Introduction aux outils Unix

Notions de base, shell, expression régulière, exécution différée…

Pierre Reinbold

pre@info.ucl.ac.be

INGI, UCL

www.info.ucl.ac.be

Une partie de ces transparents est inspirée du cours Linux du Namulug (www.namurlug.org)

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Introduction aux systèmes Unix

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

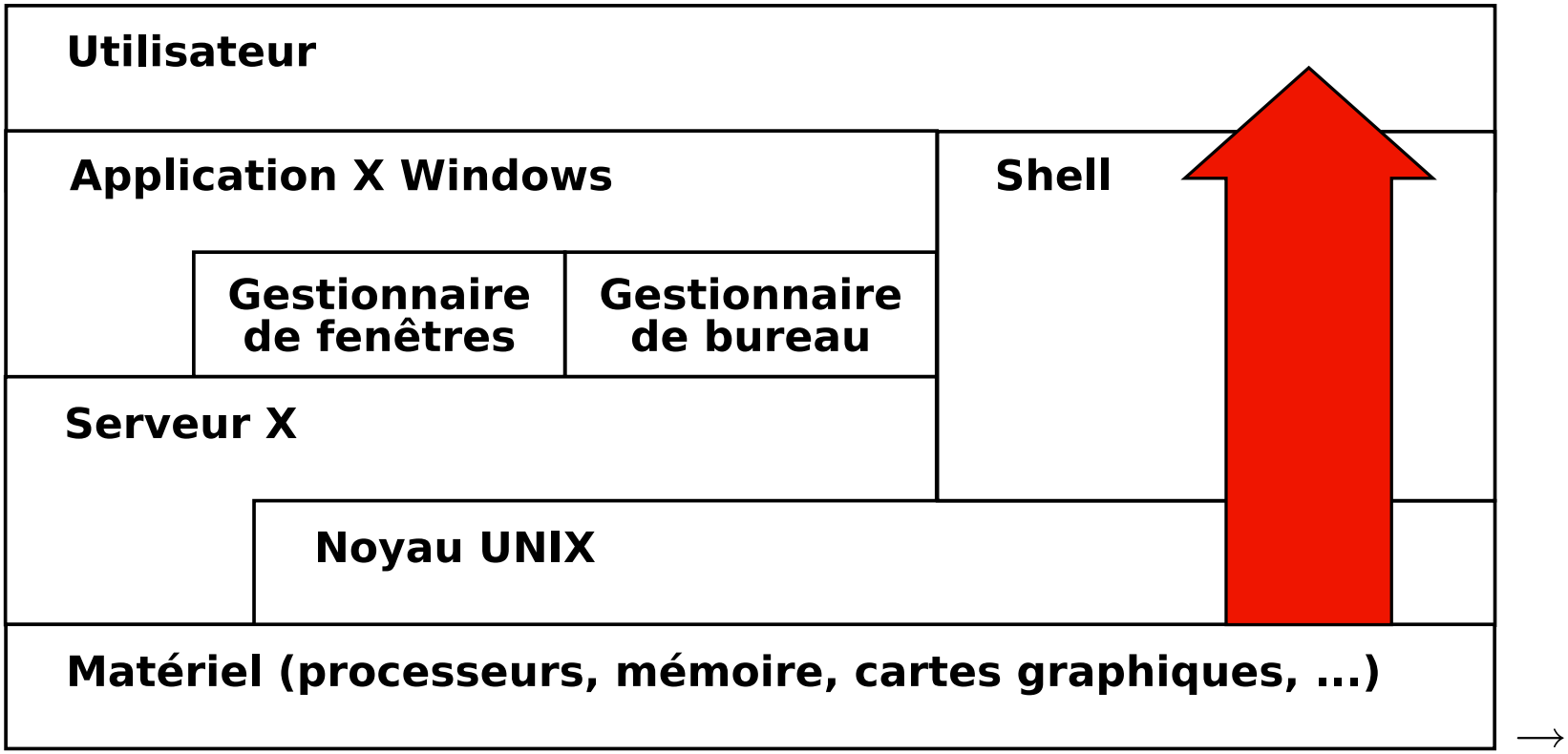
Architecture Unix

- Un Unix ? **Des** Unix !
  - TODO
  - blabla historique Unix
  - mettre en éévidence le poids historique
  - des multiples branches de Unix
  - éventuellement un schémas de branches ?

