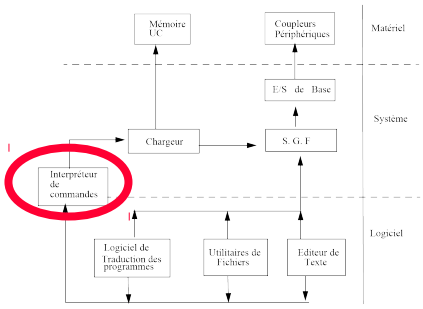
Langage(s) de commande
et
Programmation shell

I : Langage(s) de commande

Plan du chapitre

1. Qu'est ce qu'un interpréteur de commandes ?
2. Comment désigne-t-on les fichiers sous Unix ?
3. Les redirections d'entrées/sorties
4. Quelques commandes de manipulation de fichiers

1. Interpréteur de commandes

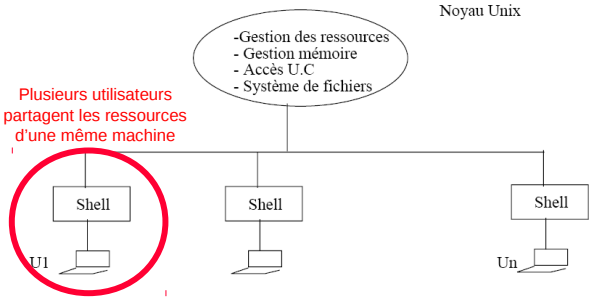


Commande : demande d'exécution d'un programme
- chargement et exécution du code exécutable du programme

Interpréteur de commande : enchaîne l'analyse et l'exécution des commandes

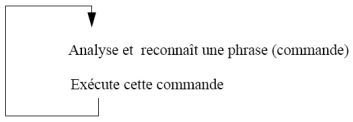
1. Unix : interpréteur shell

- Interpréteur du langage de commande : "shell"

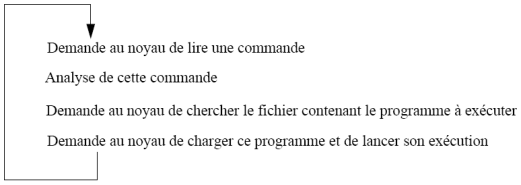


1. Unix : interpréteur shell

- Shell



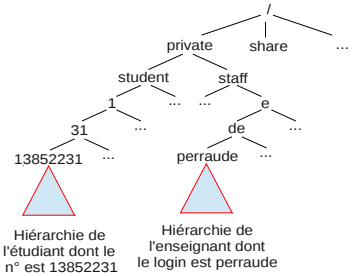
- S'exécute sur le noyau :



I.2 : Comment désigne-t-on les fichiers sous Unix ?

2. Organisation des fichiers sous Unix

- Fichiers organisés en une hiérarchie de catalogues
  - Un catalogue contient des fichiers ou des catalogues
  - Le catalogue / désigne la racine du système de fichier
- Exemple à l'ISTIC :



2. Répertoire maison et répertoire de travail

- Répertoire maison *home directory*
  - Chaque utilisateur dispose d'un répertoire de la hiérarchie appelé *home directory*
  - C'est le nom d'usager qui permet d'affecter le *home directory*
  - Exemple : le nom d'usager **323456** est associé au home directory **/private/student/6/56/323456**
- Répertoire de travail *working directory*
  - C'est le répertoire dans lequel on « travaille »
  - A la connexion, le répertoire de travail est toujours le répertoire maison.
  - On change de répertoire de travail par la commande **cd**

2. Chemin d'accès aux fichiers

- Indique au SGF où trouver un fichier dans la hiérarchie
  - Deux types de chemins existent (absolus, relatifs)
- Chemin d'accès absolu (à partir de la racine /)
  - Pour l'utilisateur n°**2363425** à l'ISTIC le chemin d'accès absolu à son home directory s'écrit

    /private/student/5/25/2363425/
- Chemin d'accès relatif (à partir du *working directory*)
  - On suppose que notre répertoire de travail est **/private/student/5/25/2363425/**
  - **nomfich** désigne le fichier **nomfich** du répertoire **2363425**

