Survie en bash

Prof d'info

Lycée Thiers

Commentaire

Les commentaires ne sont pas interprétés :

- La commande `kalin` n'existe pas, elle soulève une erreur :

```
$ kalin
La commande <<kalin>> n'a pas été trouvée, ...
```

Commentaire

Les commentaires ne sont pas interprétés :

- La commande `kalin` n'existe pas, elle soulève une erreur :

```
$ kalin
La commande <<kalin>> n'a pas été trouvée, ...
```

- Le caractère **#** n'est pas interprété. On peut écrire `kalin` sans erreur :

```
$ # kalin
$
```

Contenu d'un répertoire ♡

La commande `ls` donne le contenu d'un répertoire :

- sans argument : les entrées du répertoire courant `ls`

Contenu d'un répertoire ♡

La commande `ls` donne le contenu d'un répertoire :

- sans argument : les entrées du répertoire courant `ls`
- avec argument : les entrées repérées par le (ou les) argument(s) :
  `ls myFile`, `ls myRep`, `ls /etc /usr/bin`

Contenu d'un répertoire ♡

La commande `ls` donne le contenu d'un répertoire :

- sans argument : les entrées du répertoire courant `ls`
- avec argument : les entrées repérées par le (ou les) argument(s) : `ls myFile`, `ls myRep`, `ls /etc /usr/bin`
- pour afficher les fichiers cachés `ls -a` (indique notamment **.bashrc** si je suis en ~)

Contenu d'un répertoire ♡

La commande `ls` donne le contenu d'un répertoire :

- sans argument : les entrées du répertoire courant `ls`
- avec argument : les entrées repérées par le (ou les) argument(s) : `ls myFile`, `ls myRep`, `ls /etc /usr/bin`
- pour afficher les fichiers cachés `ls -a` (indique notamment **.bashrc** si je suis en ~)
- pour pour tous les attributs (type, droits, liens physiques, propriétaire, groupe, taille, date, nom) `ls -l`

Contenu d'un répertoire ♡

La commande `ls` donne le contenu d'un répertoire :

- sans argument : les entrées du répertoire courant `ls`
- avec argument : les entrées repérées par le (ou les) argument(s) : `ls myFile`, `ls myRep`, `ls /etc /usr/bin`
- pour afficher les fichiers cachés `ls -a` (indique notamment **.bashrc** si je suis en ~)
- pour pour tous les attributs (type, droits, liens physiques, propriétaire, groupe, taille, date, nom) `ls -l`
- `ls -al` au lieu de `ls -a -l`.

```
$ ls -lh
total 1,8M
-rw-rw-r-- 1 ivan ivan  67K août  25  2021 accesDirec
-rw-rw-r-- 1 ivan ivan 138K août  25  2021 accesSeque
```

Dans l'ordre : droits, nombres d'alias, username, groupname, taille (par défaut en octet), date de dernière modif., nom de fichier.

Contenu d'un répertoire ♡

La commande `ls` donne le contenu d'un répertoire :

- sans argument : les entrées du répertoire courant `ls`
- avec argument : les entrées repérées par le (ou les) argument(s) : `ls myFile`, `ls myRep`, `ls /etc /usr/bin`
- pour afficher les fichiers cachés `ls -a` (indique notamment **.bashrc** si je suis en ~)
- pour pour tous les attributs (type, droits, liens physiques, propriétaire, groupe, taille, date, nom) `ls -l`
- `ls -al` au lieu de `ls -a -l`.

```
$ ls -lh
total 1,8M
-rw-rw-r-- 1 ivan ivan  67K août  25  2021 accesDirec
-rw-rw-r-- 1 ivan ivan 138K août  25  2021 accesSeque
```

Dans l'ordre : droits, nombres d'alias, username, groupname, taille (par défaut en octet), date de dernière modif., nom de fichier.

`ls -i` affiche le numéro d'inode

Affichage d'une chaîne de caractère

La commande `echo` affiche une ligne de texte

- (Le caractère **#** indique le début d'un commentaire

```
$ echo 'coucou' # afficher coucou
coucou
```

Affichage d'une chaîne de caractère

La commande `echo` affiche une ligne de texte

- (Le caractère **#** indique le début d'un commentaire

```
$ echo 'coucou' # afficher coucou
coucou
```

- le choix des guillements est important : ' (touche 4), " (touche 3) et ` (ALT GR + 7) n'ont pas le même sens.

