Unix: Les Permissions

Utilisateurs et Groupes

- UNIX est un système d'exploitation Multi-Utilisateur
- =>
  - Plus qu'un utilisateur peut interagir avec (se loguer) le système a chaque instant.
  - Chaque utilisateur a un ensemble séparé de privilèges d'accès pour les ressources système (e.g. les fichiers)
  - On définit deux notions pour identifier et classifier les utilisateur:
    - '*user*' et '*group*'

Système d'exploitation	Utilisateurs Simultanés?	privilèges Séparés?
UNIX	Y	Y
Windows 2000/XP	Y	Y
Windows NT	N	Y
Windows 95/98	N	N

Utilisateurs et Groupes (cont.)

- Un utilisateur ou '*user*' est un ID système qui permet a chaque utilisateur qui se logue de s'identifier pour une question d'accès aux ressources.
- Quand un ensemble d'utilisateurs ont besoin d'un accès similaire a une ressource, ils peuvent faire partie d'un '*group*', et le groupe est octroyé avec ce privilège d'accès.
- le tableau suivant présente quelques commandes liées a la sécurité:

Command	Objectif
`who am i`	Affiche les informations du login
`id`	Affiche les informations sur l'utilisateur et le groupe courrant
`su` *`username`*	Temporairement exécute un shell comme un autre utilisateur
`useradd`	Ajouter un utilisateur
`groupadd`	Ajouter un nouveau groupe
`newgrp` *`groupname`*	Change le groupe courrant

Sécurité des fichiers

- Les informations qui suivent sont liées a la notion de sécurité et sont définies pour chaque fichier/répertoire dans le système de fichiers UNIX :
  - Le propriétaire du fichier (un nom d'utilisateur)
  - Le groupes propriétaire de ce fichier (un nom de groupe)
  - Les *permissions d'accès a ce fichier pour les différents classes d'utilisateurs*
- Il y a trois ensembles de permissions:
  1. Les privilèges d'accès pour le propriétaire du fichier
  2. Les privilèges d'accès pour l'un des membres du *groupe du fichier*
  3. Les privilèges d'accès pour tous les autres.
- Seulement le propriétaire du fichier a le droit de modifier ces privilèges
- Toutes ces détails sont affichés par la commande **`ls -l`**

Protection des fichiers (cont.)

- Les trois caractères de permission (`r`, `w`, et `x`):

Permission	Pour les *Fichiers*	Pour les *Dossiers*
`r`	lire	Voir le contenu (e.g. `ls`)
`w`	modifier (écrire)	créer ou effacer fichiers
`x`	exécuter	accéder (e.g. `cd`) ou faire partie d'un chemin

Modification des Permissions

- La commande utilisée pour modifier les permissions des fichiers est `chmod`.
- Il y a deux méthodes d'utiliser `chmod`:
  1. Mode symbolique
  2. Mode Numérique
- Les deux méthodes ont la forme suivante:

  `chmod` ***`permissions filename(s)`***

  Seulement les ***`permissions`*** différencie les méthodes
- *Le mode symbolique* est utilisé en général par les débutants.

chmod Mode Symbolique

- en mode symbolique, les permissions sont spécifiées en utilisant les lettres, comme suit:
  - `chmod u+w fichier1` donne le droit d'écriture au propriétaire
  - `chmod g-r fichier1` enlève le droit de lecture au groupe
- les lettres utilisées :
  - ***Chmod*** personnes action permissions

Personnes		Action		Permissions	
u	propriétaire	+	ajoute	r	lecture
g	groupe	-	retire	w	écriture
o	autres	=	positionne	x	exécution
a	all				

chmod Mode Symbolique (cont.)