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Architecture Unix

- Principes généraux et blocs fonctionnels commun



Le chemin le plus efficace pour exploiter les ressources de la machines passe par le shell ! ☺

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Architecture Unix (2)

- Systèmes multi-utilisateurs
  - L'utilisation du système passe par une étape d'authentification : **login** et **password**
  - Les utilisateurs appartiennent à des **groupes** (groupe de base, groupes supplémentaires)
  - Les droits d'accès (sur le système de fichier principalement) sont définis sur base de
    - utilisateur propriétaire (d'un fichier)
    - membres du groupe propriétaire
    - reste du monde
  - Login (txt) → identifiant (num.) : **user id** (uid)
  - Nom de groupe (txt) → identifiant (num.) : **group id** (gid)
  - **Le super utilisateur**
    - login : **root** (uid = 0)
    - groupe de base : **root** (gid = 0)

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Architecture Unix (3)

- Systèmes multi-tâches
  - Plusieurs processus tournent en même temps dans le système
  - Processus associés à l'utilisateurs qui les lance, avec son groupe de base (par défaut) → uid et gid du processus
  - Droits associés à un processus == droits associés à l'utilisateur et au groupe de ce processus
  - Processus identifié par un numéro : **process id** (pid)
  - Ressources systèmes distribuées aux différents processus en fonction de leurs **priorités relatives**.
  - Interface sécurisée (liées aux droits des utilisateurs) pour interagir avec les processus
    - interruption/relance
    - changement de priorité
    - passage en tâche de fond/retour au premier plan
    - envoi de signaux spécialisés
    - ...

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Système de fichiers

- Concept de **fichier** == central dans tout système Unix
  - Fichier == unité élémentaire pouvant contenir des informations
  - adressable/consultable/modifiable à travers un **système de fichiers** (ex. reiserfs, ext3, ufs, ntfs, . . .)
  - toujours (au moins) lié à un utilisateur propriétaire et un groupe propriétaire
  - généralement structuré → type d'information contenue. L'extension **peut** indiquer de quel type de fichier il s'agit : **ce n'est qu'une indication !**
  - sous Unix, les **noms de fichiers** sont traditionnellement **sensibles à la casse** !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Système de fichiers

- Concept de **répertoire** → structurer logiquement les fichiers
  - Peut contenir des fichiers ou des sous-répertoires
    ⇒ structure **arborescente**
  - Structure des répertoire Unix «standardisée» de fait (tradition, LSB, ...). Par exemple,

`/`	→	racine générale
`/etc`	→	fichiers de configuration
`/home`	→	répertoires personnels des utilisateurs
`/usr`	→	programmes utilisateurs
`/usr/local`	→	programmes locaux (à une organisation)
`/bin`	→	applications/exécutables systèmes (binaires)
`/sbin`	→	applications systèmes réservées au root
`/dev`	→	accès aux périphériques

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Système de fichiers (2)

- Ressources systèmes multiples → mode d'accès rationnalisé à travers **des accès fichiers**. (au moins en partie)

Application	Kernel
Accès fichier	Accès périphériques (→ pilote dédié)

- Ex. Les périphériques :
  - Périphériques identifiés par **fichiers de périphériques**, situés dans `/dev/`
  - Noms de fichiers, permissions et modes d'accès pour un périphérique donné → dépendants du kernel
  - Deux types de périphériques :
    - **blocs** *i.e.* adressables (disques, cdrom, dvd, . . .)
    - **caractères** *i.e.* non-adressables : flux de données (carte son, souris, port série, . . .)

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Système de fichiers (3)

- Quelques périphériques et leur fichier spécifique :

b	disques IDE	`/dev/hd?`
b	partitions	`/dev/hd??`
b	lecteur de disquettes	`/fd`
b	périphériques SCSI	`/sd?`
c	port série	`/dev/ttyS?`
c	port PS2	`/dev/psaux`
c	souris	lien : `/dev/mouse` vers device connecté
c	port parallèle	`/dev/par`
c	consoles virtuelles	`/dev/tty?`
c	générateur aléatoire	`/dev/random` ou `/dev/urandom`
c	générateur de zéro	`/dev/zero`
c	le trou noir	`/dev/null`

...

