

Shell: Interpréteur de commande

M1105 - Systèmes d'exploitation

`Sebastien.Kramm@univ-rouen.fr`

IUT de Rouen, dépt. Réseaux & Télécoms

Version du 17 octobre 2018

Licence

Ce document est placé sous licence [CC BY-NC-SA]
(Attribution - Pas d'utilisation commerciale - Partage dans les mêmes conditions)



Pour plus de détails, [voir la page Creative Commons](#).

Production

Ce document est généré à partir du fichier source Latex en 3 versions :

- Une version pour le cours lui-même → suffixée par "_B"
- Une version pour l'impression, avec 4 diapos par page A4 → suffixée par "_P"
- Une version pour la lecture à l'écran, similaire à la première mais sans les animations → suffixée par "_H"

Information

Ce document contient des liens vers des pages ressources, qui apparaissent avec une couleur distinctive.

Sommaire

- 1 Introduction
- 2 Commandes de base
- 3 Enchaînement et redirection
- 4 Variables
- 5 Scripts : structures de contrôle
 - Windows
 - Bash

Introduction : besoin ?

Un OS doit permettre à un utilisateur humain de :

- lancer des programmes
 - programmes applicatifs de type *desktop* (interactifs)
 - programmes applicatifs de type serveur (non interactifs)
 - Utilitaires de gestion
 - etc.
- Manipuler le système de fichier (copier, renommer, ...)

Introduction : besoin ?

Un OS doit permettre à un utilisateur humain de :

- lancer des programmes
 - programmes applicatifs de type *desktop* (interactifs)
 - programmes applicatifs de type serveur (non interactifs)
 - Utilitaires de gestion
 - etc.
- Manipuler le système de fichier (copier, renommer, ...)

Ceci se fait via une **Interface Homme - Machine** (IHM)
ou *User Interface* (UI) en anglais, via deux paradigmes de contrôle :

- Interface graphique ou *Graphical User Interface* : **GUI**
- Interface en ligne de commande ou *Command Line Interface* : **CLI**

Introduction : besoin ?

Un OS doit permettre à un utilisateur humain de :

- lancer des programmes
 - programmes applicatifs de type *desktop* (interactifs)
 - programmes applicatifs de type serveur (non interactifs)
 - Utilitaires de gestion
 - etc.
- Manipuler le système de fichier (copier, renommer, ...)

Ceci se fait via une **Interface Homme - Machine** (IHM)
ou *User Interface* (UI) en anglais, via deux paradigmes de contrôle :

- Interface graphique ou *Graphical User Interface* : **GUI**
- Interface en ligne de commande ou *Command Line Interface* : **CLI**

Dans les deux cas, le programme permettant cette interaction peut être intégré à l'OS ou distinct.

- Windows : GUI et CLI complètement intégré à l'OS
- Linux : GUI et CLI indépendant du noyau, mais lié au choix de la distribution

Notions de "Shell"

- Définition : couche logicielle qui fournit l'interface utilisateur d'un système d'exploitation.

Notions de "Shell"

- Définition : couche logicielle qui fournit l'interface utilisateur d'un système d'exploitation.
- En pratique : un shell est un **interpréteur de commande** qui convertit des chaînes de caractères dans un langage codifié en une action sur le système.

Notions de "Shell"

- Définition : couche logicielle qui fournit l'interface utilisateur d'un système d'exploitation.
- En pratique : un shell est un **interpréteur de commande** qui convertit des chaînes de caractères dans un langage codifié en une action sur le système.
- Par extension, peut désigner aussi l'**émulateur de terminal** : console interactive permettant de saisir au clavier ces commandes et d'avoir à l'écran la **sortie** de la commande.