- Les permissions multiple peuvent être spécifie comme suit:
  - `chmod uo+w,u-rx fichier1`
    donne au propriétaire et aux autre le droit d'écriture et enlève le droit de lecture et d'exécution a l'*utilisateur*
- Pour créer des permissions complexe,ce mode est onéreux

chmod Mode Numérique

- En mode numérique ou octal, les permissions sont spécifies on utilisant trois nombres.
- Toutes les permissions sont affectées en une commande
- Si on définit `r=4`, `w=2` et `x=1`, on peut positionner les permissions pour les trois types d'utilisateurs (propriétaire, groupe et les autres) on ajoutant les numéros correspondant aux permissions volues
- Par exemple:
  - `chmod 640 file1` `rw-r-----`
    donner au proprietaire les permissions de lecture et ecriture, le groupe la permission de lecture, et pas de permission pour les autres
  - `chmod 070 file1` `---rwx---`
    donne toutes les permissions au *groupe et seulement pour le groupe*

Modification de propriétaire

- On peut changer le propriétaire d'un fichier en utilisant la commande chown :

  `chown` *`propriétaire fichier(s)`*

- Exemple:

  `chown usr01 *.doc`

- N.B:
  - La notion de propriété d'un fichier n a rien avoir avec son emplacement. Autrement, un fichier peut appartenir a X mais réside dans le répertoire personnel de Y.
  - Vous ne pouvez pas changer les permission d'un fichier qui ne vous appartient pas, si vous faite des modifications sur un fichier, changer l'appartenance en dernier lieu
  - Si vous changez le propriétaire d'un fichier vous ne pouvez plus revenir en arrière

Changing File Group

- On peut changer le propriétaire d'un fichier en utilisant la commande `chgrp` :

  `chgrp` *`groupe fichier(s)`*

- Exemple:

  ```
  chgrp commercial donnees0 donnees02
  ```

- N.B:
  - Si vous ete membre d'un group, et ce groupe a des permissions sur un fichier, si on voulant accéder a ce fichier avec cette permission vous avez un problème «'permission non permise', utiliser la commande `newgrp` pour changer votre groupe courrant

chmod Mode Numérique (cont.)

- Malgré qu'il y aune centaine de combinaisons légales, seulement quelques unes sont usuellement utilisées:

	chmod	ls
Fichiers de données	444	r--r--r--
	644	rw-r--r--
	664	rw-rw-r--
	666	rw-rw-rw-
Programmes	750	rwxr-x---
	755	rwxr-xr-x
	777	rwxrwxrwx
Répertoires	755	rwxr-xr-x
	775	rwxrwxr-x
	777	rwxrwxrwx

Permission par défaut

- Tout fichier crée prend des permissions par défaut. Ces permission sont définies dans un mask de création.
- Ce mask peut être modifier par la commande ***umask*** comme suite:
  - umask *permissions*
- Ou permissions sont similaires aux permission de chmod en mode numerique sauf que les position avec un un designent les permission qui doivent avoir comme valeur zero et vice versa.

- la commande ***umask*** est utilisee pour que tout fichier cree dans le shell courant aient les permissions specifies par le mak defini.

- Exemple:
  - umask 022
- les fichiers crees apres cette commande auront comme permissions: **644 (rw-r--r--) = 666 - 022** (normalement c'est: **777-022=755** mais la permission "execute" est mis a zero qlq soit le msk)

- Pour les repertoire les permissions sont : **755 (rwxr-xr-x)**

Un dangereux trou de Sécurité

- Supposer qu'on a les permissions:
  ```
  drwxrwxrwx ... rep1
  -r--r--r-- ... rep1/fich1
  ```
- En peut modifier `fich1` en exécutant les étapes suivantes:
  1. `cp fich1 fich2`
  2. Modifier `fich2`
  3. `rm fich1` (c'est possible parce que `rep1` est a la permission d'ecriture)
  4. `mv fich2 fich1`
  5. Changer les atributs pour que le fichier ressemble au fichier original `fich1`
- NB: il faut toujours faire attention aux permissions des repertoires

Combinaison de commande

- Redirection, tubes et Filtres

Sortie standard et redirection

- La plus part des programmes UNIX en exécution produisent des résultats en sorties
  - De telles sorties finissent en général dans l'écran de l'utilisateur
- La sortie peut être rediriger vers l'un des deux "places":
  1. Un fichier
  2. Un autre programme en exécution (processus)
- pour rediriger la sortie vers un *fichier*, on utilise le symbole ">"
- Par exemple:

```
ls –l > fichier.txt
```