    ../3363425/nomfich le fichier du répertoire **3363425**

2. Les liens

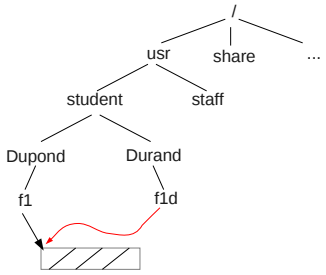
- **Synopsis**
  - **ln [-s] nom1 nom2**
- **Lien matériel**
  - **ln *nom1* *nom2***
  - ***nom1*** désigne un fichier existant
  - ***nom2*** est le nouveau nom permettant de désigner le même fichier
- **Lien symbolique**
  - **ln -s *nom1* *nom2***
  - Utilisé si **nom1** n'appartient pas au même système de fichiers, ou si **nom1** est un répertoire.

2. Les liens : exemple

- Si **Durand** exécute chez lui

  ln /usr/student/Dupond/f1 f1d

  more f1d affiche le contenu de **/usr/student/Dupond/f1**



2. Unix : protection des fichiers

- Les utilisateurs sont regroupés en « groupes »
  - Exemple : un groupe étudiants, un groupe enseignants
- Pour chaque fichier/répertoire, on spécifie des droits
  - Pour le propriétaire, pour le groupe du propriétaire, et pour les autres
- Plusieurs types de droits
  - Pour les fichiers : **r**=*lire*, **w**=*écrire*, **x**=*exécuter*
  - Pour les répertoires : **r**=*lister le contenu*, **w**=*écrire*, **x**=*accès* (cd)
  - On utilise 9 bits pour représenter ces droits d'accès :

r w x	r w x	r w x
- - -	- - -	- - -
propriétaire	groupe	autres

- Représentation simple, qui ne peut pas tout exprimer
  - Par exemple, comment donner les droits de lecture uniquement à son binôme qui appartient au même groupe d'utilisateur ?

2. Unix : protection des fichiers

- Changement de droits : la commande **chmod**

  chmod [qui] op accès [,op accès]* nom
- **qui** : quelle est la classe d'utilisateurs concernée ?
  - **u** : utilisateur (*user*)
  - **g** : groupe (*group*)
  - **o** : autres (*others*)
  - *rien* : tous
- **op** : que fait-on pour cette classe d'utilisateurs ?
  - **+** : ajoute un droit
  - **-** : supprime un droit
  - **=** : définit un nouveau droit
- **accès** : type d'accès (**r,w,x**)

2. Unix : protection des fichiers

- Exemples :

  chmod o-rwx f
  - enlève les droits sur **f** pour les autres

  chmod +x f
  - ajoute le droit d'exécuter **f** pour tous

  chmod g=rx f
  - définit les droits(rx) sur **f** pour le groupe

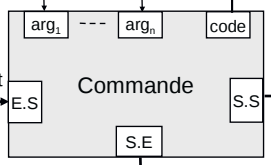
I.3 - Redirection des entrées/sorties

3. Redirection d'entrées-sorties

Échanges d'informations entre la commande et l'interpréteur :

Les arguments de la commande sont les paramètres qui lui passés lors de son appel (ex: **`cat f1 f2 … fn`**)

Le code retour (un entier) est une information retournée à l'issue de l'exécution de la commande



L'entrée standard (E.S) est par défaut lue au clavier.

La sortie standard (S.S) du programme est par défaut affichée sur la console (écran).