Affichage d'une chaîne de caractère

La commande `echo` affiche une ligne de texte

- (Le caractère **#** indique le début d'un commentaire

```
$ echo 'coucou' # afficher coucou
coucou
```

- le choix des guillements est important : ' (touche 4), " (touche 3) et ` (ALT GR + 7) n'ont pas le même sens.
- Pour afficher le contenu d'une variable d'environnement :

```
$ echo $LANG
fr_FR.UTF-8
```

Les métacaractères

Les métacaractères du shell permettent :

- de construire des chaînes de caractères génériques

Les métacaractères

Les métacaractères du shell permettent :

- de construire des chaînes de caractères génériques
- de modifier l'interprétation d'une commande

Métacaractères de construction

Prioritaires : `*, ?`

- `*` désigne une chaîne de caractères quelconque ♡

Métacaractères de construction

Prioritaires : `*, ?`

- `*` désigne une chaîne de caractères quelconque ♡
- `?` désigne un caractère quelconque ♡

Métacaractères de construction

Prioritaires : `*, ?`

- `*` désigne une chaîne de caractères quelconque ♡
- `?` désigne un caractère quelconque ♡
- `[...]` désigne les caractères entre crochets, définis par énumération ou par un intervalle :

Métacaractères de construction

Prioritaires : `*, ?`

- `*` désigne une chaîne de caractères quelconque ♡
- `?` désigne un caractère quelconque ♡
- `[...]` désigne les caractères entre crochets, définis par énumération ou par un intervalle :
  - `[Aa]` désigne les caractères A ou a,

Métacaractères de construction

Prioritaires : `*, ?`

- `*` désigne une chaîne de caractères quelconque ♡
- `?` désigne un caractère quelconque ♡
- `[...]` désigne les caractères entre crochets, définis par énumération ou par un intervalle :
  - `[Aa]` désigne les caractères A ou a,
  - `[0-9a-zA-Z]` désigne un caractère alphanumérique quelconque.

Métacaractères de construction

Prioritaires : `*, ?`

- `*` désigne une chaîne de caractères quelconque ♡
- `?` désigne un caractère quelconque ♡
- `[...]` désigne les caractères entre crochets, définis par énumération ou par un intervalle :
  - `[Aa]` désigne les caractères A ou a,
  - `[0-9a-zA-Z]` désigne un caractère alphanumérique quelconque.
  - `[!0-9]` désigne l'ensemble des caractères sauf les chiffres.

Métacaractères de construction

Voici le contenu du répertoire courant :

```
$ ls
alain   Ali   tata   titi   toto   tutu   zut
```

- `ls t[ao]t[ao]` retourne **tata** et **toto**,

Métacaractères de construction

Voici le contenu du répertoire courant :

```
$ ls
alain   Ali   tata   titi   toto   tutu   zut
```

- `ls t[ao]t[ao]` retourne **tata** et **toto**,
- `ls ???` retourne les noms de 3 lettres donc **zut** et **Ali**

Métacaractères de construction

Voici le contenu du répertoire courant :

```
$ ls
alain   Ali   tata   titi   toto   tutu   zut
```

- `ls t[ao]t[ao]` retourne **tata** et **toto**,
- `ls ???` retourne les noms de 3 lettres donc **zut** et **Ali**
- `ls A*` retourne les noms qui commencent par A donc **Ali**

Métacaractères de construction

Voici le contenu du répertoire courant :

```
$ ls
alain   Ali   tata   titi   toto   tutu   zut
```

- `ls t[ao]t[ao]` retourne **tata** et **toto**,
- `ls ???` retourne les noms de 3 lettres donc **zut** et **Ali**
- `ls A*` retourne les noms qui commencent par A donc **Ali**
- `ls t??o` retourne les noms de 4 lettres qui terminent par o et commencent par t donc **toto**

Métacaractères de construction

Voici le contenu du répertoire courant :

```
$ ls
alain   Ali   tata   titi   toto   tutu   zut
```

- `ls t[ao]t[ao]` retourne **tata** et **toto**,
- `ls ???` retourne les noms de 3 lettres donc **zut** et **Ali**
- `ls A*` retourne les noms qui commencent par A donc **Ali**
- `ls t??o` retourne les noms de 4 lettres qui terminent par o et commencent par t donc **toto**
- `ls [!b-z]*` désigne les noms qui ne commencent pas par une lettre entre b et z donc **alain** et **Ali**.

