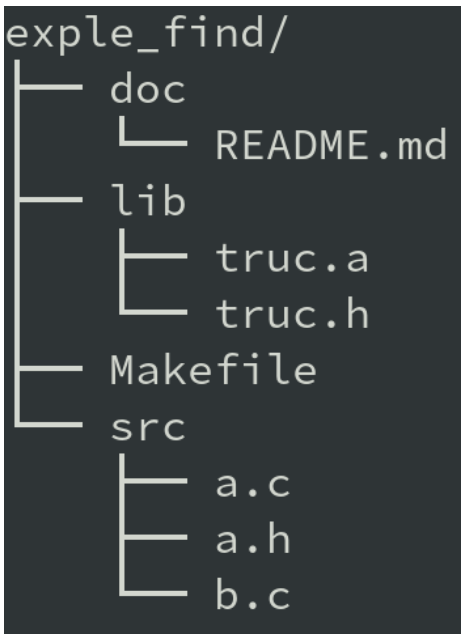
Initiation à l'environnement Unix
CM sur `find`

Pierre Rousselin

Université Sorbonne Paris Nord
L1 informatique et DL
novembre 2024

Commande `find`, premiers exemples

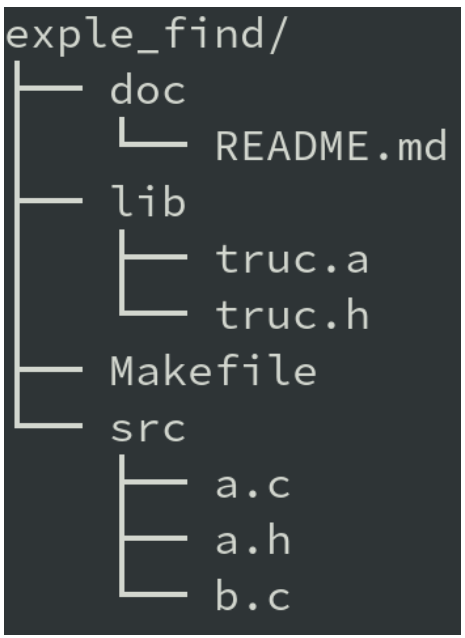
Hypothèse : on se trouve dans le répertoire parent de `exple_find`.




```
$ find exple_find
exple_find
exple_find/src
exple_find/src/a.c
exple_find/src/b.c
exple_find/src/a.h
exple_find/doc
exple_find/doc/README.md
exple_find/lib
exple_find/lib/truc.a
exple_find/lib/truc.h
exple_find/Makefile
```

Commande `find`, premiers exemples

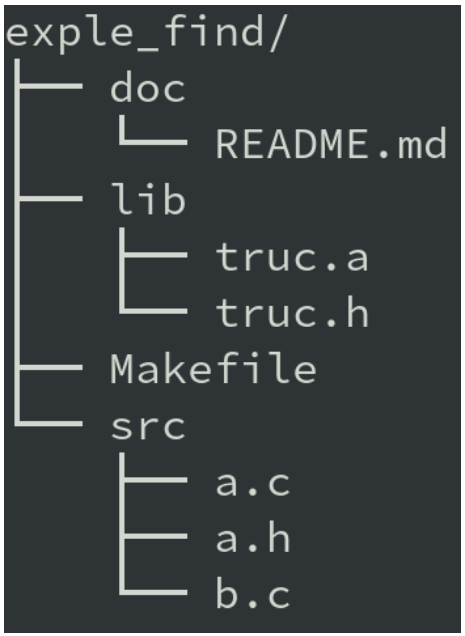
Hypothèse : on se trouve dans le répertoire parent de `exple_find`.




```
$ find exple_find -name '*.c'
exple_find/src/a.c
exple_find/src/b.c
$ find ./exple_find -name '*.h'
./exple_find/src/a.h
./exple_find/lib/truc.h
$ find exple_find/ -type d
exple_find/
exple_find/src
exple_find/doc
exple_find/lib
$ find exple_find/ -type d -name '??c'
exple_find/src
exple_find/doc
```

Commande `find`, premiers exemples

Hypothèse : on se trouve dans le répertoire parent de `exple_find`.




```
$ find exple_find/ -type d -o -name '*.h'
exple_find/
exple_find/src
exple_find/src/a.h
exple_find/doc
exple_find/lib
exple_find/lib/truc.h
$ find exple_find -type f ! -name '*.*'
exple_find/Makefile
$ find exple_find -path 'exple_find/*i*'
exple_find/lib
exple_find/lib/truc.a
exple_find/lib/truc.h
exple_find/Makefile
$ find exple_find /usr/include -name 'tr*c.h'
/usr/include/clc/math/trunc.h
exple_find/lib/truc.h
```

Commande `find`, résumé

```
find rep_depart... [critère_ou_action...]
```

- La commande `find` recherche des fichiers correspondant à certains critères dans une (ou plusieurs) partie(s) de l'arborescence des fichiers et effectue une action sur ces fichiers (par défaut, imprime un nom de chemin du fichier).
- Les critères ont le plus souvent la forme

  `-nom_critere argument_critere`
- La recherche est récursive à partir du ou des répertoires de départ.
- C'est l'un des utilitaires standard les plus puissants (et donc, comme d'habitude, l'un des plus dangereux...)

