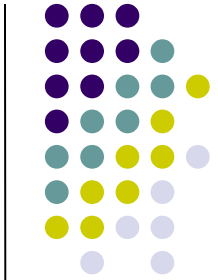
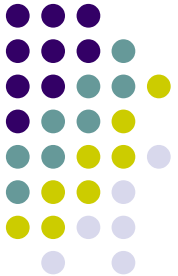


Le shell
Principales commandes

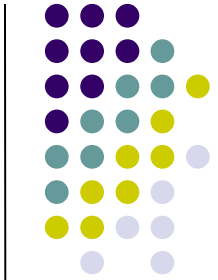
Xavier HILAIRE
x.hilaire@esiee.fr

Qu’est-ce que le shell ?



- Le shell est un programme exécutable, dont la principale fonction est de permettre à l’utilisateur d’interagir avec le système via un terminal.
- Il est aussi appelé *interpréteur de commandes*.
- Deux modes d’utilisation :
  - Interactif : l’utilisateur saisit et exécute ses lignes de commandes une par une dans un terminal ;
  - Non interactif : le shell lit un ensemble de commandes à partir d’un fichier appelé *script*.
- Il existe aujourd’hui plus d’une trentaine de shells différents, mais deux grandes familles dominent :
  - Les csh : shells orientés administration, avec une syntaxe inspirée du langage C

Qu’est-ce que le shell ?

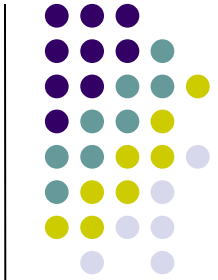


- La famille des sh (à l’origine: ash), bsh (Bourne shell), bash (Bourne again shell) : shells orientés utilisateur, majoritaires aujourd’hui. La plupart des scripts shell sont écrits en sh, ou au moins compatibles sh.

- Le shell UNIX standard est sh.

- Mais Bash supplante de plus en plus souvent sh (c’est le cas sur Linux) :
    - Il consiste en un mélange de sh, de quelques fonctions du csh, et d’autres du Korn shell (ksh)
    - Il est 100% compatible sh.

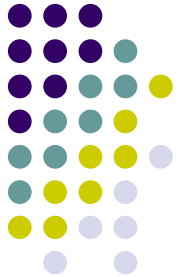
Principe d’exécution



Pour simplifier, le shell adopte toujours le même plan d’exécution :

1) Lit une **ligne de commande** soit à partir du terminal, soit à partir d’un fichier script
2) Effectue une première analyse qui détermine quels en sont les **opérateurs**, et quels en sont les **mots**
3) Décompose la ligne de commande en **commandes simples**, en appliquant un jeu de priorités fixé par les opérateurs identifiés
4) Pour chaque commande simple :
   - Réécrit les mots de cette commande lorsque c’est possible
   - Lance la commande en question

Grammaire



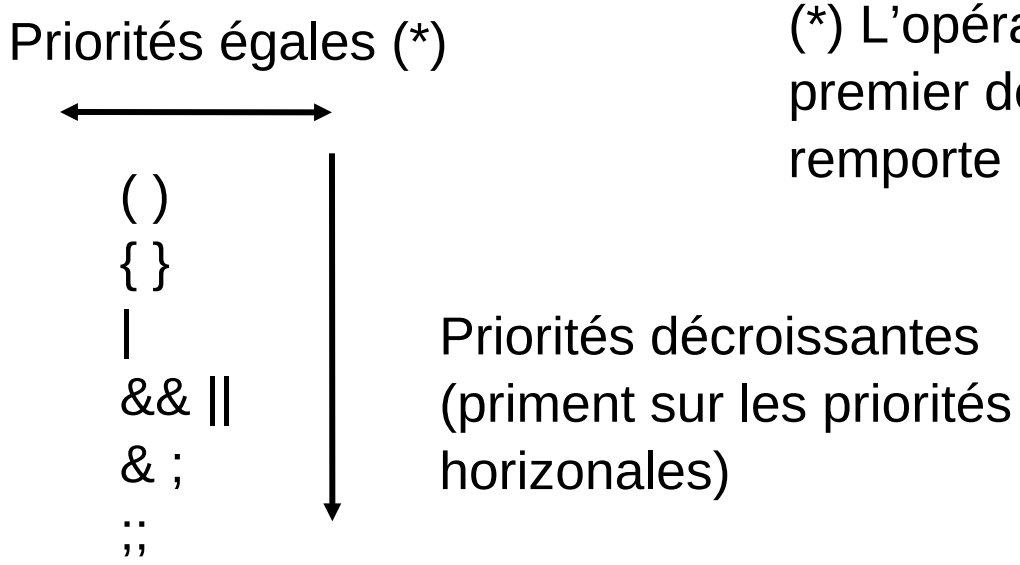
- Les mots sont soit des suites ininterrompues de caractères, soit des chaînes de caractères entourées de quotes simples ' ' ou doubles " "
- Les quotes simples interdisent de modifier le contenu qu'elles délimitent
- Les quotes doubles l'autorisent, mais seulement pour le métacaractère $
- Deux types d'opérateurs :
  - ***Contrôle*** : ils servent à séparer deux commandes ou deux listes. Ce sont: `&` `&&` `( )` `{}` `;` `;;` `|` `||`
  - ***Redirection*** : ils servent à rediriger les entrées/sorties et portent sur une seule commande. Ce sont: `<` `>` `>|` `<<` `>>` `<&` `>&` `<<-` `<>`

Grammaire

Une ligne de commande est :

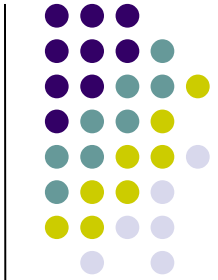
- **simple** si elle ne comporte aucun opérateur de contrôle ;
- **composée** dans le cas contraire.