Shell : interpréteur de commandes & console

- Linux & systèmes Unix (Mac OSX) :
 - les deux sont dissociés
 - plusieurs shells historique : `sh` , `csh` , `tcsh` , `ksh` , ...
 - shell dominant aujourd'hui : **bash**

Shell : interpréteur de commandes & console

- Linux & systèmes Unix (Mac OSX) :
 - les deux sont dissociés
 - plusieurs shells historique : `sh` , `csh` , `tcsh` , `ksh` , ...
 - shell dominant aujourd'hui : **bash**
- Windows : les deux sont liés
 - Historiquement : `COMMAND.COM` sur MSDOS et Windows 3.1
 - Depuis XP/NT : programme `CMD.EXE`. Toujours existant, mais développement arrêté.
 - Depuis Windows 7 : apparition de PowerShell : approche "objet" (non traité ici)
 - 2016 : annonce par MS du portage de Bash sur Windows.

```

C:\Windows\system32\cmd.exe
kevin@localhost:~$ ls
#log.txt# boot dev init log.txt mn
acct cache etc lib lost+found op
bin data home lib64 media pr
kevin@localhost:~$ cd /mnt/
c/ d/ e/ f/
  
```

- Ce cours traite des deux en parallèle, les spécificités de chaque seront vues plus tard.

Utilisation de la console

De façon à peu près universelle, des facilités sont intégrées aux programmes "console" :

- Autocomplétion via la touche TAB
- Historique des commandes, via ↑ (fleche haut curseur)
- "Glisser-déposer" depuis un explorateur de fichier GUI
- Personnalisation du "prompt"

```
sk@PORT-LITIS-SKRAMM: ~
File Edit View Search Terminal Help
sk@PORT-LITIS-SKRAMM: /usr/share$ cd
sk@PORT-LITIS-SKRAMM: ~$
```