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

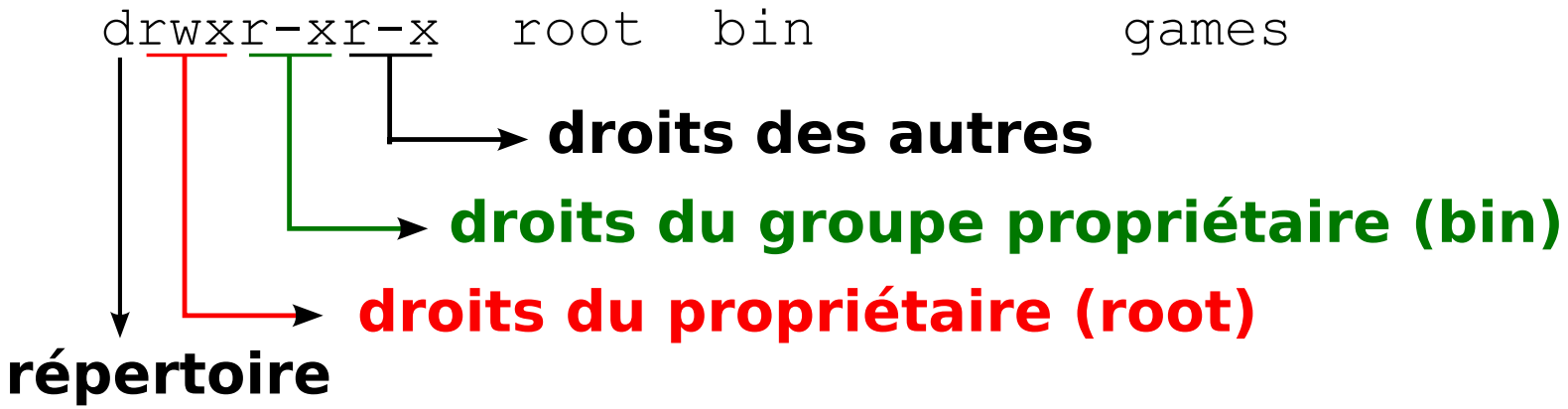
Système de fichiers (4)

- 3 type de droits par fichier/répertoire
  - Autorisation de **lire**
  - Autorisation **d'écrire/de supprimer**
  - Autorisation **d'éxécuter** un fichier, de «traverser» un répertoire
- 3 ensembles d'utilisateurs
  - **L'utilisateur propriétaire** (un seul !), par défaut le créateur du fichier/répertoire
  - **Le groupe système propriétaire** (un groupe), par défaut le groupe principal du propriétaire
  - **Les autres**, le reste du monde
- type de droits → défini indépendamment par ensemble d'utilisateur(s)

Système de fichiers (5)

- Signification différente en fonction de l'inode

droit	fichier	répertoire
lecture	lire le fichier	lister le contenu
écriture	modifier	créer/supprimer fichier/ss-répertoire
éxécution	éxécuter	«traverser» *i.e.* atteindre fichiers/ss-répertoires



UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Système de fichiers (6)

- Droits étendus : 3 flags (notions)

drapeau	fichier	répertoire
suid	éxécution avec l'uid du propriétaire (sécurité !)	-
gid	éxécution avec le gid du groupe propriétaire (sécurité !)	fichiers/ss-répertoires appartiennent au groupe propriétaire
stiky	exécutable restant en mémoire	suppression des seuls fichiers/ss-répertoires dont un utilisateur est propriétaire

Ex. **répertoire partagé** → gid et sticky bit activés

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Système de fichiers (7)

- Extensions possibles des droits de base
  - droits différenciés pour plusieurs groupes
  - droits différenciés sur base d'ACL
  - plus de types de droits
  - évolution dynamique des droits

  → ACL ext3, XFS, . . .

- En général, il se révèle que l'utilisation de droits étendu est contre-intuitive, mal acceptée par les utilisateurs, et généralement contre-productive d'un point de vue sysadmin !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Ligne de commande

- Prompt
- shell

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Ligne de commande (2)

- Syntaxe

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Ligne de commande (3)

- Tips
  - raccourcis
  - historiques
  - environnement

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Ligne de commande (4)

- `man`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Commandes de base : se situer, bouger

- **pwd**
- **cd**

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Commandes de base (5) : lister

- `ls`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Commandes de base (6) : répertoires

- `mkdir`
- `rmdir`

UCL - INGI

Département d'ingénérie informatique

www.info.ucl.ac.be

Commandes de base (7) : changer

- `mv`
- `cp`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Commandes de base (8) : inode

- `stat`
- `touch`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Commandes de base (9) : supprimer

- `rm`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Commandes de base (10) : concaténer

- `cat`

UCL - INGI

Département d'ingénérie informatique

www.info.ucl.ac.be

Commandes de base (11) : lire (mal)

- `more`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Commandes de base (12) : lire (mieux)

- `less`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Commandes de base (13) : usage

- `df`
- `du`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Commandes de base (14) : chercher