Priorités des opérateurs de contrôle :



(*) L'opérateur rencontré le premier de gauche à droite remporte la priorité

Grammaire



Exemple : analyse de la ligne de commande

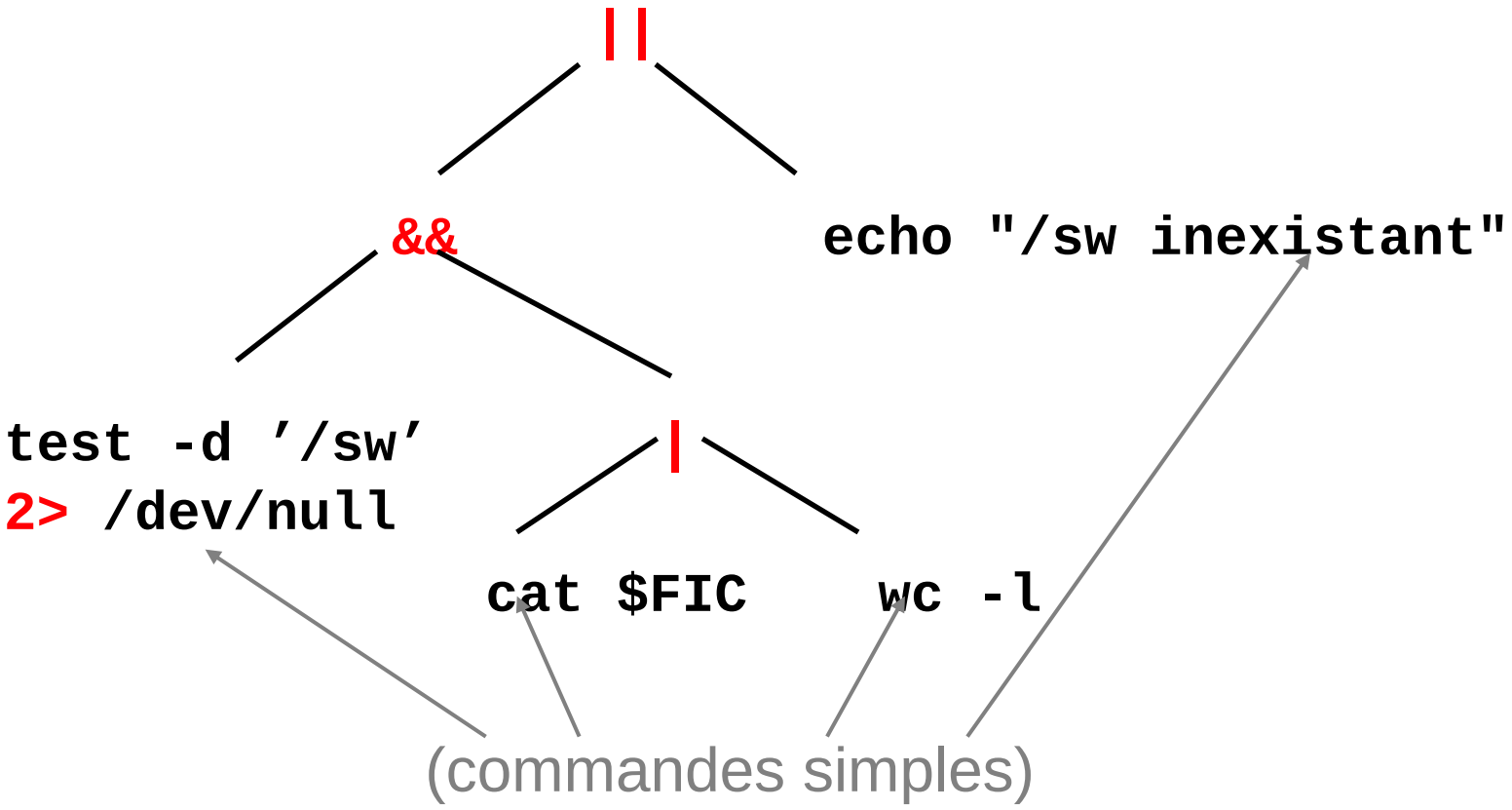
```
test -d   ’/sw’  2>/dev/null && cat $FIC |
wc -l || echo  "/sw inexistant"
```

Etapes 2 et 3 : distinguer les mots des opérateurs.
On obtient :

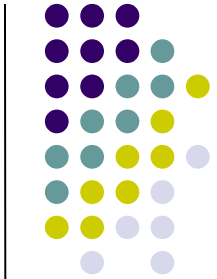
```
test -d ’/sw’ 2> /dev/null && cat $FIC | wc
-l || echo  "/sw inexistant"
```