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Mrhope>
```

Utilisation interactive ou "script"

- Le shell peut être utilisé de façon

Utilisation interactive ou "script"

- Le shell peut être utilisé de façon
 - interactive : répétition du cycle : commande → affichage

Utilisation interactive ou "script"

- Le shell peut être utilisé de façon
 - interactive : répétition du cycle : commande → affichage
 - non-interactive : notion de "script" .

Utilisation interactive ou "script"

- Le shell peut être utilisé de façon
 - interactive : répétition du cycle : commande → affichage
 - non-interactive : notion de "script" .
- Un script est un fichier texte contenant une suite de commandes : forme de "programme exécutable"

Utilisation interactive ou "script"

- Le shell peut être utilisé de façon
 - interactive : répétition du cycle : commande → affichage
 - non-interactive : notion de "script" .
- Un script est un fichier texte contenant une suite de commandes : forme de "programme exécutable"
- Peut ensuite être exécuté :

Utilisation interactive ou "script"

- Le shell peut être utilisé de façon
 - interactive : répétition du cycle : commande → affichage
 - non-interactive : notion de "script" .
- Un script est un fichier texte contenant une suite de commandes : forme de "programme exécutable"
- Peut ensuite être exécuté :
 - en saisissant son nom dans une console ;

Utilisation interactive ou "script"

- Le shell peut être utilisé de façon
 - interactive : répétition du cycle : commande → affichage
 - non-interactive : notion de "script" .
- Un script est un fichier texte contenant une suite de commandes : forme de "programme exécutable"
- Peut ensuite être exécuté :
 - en saisissant son nom dans une console ;
 - via GUI (double clic sur le fichier).

Utilisation interactive ou "script"

- Le shell peut être utilisé de façon
 - interactive : répétition du cycle : commande → affichage
 - non-interactive : notion de "script" .
- Un script est un fichier texte contenant une suite de commandes : forme de "programme exécutable"
- Peut ensuite être exécuté :
 - en saisissant son nom dans une console ;
 - via GUI (double clic sur le fichier).
- Certaines constructions grammaticales n'ont de sens que dans un contexte de script (boucles, ...)

Utilisation interactive ou "script"

- Le shell peut être utilisé de façon
 - interactive : répétition du cycle : commande → affichage
 - non-interactive : notion de "script" .
- Un script est un fichier texte contenant une suite de commandes : forme de "programme exécutable"
- Peut ensuite être exécuté :
 - en saisissant son nom dans une console ;
 - via GUI (double clic sur le fichier).
- Certaines constructions grammaticales n'ont de sens que dans un contexte de script (boucles, ...)
- Scripts : Contexte d'utilisation

Utilisation interactive ou "script"

- Le shell peut être utilisé de façon
 - interactive : répétition du cycle : commande → affichage
 - non-interactive : notion de "script" .
- Un script est un fichier texte contenant une suite de commandes : forme de "programme exécutable"
- Peut ensuite être exécuté :
 - en saisissant son nom dans une console ;
 - via GUI (double clic sur le fichier).
- Certaines constructions grammaticales n'ont de sens que dans un contexte de script (boucles, ...)
- Scripts : Contexte d'utilisation
 - automatisation de tâches répétitives ou complexes

Utilisation interactive ou "script"

- Le shell peut être utilisé de façon
 - interactive : répétition du cycle : commande → affichage
 - non-interactive : notion de "script" .
- Un script est un fichier texte contenant une suite de commandes : forme de "programme exécutable"
- Peut ensuite être exécuté :
 - en saisissant son nom dans une console ;
 - via GUI (double clic sur le fichier).
- Certaines constructions grammaticales n'ont de sens que dans un contexte de script (boucles, ...)
- Scripts : Contexte d'utilisation
 - automatisation de tâches répétitives ou complexes
 - "en général", pas d'entrées utilisateur (mais des sorties consoles, en quantité limitée)

Utilisation interactive ou "script"

- Le shell peut être utilisé de façon
 - interactive : répétition du cycle : commande → affichage
 - non-interactive : notion de "script" .
- Un script est un fichier texte contenant une suite de commandes : forme de "programme exécutable"