Commande `find` : critères

Logique des critères :

- Les différents critères sont implicitement reliés par un « et logique ».
- Le « ou logique » s'obtient avec l'argument `-o`, la négation logique avec l'argument `!` et un « et logique » explicite avec l'arguments `-a` (précédés et suivis de blancs).
- Priorité des opérateurs : `!` puis `-a` puis `-o`.
- L'évaluation est toujours « court-circuit » (l'évaluation des critères s'arrête dès que l'on peut conclure).
- On peut regrouper des critères avec des parenthèses inhibées, par exemple `\(` et `\)`.

```
$ find ~ -type d -a \( -name '*shell*' -o ! -name '*[0-9]*' \)
```

Commande `find` : critères

Logique des critères :

- Les différents critères sont implicitement reliés par un « et logique ».
- Le « ou logique » s'obtient avec l'argument `-o`, la négation logique avec l'argument `!` et un « et logique » explicite avec l'arguments `-a` (précédés et suivis de blancs).
- Priorité des opérateurs : `!` puis `-a` puis `-o`.
- L'évaluation est toujours « court-circuit » (l'évaluation des critères s'arrête dès que l'on peut conclure).
- On peut regrouper des critères avec des parenthèses inhibées, par exemple `\(` et `\)`.

```
$ find ~ -type d -a \( -name '*shell*' -o ! -name '*[0-9]*' \)
```

Cherche dans mon répertoire personnel les fichiers qui

- sont des répertoires
- dont le nom de base contient la chaîne `shell` ou
- dont le nom de base ne contient aucun chiffre.

Critères sur le nom ou le type du fichier

- `-name` : avec en argument un motif shell qui doit être inhibé !
  A la valeur « vrai » ssi le nom de base du fichier trouvé correspond au motif shell.
- `-path` : avec en argument un motif shell qui doit être inhibé !
  A la valeur « vrai » ssi le nom entier du fichier trouvé correspond au motif shell.
- `-type` : suivi d'un caractère parmi `f`, `d` (ou d'autres dont on ne parlera pas) vrai ssi le fichier est un fichier normal pour `-type f`, un répertoire pour `-type d`, ...

Critères numériques

Les arguments numériques sont du type

- `n` : exactement `n` ;
- `+n` : plus de `n` ;
- `-n` : moins de `n`.

1. raisons historiques...

Critères numériques

Les arguments numériques sont du type

- `n` : exactement `n` ;
- `+n` : plus de `n` ;
- `-n` : moins de `n`.
- `atime`, `ctime`, `mtime` : respectivement date de dernier accès (*access time*), date de création (*creation time*) et date de dernière modification (*modification time*). L'argument numérique est un nombre de jours. Exemples :
  `find /tmp/ -mtime -3` recherche à partir du répertoire `/tmp/` les fichiers modifiés il y a moins de 3 jours.
  `find ~ -atime +100 -name '*.pdf'` recherche à partir du répertoire personnel les fichiers `pdf` n'ayant pas été lus pendant les 100 derniers jours.

1. raisons historiques...

Critères numériques

Les arguments numériques sont du type

- `n` : exactement `n` ;
- `+n` : plus de `n` ;
- `-n` : moins de `n`.
- `atime`, `ctime`, `mtime` : respectivement date de dernier accès (*access time*), date de création (*creation time*) et date de dernière modification (*modification time*). L'argument numérique est un nombre de jours. Exemples :
  `find /tmp/ -mtime -3` recherche à partir du répertoire `/tmp/` les fichiers modifiés il y a moins de 3 jours.
  `find ~ -atime +100 -name '*.pdf'` recherche à partir du répertoire personnel les fichiers `pdf` n'ayant pas été lus pendant les 100 derniers jours.
- `-size` : l'argument numérique est un nombre de blocs de 512 octets [1] ou bien un nombre d'octets si le nombre est suivi du caractère `c`. Exemple :
  `find ~ -size +2000c` recherche les fichiers de plus de 2000 octets dans le répertoire personnel.

1. raisons historiques...