Grammaire

Etape 4 : résultat de l'analyse



Commandes simples

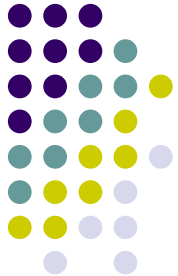


Une commande simple a la forme suivante (les [ ] marquent le caractère optionnel):

`[env] [com] [args…] [redir]`

- **env** consiste en aucune, une ou plusieurs affectations de *variables d'environnement* portant sur la commande qui suit (et seulement elle)
- **com** est la commande elle-même. Il s'agit soit d'un nom de commande interne, soit d'un nom de fichier exécutable accessible par la *variable d'environnement* PATH
- **args** sont les arguments optionnels de la commande
- **redir** consiste en aucune, une, ou plusieurs redirections de fichiers

Commandes simples

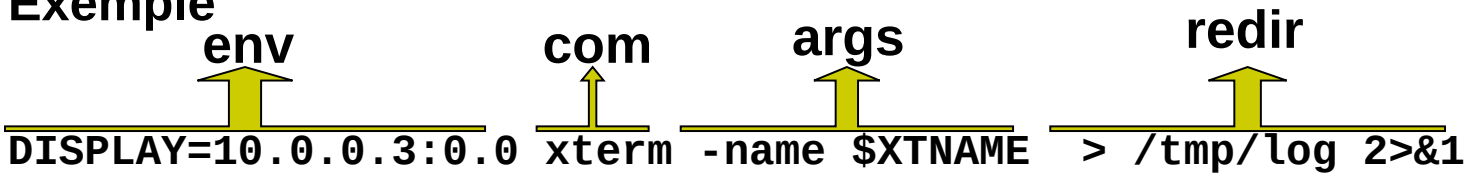


Evaluation d'une commande simple : ce qui se passe (dans l'ordre)

1. Chaque mot de la commande est **évalué**, c'est à dire soit laissé tel quel s'il est interdit de l'interpréter (quotes simples), soit réécrit sur la ligne de commande dans le cas contraire (quotes doubles ou pas de quotes)
2. Les redirections **[redir]** sont appliquées, puis retirées de la ligne de commande
3. Le bloc **[env]** est évalué, appliqué, puis retiré de la ligne de commande.
4. La commande **[com]** est lancée, si elle existe (*oui, il est possible d'avoir un bloc [com] vide!*)

Commandes simples

Exemple



1. Variables d'environnement : une seule ici , `DISPLAY=10.0.0.3:0.0` qui est exécutée, puis retirée. Il reste :

`xterm -name $XTNAME > /tmp/log 2>&1`

2. Résolution : une seule substitution à faire ici, `$XTNAME` qui est une variable. En supposant que XTNAME=myterm, il reste :

`xterm -name myterm > /tmp/log 2>&1`

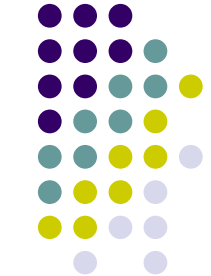
3. Redirections : `> /tmp/log` indique que la sortie standard doit être redirigée vers le fichier `/tmp/log`, `2>&1` que la sortie d'erreur standard et la sortie standard doivent être unifiées. Le shell créé /tmp/log, fait la redirection, puis retire la fin de ligne. Il reste :

`xterm -name myterm`

4. Le programme xterm est lancé, avec deux arguments seulement :

`xterm -name myterm`

Les variables



- Le shell connaît deux types de variables : **ordinaires** et **d'environnement**.

- Une **variable ordinaire** n'est connue que par le shell : aucun des programmes que le shell lancera ne pourra la consulter/modifier.

- A l'inverse, une **variable d'environnement** est une variable ordinaire dont les processus lancés par le shell recoivent une copie.

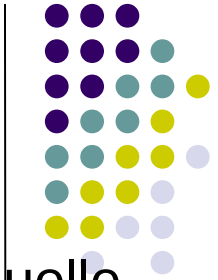
- On peut affecter une valeur à une variable avec l'opérateur '=' (attention: **pas d'espace !**)

```
$ mavariable=3
$ echo $mavariable
3
```

- Pour faire d'une variable ordinaire une variable d'environnement, on utilise la commande **export** :

```
$ sh -c 'echo $mavariable' # lance la com. dans un sous-shell
$ export mavariable
$ sh -c 'echo $mavariable'
3
```

Evaluation



- C’est une technique de réécriture très puissante durant laquelle les variables, les noms de fichiers, et certains caractères sont substitués dans chaque mot qui les contient soit par leur propre valeur, soit par autre chose.
- Cela n’est toutefois pas toujours possible :
  - Si le mot concerné est entouré par des quotes simples : ' ' le shell n’est pas autorisé à interpréter quoi que ce soit, il laissera le mot tel quel, mais retirera les quotes délimitantes
  - Si le mot est entouré par des quotes doubles " " : le shell est autorisé à évaluer l’opérateur $, à substituer le résultat sur la ligne de commande, et à retirer les quotes délimitantes
  - Si le mot n’est entouré par rien, le shell est autorisé à évaluer l’opérateur $ et les noms de fichiers, et à substituer le résultat sur la ligne de commande