- Peut ensuite être exécuté :
 - en saisissant son nom dans une console ;
 - via GUI (double clic sur le fichier).
- Certaines constructions grammaticales n'ont de sens que dans un contexte de script (boucles, ...)
- Scripts : Contexte d'utilisation
 - automatisation de tâches répétitives ou complexes
 - "en général", pas d'entrées utilisateur (mais des sorties consoles, en quantité limitée)
- Windows : certaines syntaxes ne sont pas identiques en interactif et en script...
(commande `for`)

Différences Linux / Windows

Windows

- pensé dès l'origine pour une utilisation en GUI : le pilotage du système via une CLI n'a pas été prioritaire ;
- encore beaucoup de manipulations doivent se faire en GUI.

Différences Linux / Windows

Windows

- pensé dès l'origine pour une utilisation en GUI : le pilotage du système via une CLI n'a pas été prioritaire ;
- encore beaucoup de manipulations doivent se faire en GUI.

Linux

- inspiration : systèmes UNIX ;
- pensé dès le départ comme pilotable via CLI ;
- GUI complètement dissociée du noyau ;
- beaucoup de manipulations ne peuvent se faire que via CLI.

Commande internes et externes

- Un shell peut utiliser des commandes **internes** ou **externes**.
 - commande interne : exécutée directement par le shell lui-même
 - commande externe : programme distinct, lancé par le shell lorsque l'utilisateur le demande.
Doit correspondre à un fichier contenant un programme exécutable, et qui soit **accessible** depuis le dossier courant.
- Pour l'OS, pas de différence entre une commande externe et un programme : les deux sont vus comme des **exécutables**.
- Linux :
 - Liste des commandes internes : `help`
 - Commande interne / externe ? : `type commande`
- Windows : les commandes externes sont des programmes stockés dans `c:\Windows\system32`

Comparaison Bash/CMD

- Programme "terminal" (console)
 - Windows : rudimentaire et peu évolué. Versions alternatives disponibles.
 - Linux : grand choix de programmes.
- Scripts Windows : beaucoup plus rudimentaire
 - moins de commandes internes
 - pas de structure de contrôle évoluées (fonctions, boucle "while")
 - support et documentation de MS aléatoire et variant suivant les versions de Windows, documentation éclatée.
- bash
 - Syntaxe parfois obscure
- Windows : insensible à la casse
DIR identique à dir

Sommaire

- 1 Introduction
- 2 Commandes de base
- 3 Enchaînement et redirection
- 4 Variables
- 5 Scripts : structures de contrôle
 - Windows
 - Bash

Syntaxe générale

- Syntaxe générale : `commande [option] [argument]` (sur **une** ligne)
 - entre crochets : optionnel. Certaines commandes pourront avoir un nombre d'arguments minimal.
 - Windows : les options sont données avec /
Exemple : `dir /w`
 - Bash : les options sont données avec - ("option courte") ou -- ("option longue")
Exemple : `ls -a` ou `ls --all`

Syntaxe générale

- Syntaxe générale : `commande [option] [argument]` (sur **une** ligne)
 - entre crochets : optionnel. Certaines commandes pourront avoir un nombre d'arguments minimal.
 - Windows : les options sont données avec /
Exemple : `dir /w`
 - Bash : les options sont données avec - ("option courte") ou -- ("option longue")
Exemple : `ls -a` ou `ls --all`
- Le caractère SPC (ASCII :32, \$20) est utilisé comme séparateur
⇒ Si nom de fichier avec des espaces, il sera nécessaire de les encadrer par des "

Exemples :

 - `commande mon super fichier.txt` : illégal
 - `commande "mon super fichier.txt"` : ok

Syntaxe générale

- Syntaxe générale : `commande [option] [argument]` (sur **une** ligne)
 - entre crochets : optionnel. Certaines commandes pourront avoir un nombre d'arguments minimal.
 - Windows : les options sont données avec /
Exemple : `dir /w`
 - Bash : les options sont données avec - ("option courte") ou -- ("option longue")
Exemple : `ls -a` ou `ls --all`
- Le caractère SPC (ASCII :32, \$20) est utilisé comme séparateur
⇒ Si nom de fichier avec des espaces, il sera nécessaire de les encadrer par des "
Exemples :
 - `commande mon super fichier.txt` : illégal
 - `commande "mon super fichier.txt"` : ok
- On peut interrompre une commande en cours (ou annuler l'édition en cours) avec `CTRL-C`.

Spécification de fichier

- Beaucoup de commandes acceptent en argument une **spécification de fichier**