Métacaractères de modification **(PI)**

- `;` sépare deux commandes sur une même ligne

Métacaractères de modification **(PI)**

- `;` sépare deux commandes sur une même ligne
- Les guillemets verticaux simples ' (touche 4) délimitent une chaîne de caractères contenant des espaces (à l'intérieur, tous les métacaractères perdent leur signification) ;

Métacaractères de modification **(PI)**

- `;` sépare deux commandes sur une même ligne
- Les guillemets verticaux simples ' (touche 4) délimitent une chaîne de caractères contenant des espaces (à l'intérieur, tous les métacaractères perdent leur signification) ;
- Les guillemets verticaux doubles " (touche 3) délimitent une chaîne de caractères contenant des espaces (à l'intérieur, tous les métacaractères perdent leur signification, à l'exception des métacaractères ` et `$`) ;

Métacaractères de modification **(PI)**

- `;` sépare deux commandes sur une même ligne
- Les guillemets verticaux simples ' (touche 4) délimitent une chaîne de caractères contenant des espaces (à l'intérieur, tous les métacaractères perdent leur signification) ;
- Les guillemets verticaux doubles " (touche 3) délimitent une chaîne de caractères contenant des espaces (à l'intérieur, tous les métacaractères perdent leur signification, à l'exception des métacaractères ` et `$`) ;
- Les guillemets obliques gauche-droite ` (ALT GR + 7)
  « capturent » la sortie standard pour former un nouvel argument ou une nouvelle commande ;

```
$ echo "$LANG"; echo '$LANG' # deux commandes successives
fr_FR.UTF-8
$LANG
$ echo "on est `date` " # observer les guillemets
on est jeu. 25 janv. 2024 19:17:45 CET
```

Métacaractères de modification **(PI)**

- `\` annule la signification du métacaractère qui suit : c'est un caractère dit d'*échappement*.

```
$ echo "on est \`date\` "
on est `date`
```

Métacaractères de modification **(PI)**

- `\` annule la signification du métacaractère qui suit : c'est un caractère dit d'*échappement*.

```
$ echo "on est \`date\` "
on est `date`
```

- le `&` à la fin d'une commande permet de lancer celle-ci en tâche de fond, donc sans bloquer le terminal.

```
$emacs toto # mon terminal va se bloquer
```

`CTRL + C` pour quitter brutalement. Ou mieux `CTRL + z` (passage à l'état zombi) suivi de `bg` (remise en tâche de fond).

```
$emacs toto & # mon terminal ne va pas se bloquer
```

Métacaractères de modification **(PI)**

Parenthèses

- `(...)` : les parenthèses encadrent une suite de commandes qui sont exécutées par un shell secondaire. En particulier, les assignements n'ont pas d'effet en dehors des parenthèses.

Métacaractères de modification **(PI)**

Parenthèses

- `(...)` : les parenthèses encadrent une suite de commandes qui sont exécutées par un shell secondaire. En particulier, les assignements n'ont pas d'effet en dehors des parenthèses.
- `{...}` : les accolades encadrent une suite de commandes qui sont exécutées par le shell principal. En particulier, les assignements ont un effet en dehors des accolades.

Métacaractères de modification **(PI)**

Parenthèses

- `(...)` : les parenthèses encadrent une suite de commandes qui sont exécutées par un shell secondaire. En particulier, les assignements n'ont pas d'effet en dehors des parenthèses.
- `{...}` : les accolades encadrent une suite de commandes qui sont exécutées par le shell principal. En particulier, les assignements ont un effet en dehors des accolades.
- `[...]` : les crochets sont utilisés pour les instructions conditionnelles. Ils encadrent une expression à valeur bouléenne.

```
$ [ -d presentationLinux.tex ] # est-ce un dossier ?
$ echo $? # afficher la réponse du test précédent
1
```

On obtient 0 si le fichier existe et est un répertoire, 1 sinon. INVERSE DE C ! !

Métacaractères de modification **(PI)**

Parenthèses

- `(...)` : les parenthèses encadrent une suite de commandes qui sont exécutées par un shell secondaire. En particulier, les assignements n'ont pas d'effet en dehors des parenthèses.
- `{...}` : les accolades encadrent une suite de commandes qui sont exécutées par le shell principal. En particulier, les assignements ont un effet en dehors des accolades.
- `[...]` : les crochets sont utilisés pour les instructions conditionnelles. Ils encadrent une expression à valeur bouléenne.

```
$ [ -d presentationLinux.tex ] # est-ce un dossier ?
$ echo $? # afficher la réponse du test précédent
1
```

On obtient 0 si le fichier existe et est un répertoire, 1 sinon. INVERSE DE C ! !

- ...

Utiliser le résultat d'une commande comme argument d'une autre **(PI)**

Pour info. Les ` (ALT GR + 7) entourant une commande permettent d'utiliser le résultat de cette commande comme argument(s) dans la ligne de commande.

- ```
  $ echo "Nous sommes le" `date +%d/%m/%y`
  Nous sommes le 27/08/21
  ```

  Affiche la date du jour avec un format choisi.
  ```

Utiliser le résultat d'une commande comme argument d'une autre **(PI)**

Pour info. Les ` (ALT GR + 7) entourant une commande permettent d'utiliser le résultat de cette commande comme argument(s) dans la ligne de commande.