Evaluation

Exercice 1

- Nous allons vérifier ce qui vient d'être dit grâce à une première manipulation.
- Ouvrir un terminal, puis taper la commande suivante dedans :

```
curl -o args1.c
   https://perso.esiee.fr/~hilairex/3R-IN3/args1.c
```

- Examiner le programme args1.c : il se contente d'afficher les arguments qu'il a reçus
- Compiler ce programme :

```
gcc -o args1 args1.c
```

- Exécuter et expliquer le résultat des commandes suivantes :
- **`./args1`**
- **`echo $PWD`**
- **`$PWD/args1`**

Evaluation

Exercice 1 (suite)

- `./args1 un    deux       trois  $PWD`
- `./args1 "un    deux       trois" $PWD`
- `./args1 'un    deux       trois' $PWD`
- `salut="Bonjour  tout le  monde !!"`
- `./args1 $salut`
- `./args1 "$salut"`
- `./args1 '$salut'`
- `./args1 "/???"`
- `./args1 '/???'`
- `./args1 /???`

Evaluation

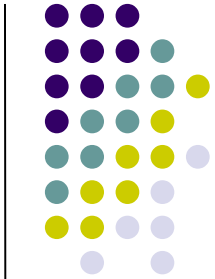
Exercice 2

- Nous allons à présent vérifier l'effet des variables d'environnement sur les programmes
- Ouvrir un terminal, et taper la commande suivante dedans :

```
curl -o args2 https://perso.esiee.fr/~hilairex/3R-IN3/args2.c
```

- Ce deuxième programme fait la même chose que le premier, mais affiche en plus les variables d'environnement qu'il a reçues dans son envp
- Examiner et expliquer l'effet des commandes suivantes (dans l'ordre indiqué) :
  - `./args2 un deux`
  - `ZZZZZZ=essai ./args2 un deux`
  - `ZZZZZZ=essai`
  - `./args2 un deux`
  - `export ZZZZZZ`
  - `./args2 un deux`
  - `man ls`
  - `LANG=C man ls`

Evaluation des variables



Evaluation des variables

- Il est possible de faire remplacer une variable par autre chose que son contenu.
- Le tableau ci-dessous donne un aperçu de quelques substitutions possibles.

Forme	var définie	var indéfinie
${var} ou $var	Substitue var	Substitue la chaîne vide
${var:-mot}	Substitue var	Substitue mot
${var:+mot}	Substitue mot	Substitue la chaîne vide
${var:?[mot]}	Substitue var	Erreur + substitue mot si spécifié
${var:=mot}	Substitue var	Affecte mot à var; substitue mot

Evaluation des variables

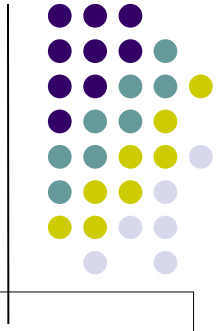
Forme	Substitution
${#var}	Longueur de la chaîne var
${var%motif}	var privée de la plus courte occurrence droite de motif
${var%%motif}	var privée de la plus longue occurrence droite de motif
${var#motif}	var privée de la plus courte occurrence gauche de motif
${var##motif}	var privée de la plus longue occurrence gauche de motif

```
$ var=aaabccc
$ echo $var
aaabccc
$ echo ${va:-mot}
mot
$ echo ${var:-mot}
aaabccc
$ echo ${var:+mot}
mot
$ echo ${va:?test}
```

```
va: test
$ echo ${va:?}
va: parameter null or
not set
$ echo ${va:=mot}
mot
$ echo $va
mot
$ echo ${#var}
7
```

```
$ echo ${var%e}
aaabccc
$ echo ${var%c}
aaabcc
$ echo ${var%%c}
aaabcc
$ echo ${var%c*}
aaabcc
$ echo ${var%%c*}
aaab
```

Evaluation des variables



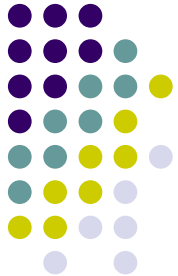
Variables spéciales (utilisables seulement dans un script) :

Nom	Substitué par
*	L'ensemble des paramètres concaténés en une seule chaîne
@	L'ensemble des paramètres en autant de chaînes que nécessaire
#	Nombre de paramètres passés
-	Option courante
0,1,2…	Chaque paramètre, y compris le nom de la commande (0)

Variables spéciales toujours utilisables :

Nom	Substitué par
$	Numéro de processus (PID) du shell courant
!	PID de la dernière commande d'arrière-plan terminée
?	Code de retour du dernier processus terminé (exit de la dernière commande)

Evaluation arithmétique



Evaluation arithmétique entière

Forme générale: $((expression)) où expression est une expression arithmétique entière

```
$ echo $((3+1))
4
$ echo $((3+2*6))
15
$ echo $((3-(1+1)*6))
-9
$ i=2
$ i=$((i+1))
$ echo $i
3
$ echo $((3+(1+1.5)*6))
sh: 3+(1+1.5)*6: missing `)' (error token is ".5)*6")
```

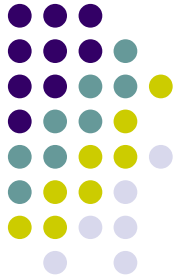
Evaluation des noms de fichiers

Evaluation des noms de fichiers

- Lorsqu'il rencontre les métacaractères '*', '?', '~', '[', et ']' dans un mot non protégé (pas de quotes), le shell cherche à les remplacer pour les faire concorder avec des noms de fichiers existants sur disque
- Le shell substitue alors autant de chaînes qu'il est possible de former de fichiers concordants. S'il n'en trouve pas, le mot est laissé tel quel.
- Le tilde '~' seul, suivi de / ou d'un nom de login est toujours substitué, et en premier.