- Une spécification de fichier est un **masque**, sur lequel des noms de fichiers peuvent "coller", via des **caractères génériques** :
 - * : un ou plusieurs caractère(s)
 - ? : un seul caractère
- Exemples (Windows, pour Linux remplacer DIR par ls)
 - `DIR a*.txt` : affiche la liste de tous les fichiers commençant par "a" et ayant l'extension txt
 - `DIR ??? .mp3` : tous les fichiers mp3 de 3 lettres
 - `DIR IUT*.pdf` : tous les fichiers pdf avec IUT comme 1^{res} lettres

Navigation et manipulation fichiers

	Linux	Windows
Copie de fichier	<code>cp source dest</code>	<code>copy source dest</code>
Copie de dossiers	<code>cp source dest</code>	<code>xcopy source dest</code>
Renommage de fichier	<code>mv ancien nouveau</code> ("move")	<code>ren ancien nouveau</code> ("rename")
Déplacement de fichier	<code>mv ancien nouveau</code>	<code>move ancien nouveau</code>
Effacement	<code>rm fich</code>	<code>del fich</code> ("delete")
Afficher le contenu d'un fichier	<code>cat fich</code>	<code>type fich</code>

- Ces commandes permettent l'utilisation de caractères génériques pour la spécification de fichier.

`copy *.txt dest` : va copier tous les fichiers d'extension txt du dossier courant vers le dossier dest.

Répertoires

	Linux	Windows
Affichage contenu rép.	<code>ls [dossier]</code>	<code>dir [dossier]</code>
Changement répertoire actif	<code>cd un/chemin</code>	<code>cd un\chemin</code>
déplacement rép. actif vers le rep. parent	<code>cd ..</code>	<code>cd ..</code>
Création répertoire	<code>mkdir rep/sousrep</code>	<code>md rep\sousrep</code>
Effacement répertoire	<code>rmdir rep</code>	<code>rmdir rep</code>
Affichage du rép. courant	<code>pwd¹</code>	<code>cd</code>

1. `pwd` : *Print Working Directory*

Autre commandes

	Linux	Windows
Aide sur commande	help commande ou man commande	help commande ou commande /?
terminer une session/un script	exit	exit
Affichage de texte	echo	echo

- Commentaires dans un script :

- Linux : toute ligne commençant par `#`
- Windows : toute ligne commençant par `::` ou `REM`

Sommaire

- 1 Introduction
- 2 Commandes de base
- 3 Enchaînement et redirection
- 4 Variables
- 5 Scripts : structures de contrôle
 - Windows
 - Bash

E/S standard

- Par défaut, chaque processus se voit attribué des "flots" d'E/S
 - un flot d'entrée, associée aux entrées clavier : `stdin`
 - un flot de sortie standard (1), associé à la console : `stdout`
 - un flot de sortie d'erreur (2), associé à la console : `stderr`
- Ces différents flots peuvent être **redirigés** ponctuellement vers un autre périphérique ou dans un fichier

E/S standard

- Par défaut, chaque processus se voit attribué des "flots" d'E/S
 - un flot d'entrée, associée aux entrées clavier : `stdin`
 - un flot de sortie standard (1), associé à la console : `stdout`
 - un flot de sortie d'erreur (2), associé à la console : `stderr`
- Ces différents flots peuvent être **redirigés** ponctuellement vers un autre périphérique ou dans un fichier
- Exemple 1 : redirection vers l'imprimante de `stdout` (1)
 - Linux : `commande > /dev/lp` ou `commande 1> /dev/lp`
 - Windows : `commande > LPT1` ou `commande 1> LPT1`

E/S standard

- Par défaut, chaque processus se voit attribué des "flots" d'E/S
 - un flot d'entrée, associée aux entrées clavier : `stdin`
 - un flot de sortie standard (1), associé à la console : `stdout`
 - un flot de sortie d'erreur (2), associé à la console : `stderr`
 - Ces différents flots peuvent être **redirigés** ponctuellement vers un autre périphérique ou dans un fichier
 - Exemple 1 : redirection vers l'imprimante de `stdout` (1)
 - Linux : `commande > /dev/lp` ou `commande 1> /dev/lp`
 - Windows : `commande > LPT1` ou `commande 1> LPT1`
 - Exemple 2 : redirection de la sortie d'erreur vers le néant
 - Windows : `commande 2> NUL` (NUL : périphérique "spécial")
 - Linux : périphérique virtuel dédié : `commande 2> /dev/null`
- Intérêt : suppression de certains affichages

Redirection vers un fichier

- Très souvent, on redirige vers un fichier
- Redirection en sortie :
 - `commande > fich` : le fichier `fich` est créé (effacé s'il existe déjà)
 - `commande >> fich` : la sortie de `commande` est ajoutée à la fin de `fich`

Enchaînement de commandes

- Exécuter deux commandes à la suite :

- Linux : `prog1; prog2`

- Windows : `prog1 & prog2`

- Exécuter deux commandes en parallèle (Linux seulement) :