La *sortie erreur* (S.E) du programme est par défaut affichée sur la console (écran) ; elle est utilisée par les commandes pour afficher les messages d'erreur

3. Redirection d'entrées-sorties

- Redirection de la *sortie standard*
  - Capturer l'affichage d'une commande (**`prog`**) et le stocker dans un fichier **`fich`** (*redirection sortie standard*).
    - **`prog > fich`** (écrase l'ancien contenu de fich)
    - **`prog >> fich`** (concatène la S.S. avec contenu de fich)
- Redirection de l'*entrée-standard*
  - Utiliser le contenu d'un fichier (au lieu du clavier) comme entrée d'une commande (*redirection entrée standard*).
    - **`prog < fich`**
- Redirection S.S vers E.S (*pipe* en Anglais - *prononcer païpe !*)
  - Utiliser les « résultats » d'une commande comme données d'une autre commande
    - **`prog1 | prog2`**

3. Redirections : exemples

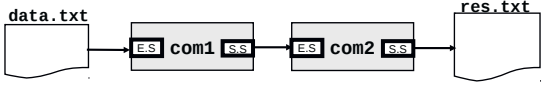
- **`com`**
  - Exécute la commande **`com`**, laquelle lit ses données d'entrée sur le clavier et affiche ses résultats à l'écran
- **`com > res.txt`**
  - Exécute la commande **`com`**, laquelle lit ses données d'entrée sur le clavier et écrit ses résultats dans le fichier **`res.txt`**

- **`com < data.txt`**
  - Exécute la commande **`com`**, laquelle lit ses données d'entrée à partir du fichier **`data.txt`** et affiche ses résultats à l'écran.

3. Redirections : exemples

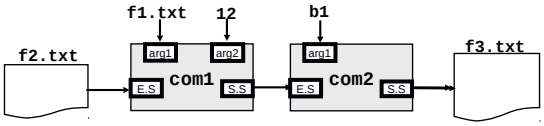
- **`com < data.txt > res.txt`**
  - Exécute la commande **`com`**, laquelle lit ses données d'entrées à partir du fichier **`donnees.txt`** et écrit ses résultats dans le fichier **`res.txt`**.

- **`com1 < data.txt | com2 > res.txt`**
  - Exécute la commande **`com1`**, qui lit ses données dans **`data.txt`** et envoie ses résultats sur l'entrée standard de **`com2`** qui écrit ses résultats dans le fichier **`res.txt`**.



3. Redirections : exemples

- **`com1 f1.txt 12 < f2.txt | com2 b1 > f3.txt`**
  - Exécute la commande **`com1 f1.txt 12`**, qui lit ses données dans **`f2.txt`** et envoie ses résultats sur l'entrée standard de **`com2 b1`** qui écrit ses résultats dans le fichier **`f3.txt`**.



Attention à ne pas mélanger arguments et flux d'entrée-sortie !

I.4 : Quelques commandes de manipulation de fichiers

4. Quelques commandes Unix simples

- Commande **`ls`**
  - **`ls`** : affiche le contenu du WD (*working directory*)
  - **`ls nomfich`** : affiche les caractéristiques du fichier **`nomfich`**
  - **`ls nomrep`** : affiche le contenu du répertoire **`nomrep`**
- Commande **`more`**
  - **`more nomfich`** : affiche le contenu page par page de **`nomfich`** sur la S.S.
  - **`less nomfich`** : idem que pour **`more`** mais permet en plus de « remonter » les pages.
- Création/suppression de répertoires
  - **`mkdir nomrep`** : Crée le répertoire **`nomrep`**
  - **`rmdir nomrep`** : Supprime le répertoire **`nomrep`**
  - **`rm nomfich`** : Supprime le fichier **`nomfich`**

4. La commande `ls`

- Permet d'afficher le contenu d'un répertoire
  - Pour les options faire **`man ls`**

```
[perraude@prshadoko TD]$ ls -ail
total 60
```