?	Remplace un caractère exactement
*	Remplace un nombre quelconque de caractères (y compris aucun)
[p]	Remplace un seul caractère parmis ceux indiqués dans p (même syntaxe que pour les expressions régulières)
~nom	Substitué par la variable $HOME de l'utilisateur *nom*

Evaluation des noms de fichiers



```
$ echo ~
/Users/xavier
$ echo $HOME
/Users/xavier
$ ls
cap1.tiff       cap2.tiff       test
$ echo cap?.tiff capi*.tiff
cap1.tiff cap2.tiff capi*.tiff
$ echo *es*
test
$ echo /???
/bin /dev /etc /lib /tmp /usr /var
$ echo /???/??
/bin/cp /bin/dd /bin/df /bin/ed /bin/ln /bin/ls /bin/mv /bin/ps
   /bin/rm /bin/sh /dev/fd /etc/rc /var/at /var/db /var/vm
   /var/yp
$ echo /???/[a-c]??
/bin/cat /bin/csh /usr/bin
```

Evaluation

En résumé :

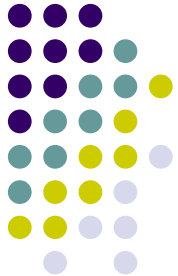
- Tous les mots non délimités par des " et des ' sont évalués : variables, arithmétique, et de noms de fichiers
- Les variables contenues dans les mots encadrés par des " sont évaluées, mais pas les noms de fichiers
- Les chaînes de caractères encadrées par des ' ne sont jamais évaluées
- Un métacaractère précédé de '\' se comporte comme un caractère ordinaire
- Les chaînes encadrées par des " ou par des ' sont toujours considérées comme un seul mot

Exemples

```
$ i=1
$ ls
cap1.tiff       cap2.tiff
   test
$ echo  $i $((i+1)) cap*
1 2 cap1.tiff cap2.tiff
```

```
$ echo  "$i" "$((i+1))" "cap*"
1 2 cap*
$ echo  '$i' '$((i+1))' 'cap*'
$i $((i+1)) cap*
$ echo  \$i \$\(\(i+1\)\) cap\*
$i $((i+1)) cap*
```

Evaluation de commande



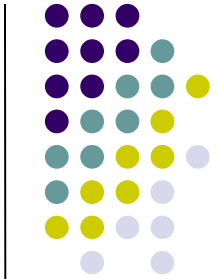
L'évaluation de commande demande à ce que le contenu produit par `commande` sur sa sortie standard soit substitué sur la ligne de commande courante. Les délimiteurs autres que l'espace (tabulations, retours chariots) sont remplacés par des espaces.

Forme générale: `$(commande)` ou `` `commande` ``

Exemple

```
$ ls
cap1.tiff       cap2.tiff       test
$ ls | wc -l
       3
$ echo Il y a $(ls | wc -l) fichiers
Il y a 3 fichiers
$
```

Les redirections



A son lancement, tout processus dispose de trois canaux ouverts par le système :

- l'**entrée standard** = stdin, canal **0**
- la **sortie standard** = stdout, canal **1**
- la **sortie d'erreur standard** = stderr, canal **2**

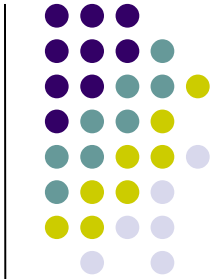
A ces canaux sont associés des fichiers par défaut :

- les caractères tapés par l'utilisateur dans le terminal d'exécution alimentent stdin
- le terminal est alimenté par stdout et stderr

Les redirections permettent d'associer d'autres fichiers aux canaux que ceux par défaut.

```
$ ls
cap1.tiff       cap2.tiff       test
$ ls > /tmp/toto # redirige stdout vers /tmp/toto
$ ls -l /tmp/toto
-rw-r--r--   1 xavier  wheel  25 Feb 25 23:24 /tmp/toto
```

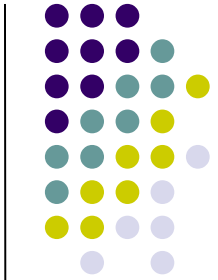
Les redirections



```
$ cat /tmp/toto
cap1.tiff
cap2.tiff
test
$ wc -l < /tmp/toto # lit stdin a partir de /tmp/toto
       3
```