```
prog1 & prog2
```

- Exécuter une 2^e commande uniquement si la 1^{re} n'échoue pas (code de retour = 0) :

```
prog1 && prog2
```

- Exécuter une 2^e commande uniquement si la 1^{re} échoue (code de retour != 0) :

```
prog1 || prog2
```

- Utiliser comme entrée d'une 2^e commande la sortie d'une 1^{re} :

```
prog1 | prog2
```

(ceci s'appelle un "*pipe*")

Sommaire

- 1 Introduction
- 2 Commandes de base
- 3 Enchaînement et redirection
- 4 Variables**
- 5 Scripts : structures de contrôle
 - Windows
 - Bash

Variables

- Comme n'importe quel langage de programmation, on peut définir des **variables**
- Quel que soit le shell, les variables sont toujours de type "texte" (chaînes de caractères)
- Mais possibilité de faire de l'arithmétique élémentaire.
- On distingue :
 - variables locales : connues uniquement dans le script.
 - variables d'environnement : communes à tous l'environnement.
Certaines sont prédéfinies par l'OS, mais possible par l'utilisateur.

Variables

- Comme n'importe quel langage de programmation, on peut définir des **variables**
- Quel que soit le shell, les variables sont toujours de type "texte" (chaînes de caractères)
- Mais possibilité de faire de l'arithmétique élémentaire.
- On distingue :
 - variables locales : connues uniquement dans le script.
 - variables d'environnement : communes à tous l'environnement.
Certaines sont prédéfinies par l'OS, mais possible par l'utilisateur.

Attention

Les variables créées dans un shell ne sont connues **que** dans celui-ci.

- En cas d'appel d'un sous-shell, les variables ne sont pas transmises.
- la création de variables d'environnement passe par des commandes spécifiques.

Manipulation de variables

	Linux	Windows
Donner une valeur à une variable	<code>var=un</code>	<code>SET mavar=un</code>
Utiliser une variable	préfixer avec '\$' Ex : <code>echo "var=\$var"</code>	Encadrer avec '%' Ex : <code>ECHO mavar=%mavar%</code>
Saisie au clavier	<code>read var</code>	<code>SET /P var=texte_invite</code>

- Test de variables : Linux/Bash

- Forme générale :

```
if [ item1=item2 ]; then <command> ; fi
```

Exemple : `if [$var=ABC]; then echo "Oui !"; fi`

- test d'inégalité : remplacer `=` par `!=`

Test de variables : Windows

- forme générale :

```
IF [/I] [NOT] item1==item2 command
```

Exemple : `if %mavar%==coucou echo ok`

- Si espaces ; alors il faut "quoter" :

```
if "%mavar%" == "Ah Que Coucou" echo ok
```

- Pour dérouter l'exécution à un label donné, on utilisera la commande goto :

```
if %var%==ABC goto labas
```

- Option `/I` pour ignorer la casse
- Test d'inégalité : ajouter `NOT`

Variables d'environnement

- Chaque OS définit ses variables d'environnement (VE).
- Permet de connaître depuis un script des informations sur le système.

- Liste complète :

Linux : `env`

Windows : `set`

- Facilités :

- Windows : `set ABC` : renvoie toutes les VE commençant par "ABC"

- Linux : `env | grep ABC` : renvoie toutes les VE contenant la chaîne "ABC"

- Exemples :

	Linux	Windows
Nom de l'utilisateur courant	USER	USERNAME
Localisation des applications	PATH	PATH
Dossier pour fichier temporaires	(néant) ²	TEMP
Date	(néant)	DATE
"Home" de l'utilisateur	HOME	HOMEPATH

Variable PATH

- Contient la liste des chemins absolus où l'OS va chercher la commande demandée.
- Avec une saisie de type `aaa bbb ccc`, l'OS va rechercher la commande `aaa` dans l'ordre suivant :
 - 1 est-ce une commande interne ?
 - 2 est-ce un fichier exécutable dans le dossier courant ? (Windows seulement)
 - 3 est-ce un programme qu'on trouve dans les chemins donnés dans la variable PATH, en faisant la recherche dans l'ordre qui est donné ?
- A défaut, il faudra donner le chemin (absolu ou relatif), par exemple :
 - Windows : `"c:\program files\superappli\aaa" bbb ccc`
 - Linux :
 - `/home/mon_id/dev/mes_applis/aaa bbb ccc`
 - `../../ici/labas/aaa bbb ccc`
- Linux : si le programme à exécuter est dans le dossier courant, alors il **faut** le spécifier explicitement en tapant `./commande`

Recherche de commande : différences Windows/Linux

Windows

- Un fichier exécutable est identifié comme tel par son extension.
- Les principales : BAT, CMD ou EXE, mais aussi beaucoup d'autres (voir la VE PATHEXT)
- On peut l'appeler uniquement par son nom, l'OS va chercher s'il trouve une correspondance, dans **cet** ordre.

Attention : un dossier contient deux fichiers, `bidule.bat` et `bidule.exe`
Si on tape `bidule`, lequel sera exécuté ?

Recherche de commande : différences Windows/Linux

Windows

- Un fichier exécutable est identifié comme tel par son extension.
- Les principales : BAT, CMD ou EXE, mais aussi beaucoup d'autres (voir la VE PATHEXT)
- On peut l'appeler uniquement par son nom, l'OS va chercher s'il trouve une correspondance, dans **cet** ordre.
Attention : un dossier contient deux fichiers, `bidule.bat` et `bidule.exe`
Si on tape `bidule`, lequel sera exécuté ?

Linux

- Il faut donner le nom entier du programme, avec l'extension éventuelle.
- Exemple : soit un script qui s'appelle `mon_script.sh`

Exécution :

`./mon_script` : erreur

`./mon_script.sh` : OK

Arguments passés au script

- Lors du lancement, on peut passer des arguments au script :