- `which`

UCL - INGI

Département d'ingénérie informatique

www.info.ucl.ac.be

Commandes de base (15) : chercher

- **`find`**
- **`locate`** et **`updatedb`**

UCL - INGI

Département d'ingénérie informatique

www.info.ucl.ac.be

Commandes de base (16) : chercher

- `grep`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Commandes de base (17) : intervertir

- `tr`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Gestion des processus

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Notion de processus

- blabla
- PID, UID
- signaux

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Gestion : lister

- `jobs`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Gestion (2) : lister

- **ps**

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Gestion (3) : lister

- `top`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Gestion (4) : lister

- `pstree`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Gestion (5) : signaux

- `kill`

Tâche de fond

- Lancer en tâche de fond

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Tâche de fond (2)

- Interrompre

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Tâche de fond (3)

- `bg`
- `fg`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Niveaux d’éxécution

- Notions

UCL - INGI
Département d’ingénérie informatique
www.info.ucl.ac.be

Niveaux d’éxécution (2)

- `nice`

UCL - INGI
Département d’ingénérie informatique
www.info.ucl.ac.be

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash
Notions Élémentaires

Le shell Bash

- Flux standards

`stdin`	:	l'entrée standard	⟷ clavier
`stdout`	:	la sortie standard	⟷ écran
`stderr`	:	le « standard d'erreur »	⟷ écran

Par défaut

Toutes les commandes ne définissent/n'utilisent pas forcément chacun de ces flux !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (2)

Chacun de ces flux peut être redirigé par la syntaxe
`<command> <symbole de redirection> <nom de fichier>`.

- Redirection de la sortie standard : « > » ou « >> »

  Ce qui apparaissait à l'écran est redirigé dans un fichier

  - `$ ls -alsh /root > out`

    `$ echo 'prof2:x:505:100:un prof:/home/prof2:/bin/bash' >> /etc/passwo`
  - Avec « > », le contenu du fichier est écrasé,
  - avec « >> », la sortie est ajoutée à la fin du fichier (qui est crée s'il n'existe pas déjà).
  - Les erreurs ne sont pas redirigées !

- Redirection du standard d'erreur : « 2> » ou « 2>> »
  - `$ ls -alsh /root 2> out`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (3)

- Redirection de la sortie standard **et** du standard d'erreur : « &> » ou « &>> »
  - `$ ls -alsh /root &>> out`
  - `$ ls -alsh /root >> out 2>&1`
- Redirection de l'entrée standard : « < »

  La demande d'une entrée au clavier est redirigée vers un fichier
  - `$ write root pts/5 < message.txt`

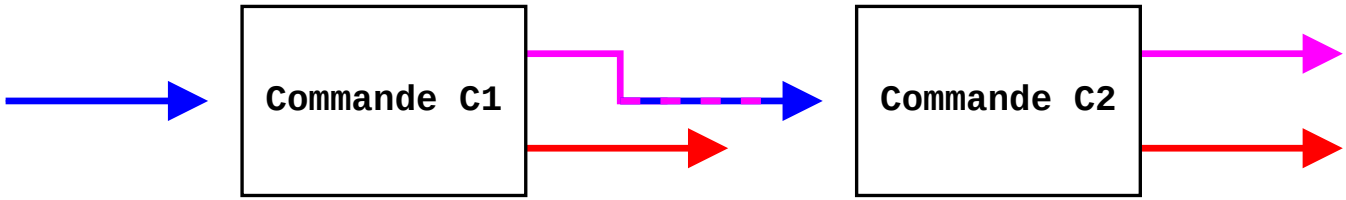
    Exécution de **`write`** en mode non-interactif puisque l'entrée standard proviendra de **message.txt** !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (4)

- Le pipe

  Idée : utiliser la sortie standard d'une commande comme entrée standard d'une autre !



- Mise en œuvre : utilisation de « | »

  `<commande1> | <commande2>`

  Le résultat de `commande1` sert de données d'entrée à `commande2`.

  - `$ cat /etc/passwd | grep prof`
  - `$ users | grep student`
  - `$ w | grep tty | grep prof`

Extrêmement utilisé dans le monde Unix !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (5)

- Scripts Bash

A partir d'une certaine complexité, il est souhaitable que les commandes Bash puissent être regroupées dans des fichiers de *script*. Si ces fichiers sont exécutables, ils deviennent de nouvelles commandes pour le système.