[n]> fichier	Redirige stdout (ou n) vers le fichier
[n]>> fichier	Ajoute stdout (ou n) en fin de fichier
[n]< fichier	Alimente stdin (ou n) à partir du fichier
[n1]>&n2	Unifie stdout (ou n1) au canal n2 en sortie
[n1]<&n2	Unifie n2 et stdin (ou n1) en entrée
[n]>&-	Fermer la sortie standard (ou n)
[n]<> fichier	Associer le fichier en lecture/écriture à stdin (ou n)
[n]<<chaine	Lit sur stdin (ou n) jusqu'à l'apparition de la chaîne

Commandes composées



Une commande composée est :

- soit une liste de commandes → on parle de liste (tout court),
- soit une exécution groupée (en sous-shell ou shell courant),
- soit une instruction de contrôle,
- soit une fonction

Listes

Une liste est une suite de commandes, éventuellement composées (opérateurs ‘&&’, ‘||’, ‘;’, ‘&’, ‘|’), et terminée par un ‘;’, un ‘&’ ou un retour chariot.

Si C1 et C2 sont deux *commandes simples,* alors:

- C1 && C2 : lance C1 puis C2 seulement si C1 a réussi
- C1 || C2 : lance C1 puis C2 seulement si C1 a échoué
- C1 ; C2 : lance C1 puis C2
- C1 & : lance C1 en arrière-plan → **exécution parallèle**
- C1 | C2 : lance C1 et C2 en parallèle et en redirigeant la sortie standard de C1 vers l’entrée standard de C2 (*tube)*

Commandes composées

Les tubes

Forme générale: C1 | C2 où C1 et C2 sont 2 commandes simples

Le tube ('|') permet de lancer C1 et C2 en parallèle après avoir connecté la sortie standard de C1 vers l'entrée standard de C2 - donc C2 lit ce que C1 a produit

Illustration

- ls -l produit des fichiers (1 fichier / ligne) sur sa sortie standard
- wc (word count) compte combien de caractères, mots, et lignes elle trouve sur son entrée standard
- Pour compter combien de lignes ls a produit, on pourrait opérer en deux temps :
  - faire rediriger la sortie de ls vers un fichier
  - faire relire ce fichier par word count

```
[hilairex@pc5352a ~]$ ls -l > /tmp/sortie
[hilairex@pc5352a ~]$ wc < /tmp/sortie
  29  259 1856
[hilairex@pc5352a ~]$
```

Commandes composées

Illustration (suite)

Le tube fait la même chose, sauf qu'il évite de créer */tmp/sortie,* et qu'en outre, ls et wc sont parallélisées :

```
[hilairex@pc5352a ~]$ ls -l | wc
     29     259    1856
[hilairex@pc5352a ~]$
```

A cause de ce mécanisme, la quasi totalité des commandes UNIX qui lisent un fichier et en produisent un autre par défaut, le font systématiquement à partir de leur entrée standard et vers leur sortie standard.

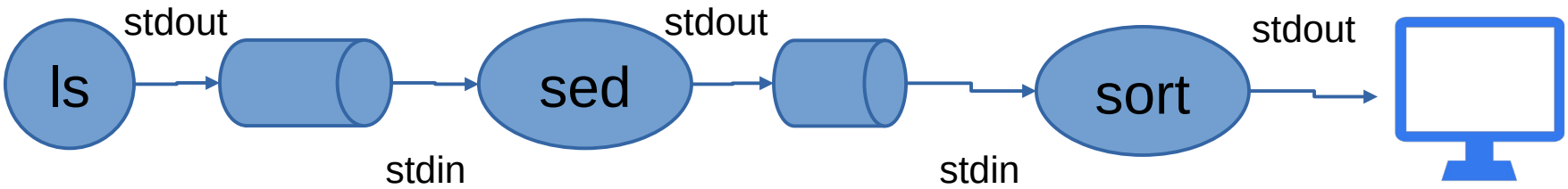
Consulter le manuel ! (**`man`** + commande)

Commandes composées

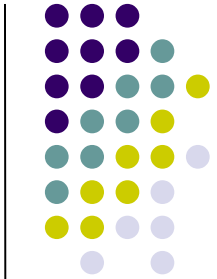
Autre exemple avec sed (stream editor), et sort (tri)

```
$ ls -1
cap1.tiff
cap2.tiff
test
$ ls -1 | sed "s/tiff/TIF/"
cap1.TIF
cap2.TIF
test
```

```
$ ls -1 | sed "s/tiff/TIF/" |
      sort -r
test
cap2.TIF
cap1.TIF
$
```



Commandes composées



Exécution groupée

Formes générales:

{ *liste*; } pour une exécution dans le shell courant

(*liste*) pour une exécution dans un sous-shell

Modifie la priorité d'analyse (lexicale) de liste. Pour l'exécution dans un sous-shell, tout se passe comme si liste constituait un processus à part entière (y compris pour les entrées/sorties) et toutes les variables (ordinaires et d'environnement) du shell courant sont préservées.

Illustration

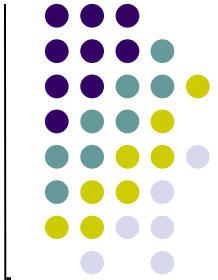
```
{ c1 && c2; } || c3
si c1 alors
        si non c2 alors c3 fsi
sinon
        c3
fsi
```

stdin

≠

```
c1 && { c2 || c3; }
si c1 alors
        si non c2 alors
                c3
        fsi
fsi
```

Commandes composées



Instructions de contrôle

Le shell admet les instructions de contrôle: if, while, for, break, continue, case. La sémantique est la même qu'en langage C.