- ```
  $ echo "Nous sommes le" `date +%d/%m/%y`
  Nous sommes le 27/08/21
  ```

  Affiche la date du jour avec un format choisi.

- ```
  $ echo "2 + 2 =" `expr 2 + 2`
  2 + 2 = 4
  ```

Positionnement/recherche dans l'arborescence ♡

pwd affiche le nom absolu du répertoire de travail

Positionnement/recherche dans l'arborescence ♡

pwd affiche le nom absolu du répertoire de travail

cd change le répertoire de travail.

Positionnement/recherche dans l'arborescence ♡

pwd affiche le nom absolu du répertoire de travail

cd change le répertoire de travail.

- Avec argument : se rend à la destination indiquée.

`cd ../Rep1` ;

Positionnement/recherche dans l'arborescence ♡

pwd affiche le nom absolu du répertoire de travail

cd change le répertoire de travail.

- Avec argument : se rend à la destination indiquée. `cd ../Rep1` ;
- sans argument : retourne au répertoire de connexion du user. `cd`

Positionnement/recherche dans l'arborescence ♡

pwd affiche le nom absolu du répertoire de travail

cd change le répertoire de travail.

- Avec argument : se rend à la destination indiquée. `cd ../Rep1` ;
- sans argument : retourne au répertoire de connexion du user. `cd`

ls liste les entrées d'un répertoire (déjà vu) `ls`.

Positionnement/recherche dans l'arborescence ♡

pwd affiche le nom absolu du répertoire de travail

cd change le répertoire de travail.

- Avec argument : se rend à la destination indiquée. `cd ../Rep1` ;
- sans argument : retourne au répertoire de connexion du user. `cd`

ls liste les entrées d'un répertoire (déjà vu) `ls`.

find pour chercher récursivement un ou plusieurs fichiers dans une arborescence. Beaucoup d'options (consulter le manuel).

Rechercher

- `find` cherche récursivement dans l'arborescence à partir du point indiqué. `find /usr -name "ls*"` cherche les fichiers dont le nom commence par `ls` dans le répertoire /**usr** et ses sous-répertoires. Il y en a beaucoup ! ♡

Rechercher

- `find` cherche récursivement dans l'arborescence à partir du point indiqué. `find /usr -name "ls*"` cherche les fichiers dont le nom commence par `ls` dans le répertoire /**usr** et ses sous-répertoires. Il y en a beaucoup ! ♡
- L'option `-type` permet de ne chercher que les fichiers (`f`) ou les répertoires (`d`). `find /var/log/ -type d -name "*sm*"` : chercher les répertoires dont le nom contient sm

Rechercher

- `find` cherche récursivement dans l'arborescence à partir du point indiqué. `find /usr -name "ls*"` cherche les fichiers dont le nom commence par `ls` dans le répertoire /**usr** et ses sous-répertoires. Il y en a beaucoup ! ♡
- L'option `-type` permet de ne chercher que les fichiers (`f`) ou les répertoires (`d`). `find /var/log/ -type d -name "*sm*"` : chercher les répertoires dont le nom contient sm
- recherche par taille :
`find /Téléchargements -size +20M -size -40M` cherche les fichiers dont la taille est comprise entre 20 Mo et 40Mo.

Rechercher

- `find` cherche récursivement dans l'arborescence à partir du point indiqué. `find /usr -name "ls*"` cherche les fichiers dont le nom commence par `ls` dans le répertoire /**usr** et ses sous-répertoires. Il y en a beaucoup ! ♡
- L'option `-type` permet de ne chercher que les fichiers (`f`) ou les répertoires (`d`). `find /var/log/ -type d -name "*sm*"` : chercher les répertoires dont le nom contient sm
- recherche par taille :
  `find /Téléchargements -size +20M -size -40M` cherche les fichiers dont la taille est comprise entre 20 Mo et 40Mo.
- Recherche par utilisateur : `find /tmp -user adrien` cherche dans `/tmp` les fichiers dont le propriétaire est **adrien**.

Rechercher

- `find` cherche récursivement dans l'arborescence à partir du point indiqué. `find /usr -name "ls*"` cherche les fichiers dont le nom commence par `ls` dans le répertoire /**usr** et ses sous-répertoires. Il y en a beaucoup ! ♡
- L'option `-type` permet de ne chercher que les fichiers (`f`) ou les répertoires (`d`). `find /var/log/ -type d -name "*sm*"` : chercher les répertoires dont le nom contient sm
- recherche par taille :
  `find /Téléchargements -size +20M -size -40M` cherche les fichiers dont la taille est comprise entre 20 Mo et 40Mo.
- Recherche par utilisateur : `find /tmp -user adrien` cherche dans `/tmp` les fichiers dont le propriétaire est **adrien**.
- Beaucoup d'autres options : par date de création, date de dernière modification, par type de permissions, recherche de fichiers vides etc...

Consulter le contenu d'un fichier texte ♡

Commandes de base :