- Mise en œuvre : un fichier de script doit
  - commencer par la ligne `#!/bin/bash`
  - être exécutable (`$ chmod +x monfichier.sh`)
- Exemple : `HelloWorld.sh`

```
#!/bin/bash
# Ceci est mon joli hello world en bash !
echo " Hello "
echo " World "
echo -n " Quel est votre nom ? "
read NM
echo " Bonjour $NM "
```

→ utilisation de **echo**, **read** et d'une variable (`$NM`)

UCL - INGI
Département d'ingénierie informatique
www.info.ucl.ac.be

Le shell Bash (6)

- Saisie d'arguments

Les scripts Bash peuvent être invoqués avec un nombre quelconque d'arguments.

- Exemple : `$ HelloWorld.sh Toto`

```
#!/bin/bash
NOM=$1
echo " Hello "
echo " World "
echo " Bonjour $NOM "
```

- les arguments sont enregistrés dans des variables spéciales notées $1 à $9,
- $0 contient le nom du script,
- $# contient le nombre d'arguments,
- $@ contient la liste de tous les arguments.
- $$ contient le PID du script

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (7)

- Quelques variables prédéfinies
  - L'environnement de l'utilisateur, entre autres :
    - `$PATH`
    - `$HOME`
    - `$USER`
    - `...`
  - `$UID` contient l'UID réel de l'utilisateur, `$EUID` contient l'UID courrant de l'utilisateur (! usage de `su`)
  - `$SECONDS` le nombre de secondes écoulées depuis le lancement du script
  - `$RANDOM` un nombre entier pseudo-aléatoire entre 0 et 32767 (assez mauvais mais souvent utile, pas pour la crypto !)
  - `$IFS` le séparateur d'arguments, par défaut toute suite non vide de tab et/ou de blancs (modifiable !)

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (8)

- Utiliser des tableaux (assez peu courant en pratique)
  - Il existe de (trop) nombreuses manières de déclarer un tableau, en voici quelques-unes

    ```
    array = ( un deux trois quatre )
    array[2] = 3
    ```

    ⇒ indexation à partir de l'indice 0 !
  - Récupérer les valeurs d'un tableau

    ```
    echo ${array[2]}
    ```
  - Récupérer toutes les valeurs d'un tableau

    ```
    echo ${array[@]}
    ```
  - Connaître le nombre d'éléments d'un tableau

    ```
    echo ${#array[@]}
    ```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (9)

- Substitution par le résultat d'une commande
  - Une des fonctionnalités les plus utilisées de Bash
  - `‘<commande>‘` est remplacé par Bash par le résultat (stdout) de `commande`, exécutée dans un « sous-shell ».
  - Syntaxe plus explicite (et plus récente) `$(<commande>)`
  - Exemple : produire un message de log

```
TMPFILE=/tmp/tmplog.$$.$RANDOM
DATEPREFIX=$(date "+%d/%m/%Y-%H:%M:%S")
...
echo "$DATEPREFIX : un beau message de log" >> $TMPFILE
...
```

- Endormir le script : **`sleep`** provoque l'interruption du script pendant un nombre de secondes passé en argument.

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (10)

- Sortie du script : **`exit`**
  - **`exit`** provoque l'interruption du script et renvoie son argument comme *code de retour*.
    Exemple : "`exit 0`" pour une terminaison normale/réussie (standard Unix)
  - Le code de retour d'une commande peut être récupéré par la variable prédéfinie "$?"

```
$ echo "tutu"
tutu
$ echo $?
0
$ ls /root
ls: /root: Permission denied
$ echo $?
1
```

  - ⇒ une commande réussie est une commande dont le code de retour est **zéro** !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (11)

- Calculs entiers

Il est possible d'évaluer des expressions entières :

- Exemple : `compute.sh`

```
#!/bin/bash
echo "Je calcule X*Y"
echo "Entrez X"
read X
echo "Entrez Y"
read Y
echo "X*Y = $X*$Y = $[X*Y]"
```

→ utilisation des crochets `$[]` pour indiquer la nécessité d'évaluer ce qui est entre-eux.

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (12)

- Structures de contrôle

Bash présente les structures de contrôle habituelles d'un langage de programmation.

- Sauts conditionnels :
  - `if/then/else`
  - `case`
- Boucles :
  - `for`
  - `while`
  - `until`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (13)

- Saut conditionnel : instruction `if`
  - Une alternative

```
X=10
Y=5
if test "$X" -gt "$Y" ; then
    echo "$X est plus grand que $Y"
else
    echo "$X est plus petit ou égal à $Y"
fi
```

  - Alternatives multiples

```
X=10
Y=5
if test "$X" -gt "$Y" ; then
    echo "$X est plus grand que $Y"
elif test "$X" -lt "$Y" ; then
    echo "$X est plus petit que $Y"
else
    echo "$X est égal à $Y"
fi
```

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (14)

- Opérateurs de comparaison

Sur les entiers	`-eq`	`-ne`	`-lt`	`-gt`	`-le`	`-ge`
Sur les chaînes	`==`	`!=`	`<`	`>`	`<=`	`>=`

- Opérateurs booléens (idem **C**)
  - && : AND logique, retourne vrai si les deux alternatives sont vraies
  - || : OR logique, retourne vrai si l'une des deux alternatives est vraie
  - ! : NOT logique, retourne vrai si la proposition est fausse
  - Exemple :

```
! grep newuser /etc/passwd && useradd newuser
```

Ajoute l'utilisateur au système si son loggin n'est pas trouvé dans le fichier **/etc/passwd**.