```
mon_script.sh toto tata
```

- Ces valeurs sont placées automatiquement dans des variables spéciales, appelées **paramètres positionnels**.

	Linux	Windows	Exemple
Nombre d'arguments	<code> \$# </code>	(néant)	2
Tous les arguments	<code> \$* </code>	<code> %* </code>	toto tata
Chemin et nom du script	<code> \$0 </code>	<code> %0 </code>	mon_script.sh
Arguments positionnels	<code> \$1 </code>	<code> %1 </code>	toto
	<code> \$2 </code>	<code> %2 </code>	tata
	<code> \$3 </code>	<code> %3 </code>	
	
Code de sortie de la dernière commande	<code> \$? </code>	<code> %ERRORLEVEL% </code>	

Convention universelle : un programme / une commande renvoie `0` en cas de succès, toutes les autres valeurs indiquant une erreur ou un échec.

Arguments passés au script : exemple

Exemple : soit le script suivant

(Version Bash)

nom : test.sh

```
#!/bin/bash
echo arg1=$1 arg2=$2
```

(Version Windows)

nom : test.bat

```
@echo off
echo arg1=%1 arg2=%2
```

L'exécution de la commande :

```
$ ./test.sh toto tata
```

```
$ test toto tata
```

va produire l'affichage :

```
arg1=toto arg2=tata
```

Arithmétique entière

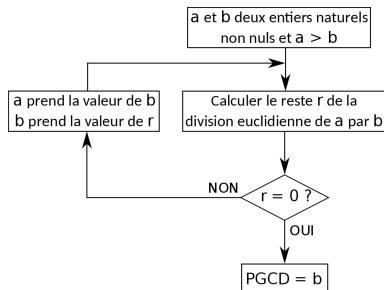
- Les Shell permettent de faire de l'arithmétique entière
- Windows :
 - option /A à la commande SET
 - De multiples opérateurs, arithmétiques et logiques (ET, OU, décalages, ...)
 - Exemple : `SET /A compt=%compt%+1`
 - Référence : ss64.com, ou `SET /?`
- Linux/Bash :
 - La notation `$(($nom_variable))` permet de traiter une variable comme un entier
 - Exemple :