26872031	drwxr-xr-x	3	perraude	cairn	4096	10 sept. 12:46	.
26871271	drwxr-xr-x	7	perraude	cairn	4096	10 sept. 14:22	..
26872066	-rwxr-----	1	perraude	cairn	90	16 juil. 16:18	argumentBis.sh
26872057	-rwxr--r--	1	perraude	cairn	105	16 juil. 16:18	argument.sh
26872060	-rw-r-----	1	perraude	cairn	371	16 juil. 16:18	cpmultBis.sh
26872069	-rwxr-----	1	perraude	cairn	198	16 juil. 16:18	cpmult.sh
26872068	-rw-r--r--	1	perraude	cairn	161	16 juil. 16:18	execInter.sh
26872067	-rw-r--r--	1	perraude	cairn	152	16 juil. 16:18	execution.sh
26872033	drwxr-xr-x	6	perraude	cairn	4096	16 juil. 16:18	RACINE
26872059	-rwxr--r--	1	perraude	cairn	113	16 juil. 16:18	recherche1.sh
26872032	-rwxr--r--	1	perraude	cairn	222	16 juil. 16:18	recherche2.sh
26872065	-rwxr--r--	1	perraude	cairn	289	16 juil. 16:18	recherche3.sh
26872061	-rwxr-xr-x	1	perraude	cairn	5169	16 juil. 16:18	renverseChaine
26872064	-rw-r-----	1	perraude	cairn	378	16 juil. 16:18	renverseChaine.c

n° inode — droits — propriétaire — groupe — taille — dernière modification — nom

4. Quelques commandes simples

- **Copie de fichiers**
  - `cp nomsrc nomdst`
    - Recopie le fichier **nomsrc** dans **nomdst**
  - `cp nomsrc nomrep`
    - Recopie le fichier **nomsrc** dans le répertoire **nomrep**
- **Copie de répertoire**
  - On utilise **cp** avec l'option **-r**, pour recopier toute une hiérarchie (c'est-à-dire le répertoire et tous ses sous répertoires)

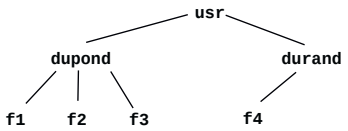
```
cp –r monrepsrc/ monrepdst/
```


4. Quelques commandes simples

- **La commande `cat`**
  - `cat f1 ... fn`
    - lit les fichiers **f1 ... fn** et les affiche sur la S.S
  - `cat`
    - affiche sur la sortie standard, l'entrée standard frappée au clavier avec ctrl+D pour marquer la fin de fichier.
- **La commande `echo`**
  - `echo "P"` : affiche **P** sur la S.S

Exercice 1

- Soit la hiérarchie suivante :



- L'utilisateur **durand** se connecte chez lui.
- En utilisant **cat** :
  - Créer un fichier **f5** qui contient
    5 2 3 -1
  - Créer un fichier **f6** contenant **f5** suivi de **f4**
  - Rajouter le fichier **f1** de Dupond à **f6**
  - Recopier **f2** de Dupond dans **f7** (de Durand)

4. Les caractères spéciaux

- Délimiteur **'…'**
  - Protège les caractères de toute interprétation par le Shell
- Délimiteur **"…"**
  - Protège les caractères de toute interprétation par le shell, sauf les caractères **$** et `
- Délimiteur **`…`**
  - Exécute la commande **…** et remplace **`…`** par ce qui est produit sur la sortie standard par cette commande
  - Exemple : répertoire courant = **/usr/dupond**

`echo pwd`	affiche **pwd**
`echo `pwd``	affiche **/usr/dupond/**

4. Les caractères spéciaux

- Le caractère *****
  - Remplace une suite de caractères quelconque
  - Exemple : la commande **ls *.java** affiche tous les fichiers d'extension java présents dans le répertoire courant.
- Le caractère **?**
  - Remplace un caractère quelconque
  - Exemple : la commande **ls toto?.java** affiche tous les fichiers d'extension **java** dont le nom occupe 5 caractères et qui commence par les caractères **toto**.
- Le caractère ****
  - considère le caractère suivant comme un caractère quelconque
  - permet (entre autre) d'annuler le retour chariot en fin de ligne
  - Si je veux afficher **$a** à l'écran : **echo \$a**

II : Programmation *shell*

Plan du chapitre

1. Création et appel d'un programme (*script*) shell
2. Quelques commandes supplémentaires
3. Affectation et référence de variables
4. Structures de contrôle (boucle, conditionnelle)
5. La commande grep
6. La commande find