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (15)

- Tests sur les fichiers

Bash présente une foule de tests possibles à effectuer sur les fichiers. On les utilise généralement dans l'instruction `if`, avec la syntaxe suivante :

```
if test <test fichier> <nom fichier> ; then
    <instructions>
fi
```

Quelques exemples de valeur que peut prendre `test fichier` :

- Existence : `-e`
- Directory : `-d`
- Fichier normal : `-f`
- Fichier lisible : `-r`
- Fichier inscriptible : `-w`
- Fichier exécutable : `-x`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (16)

- Exemple : vérifier si un fichier existe et en faire un backup si c'est le cas

```
#!/bin/bash

SUFFIX=".bak"
FICHIER=$1
if test -f $FICHIER ; then
    cp $FICHIER $FICHIER$SUFFIX
fi
```

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (17)

- Boucle : instruction `for`

```
#!/bin/bash

for I in "la vache" "le mouton" "le poulet" "le cochon" ;
do
    echo "$I est un animal de ferme"
done
```

Parcours des arguments :

```
#!/bin/bash

echo $#
for I in $@;
do
    echo $I
done
```

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Le shell Bash (18)

Boucle : instruction `while`

```
#!/bin/bash

N=1
while test "$N" -le "10"
do
    echo "Number $N"
    N=$[N+1]
done
```

Boucle : instruction `until`

```
#!/bin/bash

N=1
until test "$N" -gt "10"
do
    echo "Number $N"
    N=$[N+1]
done
```

Le shell Bash (19)

