ Concepts clés

- Terminal, shell
- Interpréteur de commande `bash`
 - » Commandes, langage `bash`
- Documentation
- Caractères spéciaux de `bash`
- Script `bash`

- ❑ Commandes clés

- `man`, `bash`, `echo`, `read`

- ❑ Commandes à connaître

- `date`

IF110

64