- `cat monfichier`, `more monfichier` : affichage simple et page par page ;

Consulter le contenu d'un fichier texte ♡

Commandes de base :

- `cat monfichier`, `more monfichier` : affichage simple et page par page ;
- `head monfichier`, `head -n monfichier` : affichage des n premières lignes ;

Consulter le contenu d'un fichier texte ♡

Commandes de base :

- `cat monfichier`, `more monfichier` : affichage simple et page par page ;
- `head monfichier`, `head -n monfichier` : affichage des n premières lignes ;
- `tail monfichier`, `tail -n monfichier` : affichage des n dernières lignes ;

Consulter le contenu d'un fichier texte ♡

Commandes de base :

- `cat monfichier`, `more monfichier` : affichage simple et page par page ;
- `head monfichier`, `head -n monfichier` : affichage des n premières lignes ;
- `tail monfichier`, `tail -n monfichier` : affichage des n dernières lignes ;
- `wc monfichier` : affichage du nombre de lignes, de mots, de caractères. Options `-l`, `-w` et `-c` pour les nombres de lignes, de mots et de caractères.

Consulter le contenu d'un fichier texte ♡

Commandes de base :

- `cat monfichier`, `more monfichier` : affichage simple et page par page ;
- `head monfichier`, `head -n monfichier` : affichage des n premières lignes ;
- `tail monfichier`, `tail -n monfichier` : affichage des n dernières lignes ;
- `wc monfichier` : affichage du nombre de lignes, de mots, de caractères. Options `-l`, `-w` et `-c` pour les nombres de lignes, de mots et de caractères.
- `cat toto titi` : affiche le contenu de **titi** à la suite de celui de **toto** (`cat` pour concatène).

Historique ♡

- Le shell bash enregistre toutes les commandes tapées et permet de les rappeler pour les ré-exécuter soit telles quelles, soit modifiées.

Historique ♡

- Le shell bash enregistre toutes les commandes tapées et permet de les rappeler pour les ré-exécuter soit telles quelles, soit modifiées.
- La commande `history` permet de lister le contenu de l'historique des commandes, de façon numérotée. Le caractère `!` permet de rappeler une commande.

Historique ♡

- Le shell bash enregistre toutes les commandes tapées et permet de les rappeler pour les ré-exécuter soit telles quelles, soit modifiées.
- La commande `history` permet de lister le contenu de l'historique des commandes, de façon numérotée. Le caractère `!` permet de rappeler une commande.

  `!!` rappelle la dernière commande

Historique ♡

- Le shell bash enregistre toutes les commandes tapées et permet de les rappeler pour les ré-exécuter soit telles quelles, soit modifiées.
- La commande `history` permet de lister le contenu de l'historique des commandes, de façon numérotée. Le caractère `!` permet de rappeler une commande.

  `!!` rappelle la dernière commande

  `!n` rappelle la commande numéro *n*

Historique ♡

- Le shell bash enregistre toutes les commandes tapées et permet de les rappeler pour les ré-exécuter soit telles quelles, soit modifiées.
- La commande `history` permet de lister le contenu de l'historique des commandes, de façon numérotée. Le caractère `!` permet de rappeler une commande.

`!!`	rappelle la dernière commande
`!n`	rappelle la commande numéro *n*
`!chaine`	rappelle la dernière commande commençant par *chaine*

Historique ♡

- Le shell bash enregistre toutes les commandes tapées et permet de les rappeler pour les ré-exécuter soit telles quelles, soit modifiées.
- La commande `history` permet de lister le contenu de l'historique des commandes, de façon numérotée. Le caractère `!` permet de rappeler une commande.

  `!!` rappelle la dernière commande
  `!n` rappelle la commande numéro *n*
  `!chaine` rappelle la dernière commande commençant par *chaine*
- On peut aussi utiliser les flèches haut et bas pour naviguer dans l'historique des commandes.

Taille du contenu d'un répertoire

- `du -h -d 1 monRepertoire` : taille des fichiers et sous-répertoires. (`du` pour Disk User))

Taille du contenu d'un répertoire

- `du -h -d 1 monRepertoire` : taille des fichiers et sous-répertoires. (`du` pour Disk User))
  - L'option `-h` force un affichage « human readable » (par exemple, 1k, 236M, 2G).

Taille du contenu d'un répertoire

- `du -h -d 1 monRepertoire` : taille des fichiers et sous-répertoires. (`du` pour Disk User))
  - L'option `-h` force un affichage « human readable » (par exemple, 1k, 236M, 2G).
  - L'option `-d` affiche la taille totale du répertoire exploré et pas seulement la taille de ses constituants. Et le paramètre 1 indique la profondeur de l'exploration (ici, on s'arrête aux fils, avec 2 comme paramètre , ce serait aux petits-fils)