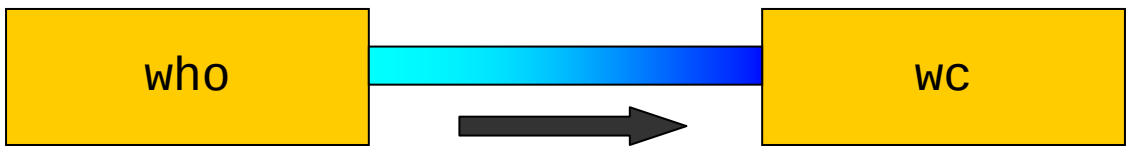
Le fichier `fichier.txt` est crée dans le répertoire courant et contiendra la sortie de la commande `ls`

Sortie Standard et redirection (cont.)

- Pour rediriger la sortie vers un autre programme, le "|" ("pipe") symbole est utilisé.
- Par exemple:

```
who | wc
```

- Quand la commande est entrée en ligne de commande, le shell fait les opérations suivantes:
  - Lance la commande `who`
  - Lance la commande `wc`
  - Connecte les deux d'une façon telle que la sortie du premier programme est redirigée vers l'entrée du second programme



Entrée Standard et redirection

- Le diagramme précédant décrit comment des processus peuvent avoir des entrées.
- Tous les processus non pas toujours des entrées
- Par défaut, les entrées d'un programme vient du clavier
- Une utilisation approprie du shell peut laisser un programme lire les entrées de l'une des deux places suivant au lieu du clavier:
  1. Un fichier
  2. D'un autre processus
- L'entrée est lue d'un fichier en utilisant le symbole "<"
- Par exemple:

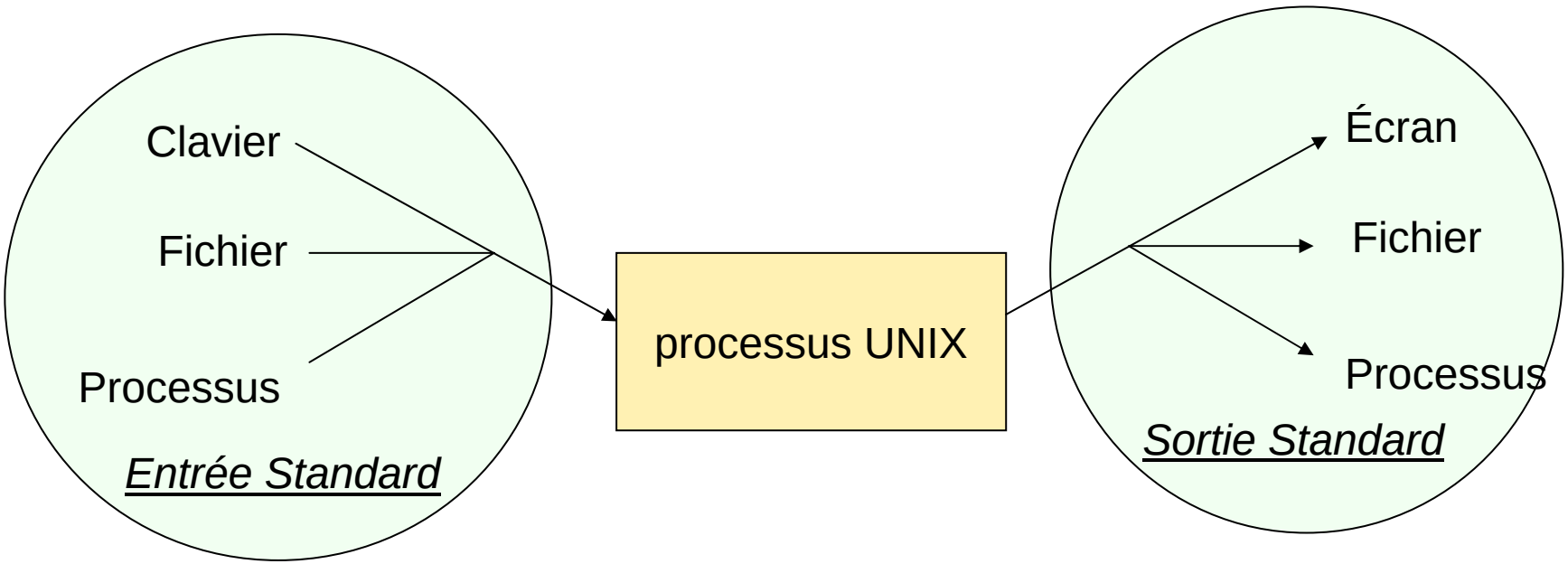
```
wc < fichier
```

Examples

- cat > fichier.txt
  - Entrer le texte et terminer par CTRL d
- cat fichier
- ls –l > liste (ecraser le contenu)
- ls –l >> liste (ajouter a la fin du fichier)
- cat file | sort | uniq
- sort –u < fichier > fichier.trie
- wc –l < liste
- sort –u < liste | wc -l

Entrée et Sortie Standards

- Les entrées et sorties décrites ci-dessous sont connues sous le nom d'entrée standard et sortie standard:

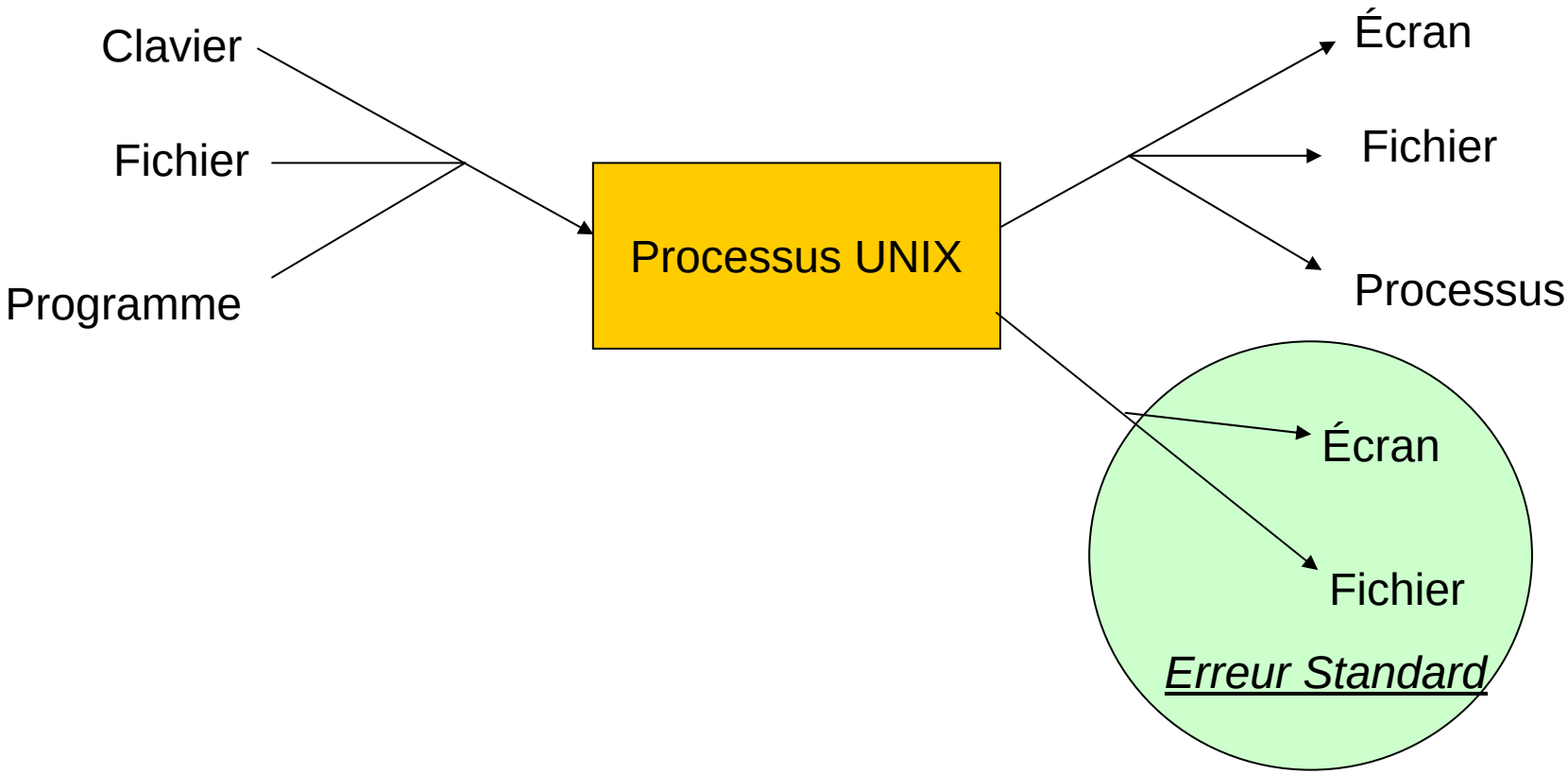


Entrée et Sortie Standards (cont.)

- Qu’il est le besoin pour faire ces enchaînement ? Pourquoi connecter deux processus ensemble, ou lire de fichiers?
- La plus part des utilitaires en ligne de commande qui vient avec Unix adhère a la philosophie que des taches "complexe" peuvent être faites en combinant de simples commandes.
- Cet approche a fait d’UNIX un système très performant, et c’est le fondement de la programmation shell

Erreur Standard

- Chaque programme produit deux sorte de sortie:



Erreur Standard (cont.)

- Pour rediriger l'Erreur Standard vers un fichier, avec "2>"
  - (la Sortie Standard peut aussi être rediriger avec "1>")
- On peut rediriger la sortie d'une commande vers deux fichiers séparés, si nécessaire par:

```
commande1 > fichierA 2> fichierB
```

- Il est possible de rediriger toutes les sortie vers la meme place, comme suit:

```
commande1 > fichierA 2>&1
```

- On peut rediriger une sortie, s'il n'est pas souhaitee, vers `/dev/null`.

Flux d’entrees sorties

- stdin (0)
  - Par défaut entrée du clavier
- stdout (1)
  - Par défaut sortie vers l’écran
- stderr (2)
  - Messages d’erreurs s’affichent sur l’écran

Les Filtres

- Un *filtre est un utilitaire* UNIX en ligne de commande qui a les propriétés suivantes:
  1. Il lit de l'entrée standard
  2. Il fait des traitements sur les données qu'il a lu
  3. Il produit des sorties basées sur ces entrées
- Par exemple, `wc` est un filtre. Le traitement qu'il fait est le comptage des lignes, des mots et des caractères. `ls`, par exemple, n'est pas un filtre (il n'a pas d'entrée)
- Les filtres sont utilisés pour traiter les données produites par un autre processus ou les données d'un fichier

Les filtres les plus utilisés

- Les commandes qui suivent sont des filtres utilisées couramment en UNIX:

Filtre	Traitement fait sur les entrées
`cat`	Pas de traitement (affichage)
`more`	Pagination
`grep`	Sélectionner seulement les lignes contenant certain texte
`sort`	Tri
`wc`	Comptage des lignes, mots et/ou caractères
`tee`	Duplication – écrit dans des fichiers et l'écran
`sed`	Édition de base
`awk`	boucoup de chose

Recherche de texte dans les fichiers

- grep:
  - `grep est utilisée pour chercher du texte dans des fichiers ou lu a partir de l'entrée standard.`
  - `grep` est un vrai filtre, dans tous les sens du mot.
- L'utilisation de grep est comme suit:

  `grep` *`motif nom-fichier(s)`*
- Exemple:

  `grep UNIX *.txt`
- `grep` utilises son propre ensemble d'expressions régulières
- Utiliser `grep` avec `find` pour trouver des fichier dans tous les dossiers.