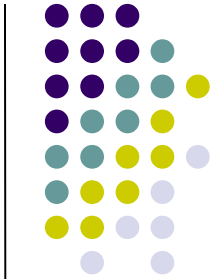
Syntaxe de if

```
if liste; # si le code de retour de liste est zéro
then liste       # alors exécuter cette liste
[elif liste
then liste ] …
[ else liste ]
fi
```

Exemple

```
$ if test "$SHELL"
> then echo SHELL="$SHELL"
> else echo "Variable indefinie"
> fi
SHELL=/bin/bash
$
```

Commandes composées



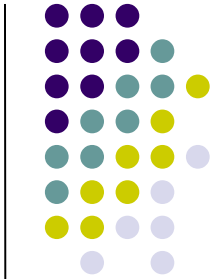
Syntaxe de while

while *liste;* # tant que code de retour de *liste* = 0
do *liste* # exécuter cette liste
done

Exemple:

```
$ i=0
$ while test $i -le 2  # tant que i <= 2
> do echo i=$i ; i=$((i+1))
> done
i=0
i=1
i=2
$ echo $i
3
$
```

Commandes composées



Syntaxe de for

for variable in *mot1 [mot2…]; do*
** *liste* # exécuter cette liste**
done

Exemple:

```
$ for i in un 2 trois; do
> echo $i
> done
un
2
trois
```

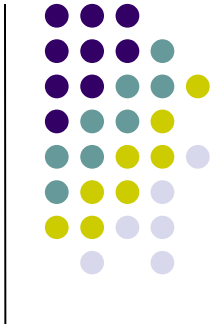
Syntaxe de break et continue

break [n]

continue [n]

Sort du nième niveau (ou du courant) d'itération soit pour arrêter (break) ou poursuivre (continue) . Idem langage C.

Commandes composées



Syntaxe de case

```
case mot in
motif) liste ;;
…
esac
```

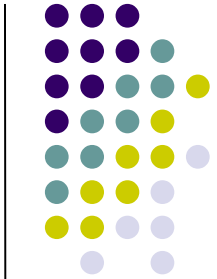
Motif peut comporter plusieurs motifs élémentaires séparés par des '|', chacun suivant les mêmes règles de développement que pour les noms de fichiers. Sémantique identique à celle du langage C, mais break n'est pas nécessaire.

Exemple:

```
$ case $OSNAME in
> Lin*) echo "motif 1";;
> tux|*nux) echo "motif 2";;
> *) echo "defaut";;
> esac
motif 1
```

```
$ case toto in
> Lin*) echo "motif 1";;
> tux|*nux) echo "motif 2";;
> *) echo "defaut";;
> esac
defaut
```

Commandes composées



Les fonctions

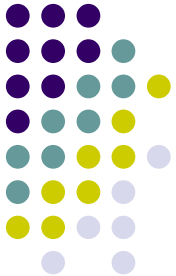
Forme générale: **`nomfonc () liste`**

où `nomfonc` est un mot et `liste` une liste (généralement entre {}).

- N'est commodément utilisable que dans un script.
- Après déclaration nomfonc est vu comme un nom de commande, qui peut recevoir des arguments
- Les arguments passés sont accessibles par $*, $@, $0, $1,…
- La fonction peut renvoyer une valeur entière via l'instruction return, récupérable par l'appelant à travers la variable $?
- Les variables qui ne sont pas déclarées `local` sont en fait celles du shell, et la fonction les altère (!)

Commandes composées

Fonctions (suite)



```
$ compte_fic ()
> {
>     local v
>
>     test -d "$1" && ls -l "$1" | wc -l && v=0 || v=1
>     return $v
> }
$
$ compte_fic /tmp/
       3
$ echo "Retour1=$?"
Retour1=0
$ compte_fic /inexistant/
$ echo "Retour2=$?"
Retour2=1
```

Scripts shell

- On peut stocker un ensemble de lignes de commandes dans un fichier texte pour en faire un script shell exécutable.
- Pour pouvoir être lancé, le fichier créé doit avoir le droit d'exécution (chmod +x)
- Si la première ligne du fichier débute par #!, elle doit être suivie du chemin du shell à lancer. Si ce n'est pas le cas, le shell courant l'exécutera dans un sous-shell.
- On peut alors lancer le script en l'appelant directement par son nom

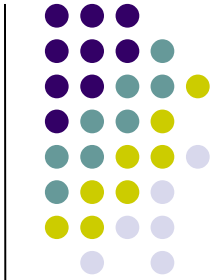
```
$ ls monscript.sh
-rw-r--r--   1 xavier  xavier  165 Feb 26 16:44
   monscript.sh
$ chmod +x monscript.sh
$ cat monscript.sh
#!/bin/sh
#
```

Scripts shell

```
compte_fic ()
{
    local v

    test -d "$1" && ls -l "$1" | wc -l && v=0 || v=1
    return $v
}


compte_fic "$1"
if test $? -ge 1
then echo Erreur
fi
$ ./monscript.sh .
       7
$ ./monscript.sh /inexsitant
Erreur
$
```