1. Programmation shell

- Il existe de nombreux interpréteurs de commandes
  - **bash, csh, zsh, ksh, …**
  - Ils se ressemblent tous + ou - …
- Dans le cadre de ce cours, nous allons utiliser **sh**
  - Connu sous le nom de bourne shell
  - Conçu pour Unix par Stephen Bourne en 1977
  - Premier programme *shell* disponible sous Unix
  - Un des plus utilisés (disponible sur tous les systèmes)
- Intérêt de la programmation *shell*
  - Être capable de programmer rapidement des opérations relativement complexes impliquant des commandes UNIX.
  - Permet d'automatiser de nombreuses tâches liées à la maintenance/utilisation d'un système.

1. Création d'un programme shell

- Première approche :
  - On crée un fichier texte **`nomscript`** dans lequel on écrit les commandes à exécuter.
  - On lance le script shell par la commande
    `sh nomscript arguments`
  - L'exécution va créer un nouveau processus « shell **» .**
    - **Attention, ceci peut avoir des conséquences en particulier sur les variables**
- Seconde approche :
  - On crée un fichier texte **`nomscript`** dans lequel on place au début du fichier la ligne **`#!/bin/sh`** qui sera suivie des commandes.
  - Une fois donné à **`nomscript`** les droits d'exécution**,** on peut directement l'exécuter en écrivant :
    `./nomscript arguments`

1. Paramètres/arguments

- Possibilité de passer des arguments lors de l'appel :
  - **`nomscript arg1 arg2 … argn`**
- On y fait référence à l'aide des identificateurs :
  - **`$#`** = nombre de paramètres passés au programme shell
  - **`$*`** ou **`$@`** = tous les paramètres
  - **`$0`** = nom du programme
  - **`$1`** = premier paramètre
  - **`$2`** = deuxième paramètre
- Exemple

nomscript
```
#!/usr/bin/sh
echo "$# arguments en paramètres"
echo "argument n°1 =$1"
echo "argument n°2 =$2"
echo "argument n°3 =$3"
```

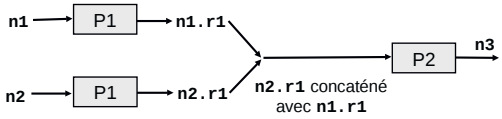
```
%nomscript toto titi tata
3 arguments en paramètres
argument n°1 =toto
argument n°2 =titi
argument n°3 =tata
%
```


Exercice 2

Faire un programme de commande

`execution n1 n2 n3`

Qui réalise les traitements illustrés ci-dessous



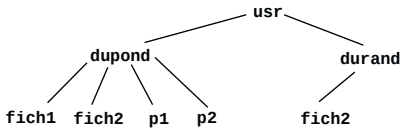
P1 et P2 sont des programmes lisant sur l'ES et produisant sur la SS.
Les paramètres des commandes sont des fichiers (ici n1,n2,n3)

Exercice 2 (suite)

Que se passe-t-il sur l'appel

`execution fich1 ../durand/fich2 fichr`

si **dupond** l'exécute dans son *home directory* avec :



II.2 : Quelques commandes supplémentaires

2. Les commandes `basename` et `dirname`

- Utilisation:
  - **`basename`** *nom* : fournit le *nom* sans le répertoire
  - **`dirname`** *nom* : fournit la partie répertoire
- Exemple :
  - Si on veut dans l'exemple précédent que tous les fichiers créés le soient dans le WD:

```
p1  < $1 > `basename $1`.r1

p1  < $2 > `basename $2`.r1

cat `basename $1`.r1 `basename $2`.r1
    | p2 > `basename $3 `
```


2. Commandes internes

Ce sont des commandes prédéfinies. Exemples:

- **`pwd`** : affiche sur S.S. le nom du WD
- **`cd`** : changement de répertoire
- **`exit`** : pour quitter le programme de commande avec en code retour celui de la dernière commande exécutée
- **`echo`** : affiche sur S.S. les arguments de la commande
- …