Taille du contenu d'un répertoire

- `du -h -d 1 monRepertoire` : taille des fichiers et sous-répertoires. (`du` pour Disk User))
  - L'option `-h` force un affichage « human readable » (par exemple, 1k, 236M, 2G).
  - L'option `-d` affiche la taille totale du répertoire exploré et pas seulement la taille de ses constituants. Et le paramètre 1 indique la profondeur de l'exploration (ici, on s'arrête aux fils, avec 2 comme paramètre , ce serait aux petits-fils)
- La commande `ncdu nomDuRepertoire`, plus conviviale, permet de connaître la place prise par les fichiers et dossiers en navigant dans l'arborescence. Par exemple `ncdu /home` indique les tailles des différents répertoires utilisateurs.

Créer et supprimer

- Créer un répertoire vide : `mkdir Rep1`

Créer et supprimer

- Créer un répertoire vide : `mkdir Rep1`
- Supprimer un répetoire vide : `rmdir Rep1`

Créer et supprimer

- Créer un répertoire vide : `mkdir Rep1`
- Supprimer un répetoire vide : `rmdir Rep1`
- Créer un fichier vide : `touch file1`

Créer et supprimer

- Créer un répertoire vide : `mkdir Rep1`
- Supprimer un répetoire vide : `rmdir Rep1`
- Créer un fichier vide : `touch file1`
- Supprimer un fichier `rm file1`, supprimer tous les fichiers du répertoire `rm *`

Créer et supprimer

- Créer un répertoire vide : `mkdir Rep1`
- Supprimer un répetoire vide : `rmdir Rep1`
- Créer un fichier vide : `touch file1`
- Supprimer un fichier `rm file1`, supprimer tous les fichiers du répertoire `rm *`
- Créer un chemin complet `mkdir -p R1/R2` crée dans la foulée `R1` puis son sous-répertoire `R2`.

Copier (`cp`) et déplacer (`mv`) ♡

Situation : un répertoire parent **P** possède deux sous-répertoires **Rep1** et **Rep2**. Dans **Rep1** on trouve le fichier **file1**.

- Copier le fichier **file1** du répertoire **Rep1** dans le répertoire **Rep2** sans changer le nom : `cp Rep1/file1 Rep2/`

Copier (`cp`) et déplacer (`mv`) ♡

Situation : un répertoire parent **P** possède deux sous-répertoires **Rep1** et **Rep2**. Dans **Rep1** on trouve le fichier **file1**.

- Copier le fichier **file1** du répertoire **Rep1** dans le répertoire **Rep2** sans changer le nom : `cp Rep1/file1 Rep2/`
- Copier le fichier **file1** du répertoire **Rep1** dans le répertoire **Rep2** en changeant le nom : `cp Rep1/file1 Rep2/file2`

Copier (`cp`) et déplacer (`mv`) ♡

Situation : un répertoire parent **P** possède deux sous-répertoires **Rep1** et **Rep2**. Dans **Rep1** on trouve le fichier **file1**.

- Copier le fichier **file1** du répertoire **Rep1** dans le répertoire **Rep2** sans changer le nom : `cp Rep1/file1 Rep2/`
- Copier le fichier **file1** du répertoire **Rep1** dans le répertoire **Rep2** en changeant le nom : `cp Rep1/file1 Rep2/file2`
- Je suis dans **Rep1**. Changer le nom du fichier **file1** en restant dans le même répertoire : `mv file1 file2`.

Copier (`cp`) et déplacer (`mv`) ♡

Situation : un répertoire parent **P** possède deux sous-répertoires **Rep1** et **Rep2**. Dans **Rep1** on trouve le fichier **file1**.

- Copier le fichier **file1** du répertoire **Rep1** dans le répertoire **Rep2** sans changer le nom : `cp Rep1/file1 Rep2/`
- Copier le fichier **file1** du répertoire **Rep1** dans le répertoire **Rep2** en changeant le nom : `cp Rep1/file1 Rep2/file2`
- Je suis dans **Rep1**. Changer le nom du fichier **file1** en restant dans le même répertoire : `mv file1 file2`.
- Je suis dans **Rep1**. déplacer le fichier **file1** dans **rep2** sans changer son nom `mv file1 ../Rep2/`.

Copier (`cp`) et déplacer (`mv`) ♡

Situation : un répertoire parent **P** possède deux sous-répertoires **Rep1** et **Rep2**. Dans **Rep1** on trouve le fichier **file1**.

- Copier le fichier **file1** du répertoire **Rep1** dans le répertoire **Rep2** sans changer le nom : `cp Rep1/file1 Rep2/`
- Copier le fichier **file1** du répertoire **Rep1** dans le répertoire **Rep2** en changeant le nom : `cp Rep1/file1 Rep2/file2`
- Je suis dans **Rep1**. Changer le nom du fichier **file1** en restant dans le même répertoire : `mv file1 file2`.
- Je suis dans **Rep1**. déplacer le fichier **file1** dans **rep2** sans changer son nom `mv file1 ../Rep2/`.
- Je suis dans le répertoire parent de **P**. Je veux copier récursivement **P** et tout ce qu'il contient (donc aussi les sous-répertoires) dans un nouveau répertoire **P2** : `cp -r P P2`

Créer, remplir, vider, supprimer un répertoire

Exemple