Principales commandes
Principaux utilitaires

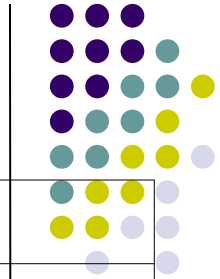
Commandes internes

Ce sont des commandes propres au shell - elles ne sont ni des scripts, ni des fichiers binaires, ni des fonctions. Commandes essentielles:

Nom	Fonction	Exemple
pwd	Donne le chemin courant	$ pwd /Users/xavier/tmp $ echo $PWD /Users/xavier/tmp

cd [dir]	Change le chemin courant à $HOME (ou dir)	$ cd /tmp
		$ pwd
		/tmp
		$ cd
		$ pwd
		/Users/xavier

Commandes internes



Nom	Fonction	Exemple
read	Lit une variable depuis l'entrée standard	$ read ligne Ceci est un test. echo $ligne Ceci est un test.
shift [n]	Décale tous les paramètres de 1 (ou n) vers la gauche	shift # $1 devient $0, $2 devient $1,…
jobs	Liste les processus lancés en arrière-plan	$ jobs [1]- Running xterm & (wd: ~/tmp) [4]+ Running xterm & (wd: ~/tmp)
fg [n]	Ramène le dernier processus d'arrière-plan (ou n) au premier-plan	$ fg 4 xterm (wd: ~/tmp)

Commandes externes

- Au sens large, comprennent tous les fichiers binaires accessibles par la variable PATH.
- La plupart des utilitaires et commandes système se trouvent dans :

/bin,/sbin,/usr/bin,/usr/sbin,/usr/local/bin,/usr/X11R6/bin

Accessibles lors du boot

Accessibles après montage de /usr

Accessibles sur réseau local

Clients X11

Une commande très utile : man

`man -a commande`

affiche les pages manuel décrivant `commande`

`man -k mot-clef`

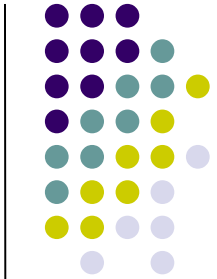
recherche les commandes qui concordent avec le mot-clé fourni.

Commandes externes

Commandes diverses

Nom	Description
`test ou [`	**Test l'existence de fichiers/répertoires, l'égalité, la supériorité, l'infériorité sur des entiers et des chaînes de caractères**
`clear`	Efface le terminal
`id`	Affiche les informations d'identité de l'utilisateur
`uname`	Nom du système d'exploitation
`who`	Liste les utilisateur connectés
`passwd`	Modification du mot de passe utilisateur
`su`	Changer d'identité utilisateur (substitute user)
`hostname`	Nom de la machine exécutant le shell

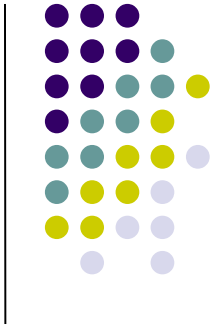
Commandes externes



Manipulation de fichiers

Nom	Description
`cp`	Copie de fichiers
`mv`	Renomme/déplace des fichiers
`mkdir`	Créer des répertoires
`rmdir`	Supprimer des répertoires
`ls`	Lister les fichiers
`find`	Chercher des fichiers dans l'arborescence
`chmod`	Modifier les droits d'accès
`chown`	Changer le propriétaire d'un fichiers
`mktemp`	Créer un fichier temporaire

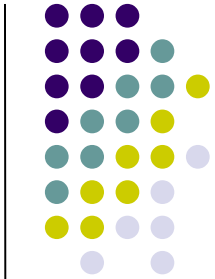
Commandes externes



Commandes sur fichiers texte

Nom	Description
`cat`	Concatène des fichiers
`sort`	Trier les lignes d'un fichier
`cut`	Sélectionner des champs/colonnes
`paste`	Fusionner des champs entre eux
`tr`	Substituer des caractères à d'autres
`grep`	Sélectionner des lignes
`sed`	Editeur de flux (stream editor)
`vi, vim, ed`	Editeurs en mode texte
`less`	Consulter un fichier interactivement

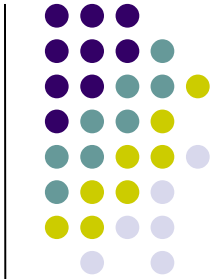
Commandes externes



Commandes sur processus

Nom	Description
`ps`	Afficher les processus chargés
`top`	Afficher les processus chargés en temps réel
`kill`	Envoyer un signal à un processus
`renice`	Changer la priorité d’exécution d’un processus

Commandes externes



Commandes réseau

Nom	Description
`ifconfig`	Afficher les informations sur les interfaces réseau
`netstat`	Afficher l'état des ressources réseau (trafic, sockets, ports, tables de routage)
`ping`	Envoyer des paquets ICMP
`arp`	Résolution d'adresse Internet->Ethernet
`traceroute`	Afficher l'itinéraire hôte-hôte par ICMP
`nslookup`	Résolution nom d'hôte -> Adresse IP
`ypbind`	Maintenance de la liaison avec un serveur NIS