II.3 : Affectation et référence de variables

3. Affectation et référence de variables

- Affectation : **`nomvar =`** *expression*
  - **Attention** : pas d'espace autour de =
  - *expression* = **`valeur`** **`(1)`**
  - *expression = référence à une variable* **`(2)`**
  - *expression* = **`` `commande` ``** **`(3)`**

  `(3)``` `commande` `` est remplacé par ce qui est produit sur la S.S
- Référence : **`$nomvar`**
  - Exemple 1 : affichage du répertoire courant

```
y=`pwd`
echo $y
```

  - Exemple 2 : affichage de deux arguments du programme shell

```
y="$1 et $2"
echo $y
```


3. Variables : affectation par lecture

- On utilise la commande **`read`**

```
read nomvar1 nomvar2
```

  - Lit une ligne sur l'entrée standard et affecte la première valeur à **`nomvar1`** la seconde à **`nomvar2`**, etc.
  - Le code retour est 0 (vrai) si la lecture s'est bien passée
  - Si lors de l'exécution de read on rencontre EOF (CTRL+D) le code de retour de **`read`** est ≠ 0 (faux)

Exercice 2 (suite)

- Écrire une version de **`execution`** interactive, c-à-d qui demande n1, n2 et n3 à l'utilisateur.

  Après un message d'invite, on lit les noms de fichiers effectifs qui ne sont donc plus passés en paramètres

3. Variables prédéfinies du shell

`#`	: Nombre de paramètres d'une commande
`HOME`	: *Home directory*
`PATH`	: Chemin d'accès pour les **commandes**
`PS1`	: Chaîne du prompt

…

Commentaire

```
# Ceci est un commentaire RC
```


II.4 : Structures de contrôle (boucle, conditionnelle)

4. Structures de contrôle : la conditionnelle

- Syntaxe

```
if
   Suite de commandes n°1
then
   Suite de commandes n°2
else
   Suite de commandes n°3
fi
```

facultatif

- Fonctionnement
  - Si la dernière commande de liste commande n°1 rend 0, on exécute liste de commande N°2,
  - Sinon on exécute la liste de commande n°3

NB: **`exit`** fait sortir du programme de commande courant

4. Structures de contrôle : les tests

- On peut tester des situations grâce à la fonction **`test`** et utiliser le résultat dans la partie "**`if`**"

  `test` *expression* ou **`[`** *expression* **`]`**

  Attention: espace obligatoire autour des crochets
- Il existe plusieurs types d'expressions pour
  - les fichiers (existence, type, droits, etc.)
  - les chaînes de caractères (ordre lexicographique)
  - les nombres (comparaisons)
- Ces expressions peuvent être combinées avec
  - **`!`** : négation
  - **`-a`** : et logique
  - **`-o`** : ou logique

4. Structures de contrôle : tests de fichiers

- Tests sur les fichiers/répertoires
  - **`-d nomrep`** : rend vrai si le répertoire **`nomrep`** existe
  - **`-f nomfich`** : rend vrai si le fichier **`nomfich`** existe
  - **`-s nomfich`** : rend vrai si **`nomfich`** existe et n'est pas vide
  - **`-r nomfich`** : rend vrai si **`nomfich`** existe et peut être lu
  - **`-w nomfich`** : rend vrai si **`nomfich`** existe et peut être modifié
  - **`-x nomfich`** : rend vrai si **`nomfich`** existe et peut être exécuté
- Tests sur les chaînes
  - **`-z ch1`** : rend vrai si longueur(*ch1*) = 0
  - **`-n ch1`** : rend vrai si la longueur (*ch1*) ≠ 0
  - **`ch1 = ch2`** : rend vrai si *ch1=ch2 (ch1,ch2* sont des chaînes)
  - **`ch1 != ch2`** : rend vrai si *ch1≠ch2 (ch1,ch2* sont des chaînes)