Le critère-action `exec`

Pour l'instant, `find` semble bien inoffensive... mais avec `exec` on peut tout casser!!!!

- `-exec nom_commande [argument...] \;`
- Si un argument est `{}` (une paire d'accolades), il est remplacé par le nom du fichier en cours d'examen par `find`.
- Le point-virgule (inhibé!) met fin à ce critère/action.
- La valeur de vérité d'un tel `exec` est donnée par le code de retour de la commande.

```
$ mkdir sauvegarde_C
$ find ~ -name '*.c' -exec cp {} sauvegarde_C \;
$ # et le shell, ça suffit, j'en ai marre !
$ find ~ \( -name '*.sh' -o -name '*shell*' \) \
> -exec rm -rf {} \; # ATTENTION : NE PAS TESTER
```

Les autres actions

- `-print` : affiche le nom du fichier, sous-entendu s'il n'y a pas d'autre action. Valeur de vérité : toujours vrai.
- `-ok` : même chose que `-exec` mais demande confirmation avant de lancer la commande. Valeur de vérité : vrai ssi la commande est confirmée et a pour code de retour `0`.
- `-exec ... +` : une variante de `-exec`, voir transparents suivants.
- On peut bien sûr utiliser plusieurs actions.

```
$ find ~ -name '*projet*' -type f -print -exec cat {} \;\
> -ok rm -f {} \;
```

Pour chaque fichier normal dont le nom de base contient la chaîne `projet`,

1. imprimer le nom de ce fichier ;
2. afficher le contenu de ce fichier ;
3. demander à l'utilisateur s'il veut le supprimer.

-exec +

- Syntaxe : `-exec nom_commande [argument...] {} +`
  `{}` apparaît seulement en dernier, avant un caractère `+` (à la place de `;`) qui met fin au critère/action.
- La commande `find` regroupe les noms de fichiers par paquets de taille variable mais si possible importante
- puis lance la commande `nom_commande` avec les `arguments` éventuels précisés avant `{}` et en remplaçant `{}` par tous les noms du paquet.
- Valeur de vérité : toujours vrai.
- Intérêt : la commande

  ```
  $ cmd fich1 fich2 ... fich1000
  ```

  est en général bien plus rapide que les 1000 commandes

  ```
  $cmd fich1; cmd fich2; ...; cmd fich1000
  ```

`-exec ... \;` versus `-exec ... {} +`

On a écrit un script `nb_args` qui écrit juste son nombre d'arguments.

```
#!/bin/sh
# nb_args : affiche son nombre d'arguments
printf '%d\n' $#
```

```
$ find / -name '*.h' -exec ./nb_args {} + 2>/dev/null
1820
1812
(...)
1323
$ find / -name '*.h' -exec ./nb_args {} \; 2>/dev/null
1
1
1
(...)
```

Que vaut la constante `EXIT_SUCCESS` en C ?

Mince, j'ai oublié dans quel fichier `.h` elle était `#define`'d...
Remarque : le mot-clé `time` avant une commande affiche sur le terminal le temps passé entre le début et la fin d'une commande.

```
$ time find / -name '*.h' \
> -exec grep 'EXIT_SUCCESS' {} \; 2>/dev/null

#define EXIT_SUCCESS 0 /* Successful exit status.  */


real    0m49,235s
user    0m17,181s
sys     0m19,311s
```

Que vaut la constante `EXIT_SUCCESS` en C ?

Mince, j'ai oublié dans quel fichier `.h` elle était `#define`'d...
Remarque : le mot-clé `time` avant une commande affiche sur le terminal le temps passé entre le début et la fin d'une commande.

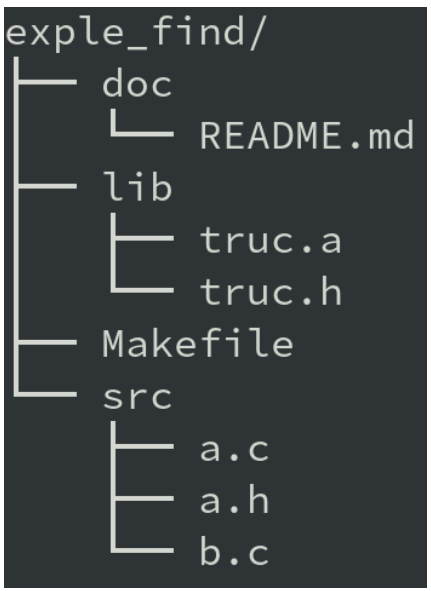
```
$ time find / -name '*.h' \
> -exec grep 'EXIT_SUCCESS' {} \; 2>/dev/null

#define EXIT_SUCCESS 0 /* Successful exit status.  */


real    0m49,235s
user    0m17,181s
sys     0m19,311s

$ time find / -name '*.h' \
> -exec grep 'EXIT_SUCCESS' {} + 2>/dev/null
/usr/include/stdlib.h:#define EXIT_SUCCESS 0 (...)

real    0m1,986s
user    0m0,826s
sys     0m1,136s
```