egrep

- Il existe deux versions :
  - grep qui supporte des expressions régulières de base
  - egrep supporte les expressions régulières étendues
- La commande egrep supporte toutes les options de grep

egrep

- Recherche dans des fichiers
  - D'un chaîne ou d'une sous chaîne de caractères
  - Simplement d'un mot
  - D'une chaîne *formalisée* par une expression régulière
- Utilisation et informations
  - **egrep [options] <chaîne recherchée> <fichier>**
  - **man egrep**
- Résultat
  - Lignes du fichier contenant ce qui est recherché
  - Ou autre résultat, suivant les options utilisées

Egrep Recherche d'une Chaîne

- egrep <chaîne> fichier

We have retained the relatively modest mathematical level of the first two editions. We have found that engineering students who have completed one or two semesters of calculus should have no difficulty reading almost all of the text.

- egrep two statistics.txt

Egrep Recherche d'une Chaîne

- egrep <chaîne> fichier

We have retained the relatively modest mathematical level of the first two editions. We have found that engineering students who have completed one or two semesters of calculus should have no difficulty reading almost all of the text.

- egrep st statistics.txt

Egrep Recherche *Inversée*

- egrep –v <chaîne> fichier

We have retained the relatively modest mathematical level of the first two editions. We have found that engineering students who have completed one or two semesters of calculus should have no difficulty reading almost all of the text.

- egrep –v two statistics.txt

Egrep Recherche d'un Mot Exact

- egrep –w <chaîne> fichier

We have retained the relatively modest mathematical level of the first two editions. We have found that engineering students who have completed one or two semesters of calculus should have no difficulty reading almost all of the text.

- egrep –w the statistics.txt

Egrep Lignes de Contexte

- egrep –<nombre de lignes> <chaîne> fichier

*We have retained the relatively modest mathematical level of the first two editions. We have found that engineering students who have completed one *or two semesters of calculus should have no difficulty reading almost all of the text.

- egrep –1 found statistics.txt

Egrep Numéro de Ligne

- egrep –n <chaîne> fichier

We have retained the relatively modest mathematical level of the first two editions. We have found that engineering students who have completed one or two semesters of calculus should have no difficulty reading almost all of the text.

- egrep –n two statistics.txt

Egrep nombre d’occurrence

- egrep –c <chaîne> fichier

We have retained the relatively modest mathematical level of the first two editions. We have found that engineering students who have completed one or two semesters of calculus should have no difficulty reading almost all of the text.

- egrep –c two statistics.txt

Egrep Recherche d'une Chaîne

- egrep –i <chaîne> fichier

We have retained the relatively modest mathematical level of the first two editions. We have found that engineering students who have completed one or two semesters of calculus should have no difficulty reading almost all of the text.

- egrep –i two statistics.txt

Expressions Régulière Introduction

- Définition
  - Formule qui représente une chaîne de caractères
  - Composée de caractères et d'opérateurs
- Utilisation
  - On recherche alors non pas un mot ou une simple chaîne de caractères mais une suite de caractères qui correspondent au critères énoncés par la formule
  - Certains opérateurs doivent être précédés d'un \ pour ne pas entrer en conflit avec le shell, ainsi : {, }, <, >, (,) et | seront écrits \{, \}, \<, \>, \(, \) et \|. C'est aussi le cas de l'espace.
  - Ou mettre l'expression entre cotes.
  - Si on veut chercher des caractères comme * il faut les précédés par \ .

Expressions Régulière Opérateurs

- Comment représenter
  - Un caractère quelconque : **.**
  - Une ou une infinité d'occurrences : **+**
  - Zéro ou une infinité d'occurrences : *****
  - Un choix parmi un ensemble : **[<liste>]**
  - Tout sauf un certain caractère : **[^< caractère >]**
- Évidement, on peut combiner les expressions
  - **.*** : Zéro ou une infinité de caractères quelconques
  - **a+b*** : Au moins un 'a' suivi de 0 ou une infinité de 'b'
  - **[ab]+** : Au moins un 'a' ou un 'b' ou une infinité
  - etc.