- Interruptions d'exécution
  - **`break`** provoque l'interruption de la boucle courante et l'exécution des commandes suivant le `done`
  - **`continue`** provoque l'interruption de l'itération courante et le passage immédiat à une nouvelle itération (avec vérification du test de boucle le cas échéant)
  - Exemple : traiter les fichiers textes d'un répertoire :

```
for FICHIER in $(ls | grep \\.txt);
do
    if ! test -r $FICHIER; then
        continue
    fi
    echo "============================"
    echo "Contenu de $FICHIER"
    echo "============================"
    cat $FICHIER
    echo "============================"
    echo "Fin de $FICHIER"
    echo "============================"
done
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Expressions Régulières

Expressions Régulières

- Définition : séquence de caractères formant un canevas (*pattern*) utilisé pour décrire des chaînes de caractères dans un texte.
- Exemple textuels :
  - Tous les mots commençant par un "a"
  - Une ligne comportant le mot "`totolulub212`"

    → définition d'une *grammaire*...
- utilisation courante dans les commandes **`grep`** et **`sed`**
  - **`egrep`** : recherche de pattern dans les lignes de l'input standard ; variante de **`grep`** qui utilise les expressions régulières étendues
  - **`sed`** : traitement de l'input standard sur base de pattern

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Expressions Régulières (2)

- Utilisation
  - Dans une recherche, généralement ligne par ligne, les expressions régulières correspondent toujours à *la plus longue sous-chaîne* correspondant au pattern de l'expression.
  - Quelques exemples avec **`grep`** (`-w` pour une recherche de mots complets) :
    - "`egrep -w 't[a-i]e'`" matche avec `tae`, `tbe`, ... `tie`
    - "`egrep '^.*Totolulub212.*$'`" matche avec toute ligne contenant `Totolulub212`
    - "`egrep -w '(toto|lulu|b212)'`" matche avec `toto`, `lulu` ou `b212`

UCL - INGI
Département d'ingénierie informatique
www.info.ucl.ac.be

Expressions Régulières (3)

- le wildcard canonique : "`.`" correspond à n'importe quel caractère (un seul !)
  - "`a.b`" matche avec `a1b`, `arb`, ...
- "`*`" correspond à la répétition du caractère (ou de l'expression) précédent(e), de 0 à un nombre quelconque de fois
  - le pattern "`.*`" correspond à n'importe quelle suite de caractères (même vide)
  - "`.*toto.*`" matche avec toute ligne de caractères contenant "`toto`"
  - "`ba*b`" matche avec "bb", "bab", "baab", ...
  - caractère d'échappement "`\`", "`\.`" matche avec "`.`"

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Expressions Régulières (4)

- Intervalles de caractères : "`[a-n]`" matche avec tout caractère compris entre "a" et "n"
  - "`[a-n]*`" matche avec la répétition des caractères entre a et n, 0 ou plusieurs fois
  - "`[aeiou]`" matche avec `a`, `e`, `i`, o ou u
  - "`[^aeiou]`" matche tout caractère différent de `a`, `e`, `i`, o ou u

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Expressions Régulières (5)

- Répétition de caractères ou de sous-expressions : `"{}"`

Expression	Répétition(s)
`"{x,y}"`	de x à y fois
`"{x,}"`	plus de x fois
`"{x}"`	exactement x fois
`"{,y}"`	jusqu'à y fois

- Raccourcis :
  - `"*"` est synonyme de `"{0,}"`
  - "?" est synonyme de `"{0,1}"`
  - "+" est synonyme de `"{1,}"`
- Exemples :
  - `"ab{1,3}c"` matche avec `abc`, `abbc` et `abbbc`
  - `"ab?c"` matche avec `ac` et `abc`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Expressions Régulières (6)

- Entités prédéfinies :
  - "^" correspond au début de ligne
  - "$" correspond à la fin de ligne
  - "`[[:alpha:]]`" est synonyme de "`[a-zA-Z]`"
  - "`[[:upper:]]`" est synonyme de "`[A-Z]`"
  - "`[[:lower:]]`" est synonyme de "`[a-z]`"
  - "`[[:digit:]]`" est synonyme de "`[0-9]`"
  - "`[[:alnum:]]`" est synonyme de "`[0-9a-zA-Z]`"
  - "`[[:space:]]`" correspond à tout espace blanc, y compris les tabulations

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Expressions Régulières (7)

- Alternative "`( | )`" :
  - "`(toto|lulu|b212)`" correspond à l'alternative `toto`, `lulu` ou `b212`

- Regroupement "`( )`" :
  - les sous-expressions peuvent être regroupées via des parenthèses, elles sont alors traitées comme un unique objet
  - "`(t[a-r]*b212)*`" correspond à la répétition du pattern "`t[a-r]*b212`"

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Expressions Régulières (8)

sed

- Outil très puissant de traitement de stream (avec **awk**)
- **sed** prend son entrée sur l'input standard et effectue sa sortie sur l'output standard
- Nous l'utiliserons dans un seul cas : *la substitution*
  - `sed -e "s/<pattern>/<string à substituer>/[g]"` remplace la première, (ou toutes si g est présent) occurrence du pattern `pattern` dans l'input standard (ligne par ligne)
  - `$ echo "toto,lulu,b212" | sed -e "s/,/ /g"` donne `"toto lulu b212"`
  - Suppression des tags HTML :

    ```
    $ echo '<H1>Titre Premier</H1>' | sed -e "s/<[^<>]*>//g"
    ```

    renvoie `"Titre Premier"`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Expressions Régulières (9)

*Echappement avec **sed** et **grep***

- **sed** et **grep** utilisent des expression régulières simples.
- Il faut échapper (\) les caractères suivants : ?, +, {, }, |, (et)
- Exemples :
  - `egrep '(toto|lulu)'`
  - `grep '\(toto\|lulu\)'`
  - `sed -e 's/\(toto\|lulu\)/xxxx/g'`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Expressions Régulières (10)

- Exemple de script utilisant les expressions régulières, **`grep`** et **`sed`** : obtenir les utilisateurs d'un groupe, à partir du fichier **/etc/group**

```
#!/bin/bash

if test -z $1 ; then
    echo "usage : $0 <nom de groupe>"
    exit
fi

USERS=‘grep $1 /etc/group | \
        sed -e "s/^.*[0-9]://" | \
        sed -e "s/,/\ /g"‘