4. Structures de contrôle : les tests

- Tests arithmétiques (ou lexicographiques)
  - ***n1* `-eq` *n2*** : rend vrai si *n1* = *n2*
  - ***n1* `-ne` *n2*** : rend vrai si *n1* ≠ *n2*
  - ***n1* `-gt` *n2*** : rend vrai si *n1* > *n2*
  - ***n1* `-lt` *n2*** : rend vrai si *n1* < *n2*
  - **`n1 -ge n2`** : rend vrai si *n1* ≥ *n2*
  - ***n1* `-le` *n2*** : rend vrai si *n1* ≤ *n2*
- Exemple :

nomscript

```
#!/usr/bin/sh
if [ !-f $1 ]
then
echo “$1 n’existe pas”
exit
fi
echo “bienvenue“
```

```
% nomscript toto
bienvenue
% rm toto
% nomscript toto
toto n’existe pas
%
```


Exemple : test dans exec non interactif

```
if [ $# -ne 3 ]
then
  echo “appel n1 n2 n3”
  exit
fi
```
Vérifie que le nombre d’arguments est bien 3

```
if [ ! -f $1 ]
then
  echo “$1 n’existe pas”
  exit
fi
```
Vérifie l’existence du fichier passé en 1er paramètre

```
if [ ! -f $2 ]
then
  echo “$2 n’existe pas”
  exit
fi
```
Vérifie l’existence du fichier passé en 2ème paramètre

```
p1 < $1 >‘basename $1‘.r1
p1 < $2 > ‘basename $2‘.r1
cat ‘basename $1‘.r1 ‘basename $2‘.r1 | p2 >‘basename $3‘
```


4. Structures de contrôle : itération

- Syntaxe

```
for variable
{ in liste de valeurs }
do
  liste de commandes
done
```

- Exemple :
  - un programme de commande qui recopie 3 fichiers du répertoire courant (f1, f2, f3) dans un autre répertoire passé en paramètre

```
for fichier
in f1 f2 f3
do
cp $fichier $1/$fichier
done
```


Exercice 4

- Reprendre l’exemple précédent en
  - Donnant les noms des fichiers en paramètres
  - En lisant le nom du répertoire
- Si le fichier de commande se nomme **`cpmult`**, l’appel

  `cpmult f1 f2 f3`

  commencera par afficher l’invite:

  `Nom du répertoire destinataire:`
- Même exercice en demandant à l’utilisateur, si le fichier à copier existe déjà avec le même nom dans le répertoire destinataire, quelle action effectuer.

4. Structures de contrôle : itération

- Syntaxe

```
while
  liste de commandes 1
do
  liste de commandes 2
done
```

- Principe de fonctionnement
  - exécute la liste de commande n°1
  - si code de retour (dernière commande de cette liste) =0 alors
    - exécute “liste commande 2” et on recommence”
  - finsi

II.5 : La commande grep

5. La commande `grep` : description

- Synopsis :

  `grep [options] MOTIF [FILE...]`
- Description :
  - Commande permettant de rechercher des motifs de caractères dans des fichiers.
  - Le motif peut-être exprimé comme une expression régulière
  - La commande rend 0 si motif trouvé et une valeur !=0 sinon
- Format utilisé pour exprimer les motifs
  - Motif = expression régulière
  - Le caractère . : n’importe quel caractère
  - Le caractère * : répétition du motif qui précède * dans l’exp. reg.
    - ATTENTION : différent avec le * utilisé pour désigner des fichiers.
  - [*x*-*y*] : intervalle de caractères (ex : [a-z], [0-9], etc.)

5. La commande `grep` : exemples

`grep mot fich`
- affiche sur la S.S toutes les lignes de **`fich`** qui contiennent la chaîne **`“mot”`**

`grep –r mot ./`
- affiche sur la S.S toutes les lignes des fichiers du répertoire de travail (et des sous répertoires) contenant la chaîne **`“mot”`**

`grep ”m.*t” fich`
- affiche sur la S.S les lignes de **`fich`** qui contiennent une chaîne commençant par **`”m”`** et se terminant par **`”t”`**

`grep ”toto[1-3]” fich`
- affiche sur la S.S les lignes de **`fich`** qui contiennent une chaîne de 5 caractères commençant par **`”toto”`** et terminée par un chiffre entre 1 et 3**.**

Exercice 5

- Écrire un programme de commande
  - (p2 est un fichier, p1 un motif)

  `recherche1 p1 p2`

  qui affiche : “p2 contient p1” ou “p2 ne contient pas p1”

- Idem mais **`recherche2 p`** demande (par lecture sur E.S.) les chaînes à rechercher dans **`p`** jusqu’à la rencontre du motif « **`fin`** ».