Expressions Régulière Utilisation

- Exemples
  - **v.+** : Les chaînes contenant un v suivi de n'importe quelle suite de caractères
    - valable
    - vital
    - svelte
  - **[vr].+** : Les chaînes contenant un 'v' ou un ‘r' suivi de n'importe quelle suite de caractères
    - valable
    - vital
    - repas
    - rival

Expressions Régulière Utilisation

- Exemples
  - **a.*a** : Les chaînes contenant au moins deux 'a'
    - valable
    - salade
    - sultanat
  - **[ps].*a.*a** : Les chaînes contenant un 'p' ou un 's‘ suivi d'une sous chaîne contenant deux 'a‘ au moins
    - salade
    - patate
    - Apprentissage primaire

Expressions Régulière opérateurs

- Comment représenter
  - Un caractère compris entre 'a' et 'z' : **[a-z]**
  - Un caractère compris entre '0' et '9' : **[0-9]**
  - Le début d'une ligne : **^**
  - La fin d'une ligne : **$**
  - Un choix entre deux chaînes : **\|**
  - Et en combinant avec le choix
    - Caractère compris entre 'a' et 'z' ou 'A' et 'Z' : **[a-zA-Z]**
- Répétitions d'une occurrence
  - Exactement **2** répétitions de 'x' : **x\{2\}**
  - Entre **2** et **5** répétitions de 'x' : **x\{2, 5\}**
  - Au moins **2** répétitions de 'x' : **x\{2, \}**

Expressions Régulière Utilisation

- Nom de variables en C
  - **`[a-zA-Z_][a-zA-Z_0-9]*`**
- Prix en DH avec éventuellement des centimes
  - **`[0-9]+(\.[0-9][0-9])? (DH|Dh|dh)`**
- Heure du jour
  - **`(2[0-4]| 1[0-9]|[1-9]):[0-5][0-9]`**
- Entete HTML <h1> <H1> <h2> …
  - **`<[hH][1-4]>`**

La commande find

- **find** *chemin* expression action
- **find** explore récursivement a partir du *chemin* et confronte *expression* avec chaque fichier ou répertoire.
- *expression* est de la forme [-critère [argument_critère]] [Oper. Log] [-critère [argument_critère]] …
- *action*: action a exécuter si expression est réussit
- Oper. Log: operateur logique –a (et) –o (ou) ! negation

La commande find (Cont.)

- Critères de sélection:

-name	*nom du fichier*	
-type	f, d, c, b, p, s, l	
-size	+-*valeur(*cbkMG)	: taille >,< à valeur (car, blocs, ko, Mega,Giga) sans signe, par défaut = taille
-user	*propriétaire*	
-group	*groupe*	
-perm	+-*droits*	: au plus/moins les droits (rwx)
-ctime	+-*nbjours*	: status fichier modifié depuis nbjours
-mtime	+-*nbjours*	: dernière modif. remonte à nbjours
-atime	+-*nbjours*	: dernier accès remonte à nbjours
-links	+-nbliens	: nobre de liens > < ou = nbliens
-newer	*fiche*	*:* f fichier nouveau que fiche

- \+ equ > - equ < blanc equ =

La commande find (Cont.)

- Critères d'exécution:
  - –print - Affiche le chemin d'accès (par défaut)
  - –exec cmd {} \; - Exécute cmd avec comme argument le fichier trouvé
  - –ok cmd {} \; - Demande une confirmation pour exécuter la cmd avec comme argument le fichier trouvé

problèmes

- Les caractères spéciaux dans une expression doivent être précédés d'un échappement (\) pour ne pas être interprété par le shell. Ces caractères sont (), [ ], ?, et *.

- chaque élément dans une expression est un argument séparé et doivent être séparés par des blancs.