- ```
  $ mkdir Asup # Creer répertoire Asup
  ```
  ```

Créer, remplir, vider, supprimer un répertoire

Exemple

- ```
  $ mkdir Asup # Creer répertoire Asup
  ```

- ```
  $ cd Asup # aller au répertoire Asup
  $ ls # pas de contenu
  ```

Créer, remplir, vider, supprimer un répertoire

Exemple

- ```
  $ mkdir Asup # Creer répertoire Asup
  ```

- ```
  $ cd Asup # aller au répertoire Asup
  $ ls # pas de contenu
  ```

- ```
  $ touch asup # créer le fichier vide asup
  $ ls -al # afficher fichiers cachés
  total 8
  drwxrwxr-x 2 ivan ivan 4096 août  19 15:28 .
  drwxrwxr-x 3 ivan ivan 4096 août  19 15:28 ..
  -rw-rw-r-- 1 ivan ivan    0 août  19 15:28 asup
  ```
  ```

Créer, remplir, vider, supprimer un répertoire

Exemple

```
$ mkdir Asup # Creer répertoire Asup
```

```
$ cd Asup # aller au répertoire Asup
$ ls # pas de contenu
```

```
$ touch asup # créer le fichier vide asup
$ ls -al # afficher fichiers cachés
total 8
drwxrwxr-x 2 ivan ivan 4096 août  19 15:28 .
drwxrwxr-x 3 ivan ivan 4096 août  19 15:28 ..
-rw-rw-r-- 1 ivan ivan    0 août  19 15:28 asup
```

```
$ rm asup # supprimer asup
$ ls # plus de contenu
```

Créer, remplir, vider, supprimer un répertoire

Exemple

```
$ cd .. # revenir au père
$ rmdir Asup # supprime rep Asup
```

Créer, remplir, vider, supprimer un répertoire

Exemple

```
$ cd .. # revenir au père
$ rmdir Asup # supprime rep Asup
```

- Les répertoires **Nouveau** et son fils **AutreNouveau** sont créés simultannément puis supprimés de même

```
$ mkdir -p Nouveau/AutreNouveau # creer un répertoire et
$ rmdir -p Nouveau/AutreNouveau # supprimer un répertoire
```

Liens physiques (hard links)

Idée

Un lien physique est un autre nom (une autre entrée de répertoire) qui pointe vers le même inode (mêmes données sur disque).

Propriétés importantes

- Même fichier, même inode $\Rightarrow$ mêmes données.
- Modifier l'un $\Rightarrow$ l'autre est modifié aussi.
- La suppression d'un nom ne supprime pas les données tant qu'il reste au moins un lien.
- En général : pas de hard link vers un répertoire (pour éviter des cycles).
- En général : impossible entre deux systèmes de fichiers différents.

Liens physiques (hard links)

Commandes et exemples

```
$ echo "bonjour" > a.txt
$ ln a.txt b.txt              # crée un lien physique
$ ls -li a.txt b.txt          # même numéro d'inode, compteur de liens
$ cat b.txt
bonjour
$ echo "salut" >> b.txt      # modifie le même contenu
$ cat a.txt
bonjour
salut
```

Liens symboliques (symlinks)

Idée

Un lien symbolique est un petit fichier spécial qui contient un chemin vers une autre cible (fichier ou répertoire).

Propriétés importantes

- Inode différent : le lien est un objet distinct.
- Peut pointer vers un fichier ou un répertoire.
- Peut traverser des systèmes de fichiers différents.
- Si la cible est supprimée/déplacée, le lien devient cassé (*dangling*).

Liens symboliques (symlinks)

Commandes et exemples