- Idem mais pour une même suite de motifs appliquée à une liste de fichiers en paramètre

  `recherche3 f1 ... fn`

- NB : **`/dev/null`** = poubelle

II.6 : La commande find

6. La commande `find`

- Synopsis :

  `find dir [critères]`
- Fonctionnement :
  - La commande **`find`** permet de retrouver des fichiers à partir de certains critères (nom, taille, propriétaire, etc.).
  - La recherche des fichiers se fait dans le (ou les hiérarchies de) répertoire(s) passé(s) en argument à la commande.
  - Par défaut, la commande affiche sur la S.S. les chemins d’accès des fichiers retrouvés.
  - Il est possible d’appliquer une commande à tous les fichiers retrouvés.

6. La commande `find` : sélection

- Quelques critères de sélection
  - **`-name`** ***`nom`*** : par nom de fichier (possible d’utiliser ‘**`*`**‘),
  - **`-usr`** ***`user`*** : par propriétaire,
  - **`-size`** ***`taille`*** [c|k|M] : par taille de fichier (octet, ko, Mo), etc.
  - **`-type`** [**`d|f|l`**] : par type (directory,file,link)
  - **`-mtime j`** : fichiers modifiés il y a moins de **`j`** jours.
  - …
- Il est possible de combiner ces critères
  - **`-not`** : négation de l’expression qui suit
  - **`\(`** : début d’une expression de sélection.
  - **`\)`** : fin d’une expression de sélection.
  - ***`-o`*** : ou logique entre deux expressions.
  - ***`-a`*** : et logique entre deux expressions.

6. La commande `find` : exemples

`find /usr -name “*.c ”`
- Affiche sur la S.S le chemin d’accès de tous les fichiers se terminant par .c dans la hiérarchie **`/usr/`** :

`find /usr -usr dupond`
- Affiche sur la S.S le chemin d’accès de tous les fichiers de la hiérarchie **`/usr/`** dont le propriétaire est l’utilisateur **`dupond`**.

`find . -not \( -name ”*.obj” -o -name “*.map” \)`
- Affiche sur la S.S. les chemins d’accès de tous les fichiers de la sous hiérarchie du *working dir* dont l’extension n’est ni **`.obj`**, ni **`.map`**.

`find . \( -usr dupond -a -name “*.c” \)`
- Affiche sur la S.S. les chemins d’accès de tous les fichiers (de la sous hiérarchie du *working dir*) appartenant à **`dupond`** et dont l’extension est **`.c`**.

6. La commande `find` : l’option `-exec`

- Plutôt que d’afficher le nom des fichiers trouvés sur la S.S, on peut leur appliquer une commande
  - Permet d’appliquer une même opération sur un ensemble de fichiers répondant à des critères précis.
- On utilise pour cela l’option **`-exec cmd {} \;`**
  - Applique la commande **`cmd`** à chaque fichier trouvé en substituant la chaîne **`{}`** par le nom du fichier.
  - La séquence **`\;`** sert à marquer la fin de la commande.
- Exemples:

  `find . -name “*.java” -exec lpr {} \;`
  - Appliquer la commande **`lpr`** (impression de fichier) à tous les fichiers **`*.java`** de la hiérarchie du répertoire **`courant`**.

Exercice

- Indiquer l’effet des commandes suivantes :
  - **`find ./ -name “tp2” -exec {} < f_in > {}.out \;`**
  - **`find /var/log -exec grep "www.athabasca.com" {} \;`**
  - **`find . -name "rc.conf" -exec chmod o+r {} \;`**
  - **`find /usr/local -type f -name "*.html"`**