- Quand on place un échappement \, on doit placer des blancs autour de \caractère: ␣sp\[sp␣
- Ou il faut mettre l'expression entre côtes

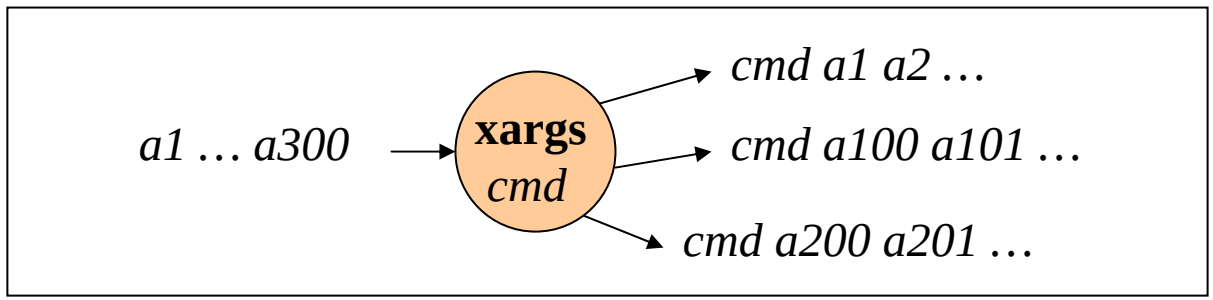
find

Exemples

- Trouver tous les fichiers sous le répertoire personnel commencant avec f
  - **`find ~ -name 'f*' -print`**
- Trouver tous les fichiers sous le répertoire personnel modifie dans le dernier jour.
  - **`find ~ -mtime 1 -print`**
- Trouver tous les fichiers sous le répertoire personnel avec taille plus large que 10K
  - **`find ~ -size 10k -print`**
- Compter les mots des fichiers sous le répertoire personnel
  - **`find ~ -exec wc -w {} \; -print`**
- Effacer les fichier core
  - **`find / -name core –exec rm {} \;`**

xargs

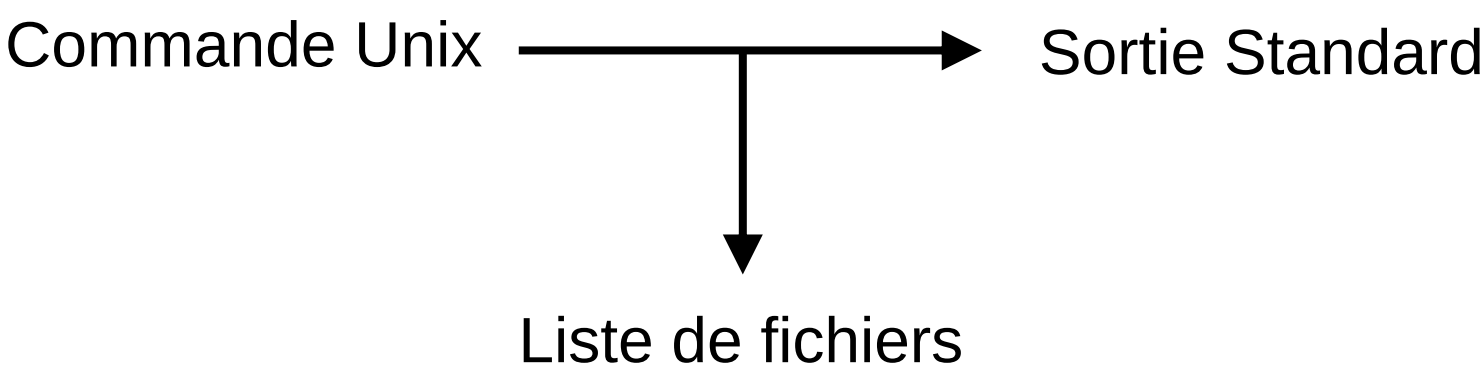
- Unix limite la taille des arguments et environnent qui peuvent être passés.
- Qui ce qui se passe c'est si on a un liste de 10,000 fichiers a envoyer a une commande?
- **xargs** résout le problème
  - Lit les arguments de l'entrée standard
  - Les envoi aux commandes qui les traitent



find et xargs

- **`find . -type f -print | xargs wc -l`**
  - **xargs** invoques **wc** 1 ou plusieurs fois
- Compare a : **`find . -type f –exec wc -l {} \;`**

tee



- Copie l'entrée standard vers la sortie standard et un ou plusieurs fichiers
  - Capture les résultats intermédiaire d'un filtre vers un tube

tee

Exemples

- **ls | head –10 | tee 10_premiers | tail –5**
- **who | tee user_list | wc**