```
var1=4
var2=$(($var1*2))
echo $var2 /* affiche 8 */
```
- Tests :
 - pour tester l'égalité : identique à de la comparaison de chaîne
 - pour < et >, il existe des syntaxes spécifiques à chaque Shell

Sommaire

- 1 Introduction
- 2 Commandes de base
- 3 Enchaînement et redirection
- 4 Variables
- 5 Scripts : structures de contrôle**
 - Windows
 - Bash

Structures de contrôle



- Windows/CMD : utilisation de branchements inconditionnels à des **labels**
- Linux/Bash : beaucoup plus évolué : boucles while, for, ...
- Avec les deux, on peut créer des **fonctions**, permettant l'appel à un traitement et le retour automatique.

Sommaire

- 1 Introduction
- 2 Commandes de base
- 3 Enchaînement et redirection
- 4 Variables
- 5 **Scripts : structures de contrôle**
 - Windows
 - Bash

Windows : branchements et labels

- Utilisation de branchements inconditionnels à des **labels**

Ex : boucle infinie :

```
:ici  
goto labas
```

Cette ligne nest jamais lue par le shell!

```
:labas  
echo coucou 1  
goto ici
```

Windows : fonctions

- Appel d'une fonction : commande `call`

```

set var=1
call :mafonction
echo "var=%var%"           # affiche 2
call :mafonction
echo "var=%var%"           # affiche 3
goto :eof

:mafonction
echo "coucou"
set /A var= %var%+1
goto :eof

```

Le label spécial `:eof` indique un "retour à l'appelant", ou sortie du script si pas de commande `call` correspondante.

Windows : fonctions et arguments

- On peut passer des arguments à une fonction

L'exécution de ceci :

```
call :mafonction aaa
call :mafonction bbb
goto :eof

:mafonction
echo "argument=%1"
goto :eof
```

affichera :

```
argument=aaa
argument=bbb
```

Windows : branchements conditionnels

- On utilise la commande goto dans un test

```
if "%var1%" == "1" goto ici  
goto labas
```

... ligne jamais interpretee...

```
:ici  
echo "la variable vaut 1"  
goto :eof
```

```
:labas  
echo "differente de 1"  
goto :eof
```

Windows : branchements ?

- L'approche du contrôle de flux par branchements amène :

- un avantage : l'implémentation à partir d'un algorithme est très facile : on "branche" où on veut.
- un inconvénient : le programme devient très rapidement **dé-structuré** : très difficile de s'y retrouver, mise à jour et maintenance compliquée...
⇒ On parle de "**code spaghetti**" :

```
public static void Main()
{
```

```
    labelA; ←
    if ( ... )
        goto labelC;
    if ( ... )
        goto labelB;
```

```
    labelD; ←
    if ( ... )
        goto labelE;
    labelC ←
```

```
    labelE; ←
```

```
    if ( ... )
        goto labelA;
    if ( ... )
        goto labelD;
    labelB; ←
```

```
}
```



Sommaire

- 1 Introduction
- 2 Commandes de base
- 3 Enchaînement et redirection
- 4 Variables
- 5 Scripts : structures de contrôle**
 - Windows
 - **Bash**

Bash : langage structuré

- Bash est un langage **structuré** : pas de branchements, tout doit être inclus dans un "bloc" de commandes.
- Les tests auront la forme suivante :

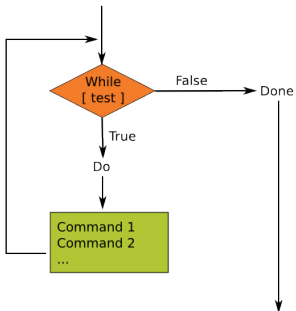
```
if [ <some test> ]
then
    <commands>
fi
```

```
if [ <some test> ]
then
    <commands>
else
    <other commands>
fi
```

- Exemple : test de chaînes de caractères
(voir p. 24)
Attention aux espaces (obligatoires)

```
if [ $name = "paul" ]
then
    echo "Paul !"
else
    echo "pas Paul!"
```


Bash : boucles *while*



```
while [ <some test> ]  
do  
    <commands>  
done
```

Bash : fonctions

- Ecriture et appel d'une fonction :

```
function mafonction
{
    echo "Bonjour $1"
}

mafonction tata
mafonction toto
```

Attention : les fonctions doivent être définies **avant** leur utilisation dans le script.