for NAME in $USERS;
do
    echo $NAME
done
```

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Expressions Régulières (11)

Exercice 1 Construire un pattern pour représenter :

- un nom simplifié d'utilisateur UNIX, limité aux caractères alphanumériques avec "`-`" et "_", doit commencer par une lettre ou un chiffre
- une adresse IP, ex. `138.48.32.107`
- un numéro de téléphone, ex. `(064) 33 44 55`
- une communication structurée, ex. `***030/3195/40178***` ou `+++030/3195/40178+++`
- une ligne des fichiers **/etc/group** ou **/etc/passwd**, ex. `etudiants:x:504:student3,student2,student1`
- une ligne d'output de la commande `ls -l`, ex. `-rw-r--r--  1  root  root  15 May 26 15:53 msg.txt`
- une date au format UNIX standard, l'output de la commande **`date`**, ex. `Tue May 27 23:23:25 CEST 2003` (hint : construire des sous-expressions pour les différents champs)

Exercice 2 Extraire l'adresse IP d'une interface dont le nom est passé en argument, en utilisant l'output de la commande **`ifconfig`** (`ifconfig <nom interface>`)

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Exécution de commandes différées : `cron` et `at`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

cron

- De nombreuses tâches doivent être exécutées périodiquement :
  - nettoyage des fichiers temporaires (ex. dans **/tmp**),
  - backups,
  - mises à jour et maintenances,
  - ...
- Le service **`cron`** permet de programmer l'exécution de ces tâches périodiquement, sur base de fichiers de configuration,
  - pour le système (définis par `root`),
  - pour chaque utilisateur (définis par l'utilisateur).

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

cron (2)

- Configuration système : **/etc/crontab** lu à chaque démarrage
  - Syntaxe : `<heure et jour> <utilisateur> <commande>`
  - Exemple :

    ```
    0 12 21 4 * root echo 'Anniversaire du système LinuxClasses !!!' | wa
    ```

    Provoque l'envoi, avec l'identité de `root`, du message anniversaire tous les 21 avril à 12h00.
  - Format de date :

`<minute>`	`<heure>`	`<jour>`	`<mois>`	`<jour de la semaine>`
0-59	0-23	1-31	1-12	0-6

    - "`*`" = toutes les valeurs
    - possibilité de liste : "`<jour> = 1,2,3`" représente les trois premiers jours du mois
    - "`/`" indique un pas : "`<heure> = */2`" représente une heure sur deux (→ les heures paires)

UCL - INGI
Département d'ingénierie informatique
www.info.ucl.ac.be

cron (3)

- Définitions automatiques :

```
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

  Pour lancer une commande avec la périodicité voulue : placer un script dans les répertoires correspondants (dépendant des distributions)

- Attention : après chaque modification au fichier **/etc/crontab**, le service **cron** doit être relancé :

```
# service crond reload
```

- Output des commandes :
  - commandes **cron** exécutées en tâche de fond ⇒ pas de terminal pour la sortie standard
  - la sortie standard est redirigée et enregistrée dans un mail → utilisateur mentionné par `MAILTO`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

cron (4)

- Accès pour les utilisateurs : **crontab**
  - Syntaxe : `crontab [-e | -l | -r] [-u <user>]`
    - `-e` édite le fichier `cron` de l'utilisateur (dans **/var/spool/cron/**)
    - `-l` donne la liste des commandes programmées
    - `-r` vide le fichier
    - `-u <user>` permet au `root` d'éditer le fichier `cron` de l'utilisateur `user`.
    - `crontab [-u <user>] <file>` installe le ficher `file` comme fichier crontab de `user`.
  - Protection d'accès :
    - Les fichiers **/etc/cron.allow** et **/etc/cron.deny** définissent qui peut ou non utiliser les services de **cron**.
      - Si **/etc/cron.allow** existe ⇒ les seuls utilisateurs autorisés sont ceux **mentionnés** dans ce fichier,
      - Si **/etc/cron.deny** existe ⇒ les seuls utilisateurs autorisés sont ceux **non-mentionnés** dans ce fichier.

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

at

- Service similaire à **cron**, mais les commandes ne sont exécutées qu'une fois, au moment spécifié.
- Les commandes à exécuter sont ajoutées via l'instruction **at** :

```
at <heure et jour>
... entrer des commandes ...
<CTRL-D>
```

  - `heure et jour` peut être défini de plusieurs façons. Par exemple :
    - `at 23:59 December 24`
    - `at 12 am Thursday`
    - `at now + 2 hours`
    - `at midnight - 5 minutes`
  - Possibilité de mode non-interactif avec l'option `-f`

    `at -f <fichier de commande> <heure et jour>`

    (ou la redirection de l'input standard)

`at` (2)

- Liste des jobs programmés : **`atq`**

```
$ atq
12      2003-05-30 00:00 a devil
```

- Suppression d'une commande programmée : **`atrm`**

  `atrm <numéro>` où l'argument est le numéro de job listé par **`atq`**

- Protection d'accès via les fichiers **/etc/at.allow** et **/etc/at.deny** (même sémantique que pour **`cron`**)

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be