`-prune`

- En anglais *to prune* veut dire *élaguer*, c'est-à-dire couper certaines branches d'un arbre.
- Il arrive souvent qu'on veuille ignorer certains répertoires lors de la recherche récursive d'une commande `find`.
- C'est exactement ce que fait le critère/action `-prune`.
- Si le « critère » `-prune` est évalué pour un répertoire, les fichiers qu'il contient ne seront pas parcourus.
- Valeur de vérité : toujours vrai.

-prune : exemples

Hypothèse : le répertoire courant est `exple_find`.




```
$ find . \( -name 'src' -prune \) -o -print
.
./doc
./doc/README.md
./lib
./lib/truc.a
./lib/truc.h
./Makefile
$ find . ! -name . -prune
./src
./doc
./lib
./Makefile
```

Autres critères standard

On ne s'étendra pas dessus... voir le manuel

- `-perm` : permissions du fichier
- `-user` et `-group` : utilisateur et groupe propriétaires
- `-depth` : parcours en profondeur d'abord : les répertoires sont visités après les fichiers qu'ils contiennent
- `-newer fich` : vrai si le fichier parcouru a une date de modification plus récente que celle de `fich`
- `-links n` : nombre de liens physiques (voir un cours ultérieur)
- `-nouser`, `-nogroup`, `-xdev` : on oublie

Que fait cette commande ?

```
$ find ~ -type f -size +2000 -atime +50 -exec du -k {} +
```

Que fait cette commande ?

```
$ find ~ -type f -size +2000 -atime +50 -exec du -k {} +
```

Affiche les noms des fichiers de l'arborescence issue du répertoire personnel, dont la taille est supérieure à 1000 ko et dont le dernier accès remonte à au moins 50 jours, précédés par leurs tailles (en ko) (voir le manuel de la commande `du` pour *disk usage*).

Que fait cette commande ?

```
$ find ~ -type f -size +2000 -atime +50 -exec du -k {} +
```

Affiche les noms des fichiers de l'arborescence issue du répertoire personnel, dont la taille est supérieure à 1000 ko et dont le dernier accès remonte à au moins 50 jours, précédés par leurs tailles (en ko) (voir le manuel de la commande `du` pour *disk usage*).

```
$ find ~ -type f -size +2000 -atime +50 -exec du -k {} + |
> sort -nr | head -n20 >big_old_files
```

Que fait cette commande ?

```
$ find ~ -type f -size +2000 -atime +50 -exec du -k {} +
```

Affiche les noms des fichiers de l'arborescence issue du répertoire personnel, dont la taille est supérieure à 1000 ko et dont le dernier accès remonte à au moins 50 jours, précédés par leurs tailles (en ko) (voir le manuel de la commande `du` pour *disk usage*).

```
$ find ~ -type f -size +2000 -atime +50 -exec du -k {} + |
> sort -nr | head -n20 >big_old_files
```

Idem, mais trie par taille de fichier décroissante la liste, puis écrit les tailles et les noms des 20 plus gros fichiers dans le fichier `big_old_files`.

Que fait cette commande ?

```
$ find . -path './*/*' -type f -exec mv {} . \;
```

Que fait cette commande ?

```
$ find . -path './*/*' -type f -exec mv {} . \;
```

Déplace chaque fichier *normal* de l'arborescence du répertoire courant, situé à une profondeur au moins 2, dans le répertoire courant.

Que fait cette commande ?

```
$ find . -path './*/*' -type f -exec mv {} . \;
```

Déplace chaque fichier *normal* de l'arborescence du répertoire courant, situé à une profondeur au moins 2, dans le répertoire courant.

```
$ find . -depth \
> -path './*/*' -type f -exec mv {} . \; \
> -o ! -name . -type d -exec rmdir {} +
```

Que fait cette commande ?

```
$ find . -path './*/*' -type f -exec mv {} . \;
```

Déplace chaque fichier *normal* de l'arborescence du répertoire courant, situé à une profondeur au moins 2, dans le répertoire courant.

```
$ find . -depth \
> -path './*/*' -type f -exec mv {} . \; \
> -o ! -name . -type d -exec rmdir {} +
```

Déplace chaque fichier *normal* de l'arborescence du répertoire courant, situé à une profondeur au moins 2, dans le répertoire courant, puis supprime tous les sous-répertoires de cette arborescence. Bref, « aplatit » l'arborescence issue du répertoire courant.