```
$ echo "bonjour" > a.txt
$ ln -s a.txt c.txt          # crée un lien symbolique
$ ls -li a.txt c.txt         # inode différent + affichage a.txt -> ...
$ cat c.txt
bonjour
$ rm a.txt                   # supprime la cible
$ cat c.txt
cat: c.txt: No such file or directory
```

Explorer un fichier

`grep` affiche les lignes vérifiant un pattern.

Contenu d'un fichier **myfile** :

```
toto et gogo
tot et gogo
totoooo et gaga
atoto et titi
titi et tutu
totatata tot toa
```

- `grep "toto" myfile` cherche le mot « toto » dans **myfile**

Explorer un fichier

`grep` affiche les lignes vérifiant un pattern.
Contenu d'un fichier **myfile** :

```
toto et gogo
tot et gogo
totoooo et gaga
atoto et titi
titi et tutu
totatata tot toa
```

- `grep "toto" myfile` cherche le mot « toto » dans **myfile**
- `grep -c "toto" myfile` cherche le nombre d'occurences du mot « toto » dans **myfile**.

Explorer un fichier

`grep` affiche les lignes vérifiant un pattern.
Contenu d'un fichier **myfile** :

```
toto et gogo
tot et gogo
totoooo et gaga
atoto et titi
titi et tutu
totatata tot toa
```

- `grep "toto" myfile` cherche le mot « toto » dans **myfile**
- `grep -c "toto" myfile` cherche le nombre d'occurences du mot « toto » dans **myfile**.
- Option `grep -n "toto"` pour afficher les numéros de lignes.

Explorer un fichier (suite)

`grep` affiche les lignes vérifiant un pattern.
Contenu d'un fichier `myfile` :

```
toto et gogo
tot et gogo
totoooo et gaga
atoto et titi
titi et tutu
totatata tot toa
```

- `grep "toto$" myfile` : les lignes qui terminent par toto

Explorer un fichier (suite)

`grep` affiche les lignes vérifiant un pattern.
Contenu d'un fichier `myfile` :

```
toto et gogo
tot et gogo
totoooo et gaga
atoto et titi
titi et tutu
totatata tot toa
```

- `grep "toto$" myfile` : les lignes qui terminent par toto
- `grep "^toto"` : les lignes qui commencent par toto

Explorer un fichier (suite)

`grep` affiche les lignes vérifiant un pattern.
Contenu d'un fichier `myfile` :

```
toto et gogo
tot et gogo
totoooo et gaga
atoto et titi
titi et tutu
totatata tot toa
```

- `grep "toto$" myfile` : les lignes qui terminent par toto
- `grep "^toto"` : les lignes qui commencent par toto
- `grep -v "toto" myfile` :les lignes ne contenant pas toto

Quantificateurs * et ? (grep -E)

Principe

Les quantificateurs s'appliquent à l'atome précédent :

- un caractère (`a`)
- ou un groupe (`(ta)`)

Signification

- `*` : 0, 1 ou plusieurs occurrences
- `?` : 0 ou 1 occurrence
- `+` : 1 ou plusieurs occurrences

Un cours sur les expressions régulières aura lieu en seconde année.

Quantificateurs * et ? (grep -E)

Exemples (expressions régulières étendues)

- `grep -E "tota*"` → `tot`, `tota`, `totaa`, ...
- `grep -E "tot(ta)*"` → `tot`, `totta`, `tottata`, ...
- `grep -E "toto?"` → `tot`, `toto`

Attention

Les parenthèses et le symbole `?` nécessitent l'option `-E`.

sed

`sed` (**s**tream **ed**itor) permet de modifier ou de supprimer une partie d'une chaîne de caractères, par exemple pour remplacer un caractère par un autre dans un fichier, ou encore supprimer des chaînes de caractères inutiles.

- `sed 's/occurrence_cherchée/occurrence_substituée/g'fichier`
  Option `s` pour substituer, `g` pour appliquer récursivement la substitution.

```
$ sed "s/ //g" myfile # supprime tous les espaces
totoetgogo
totetgogo
...
$ sed "s/o/AAA/g" myfile #remplace o par AA
tAAAtAAA et gAAAgAAA
tAAAt et gAAAgAAA
```

sed

`sed` (**s**tream **ed**itor) permet de modifier ou de supprimer une partie d'une chaîne de caractères, par exemple pour remplacer un caractère par un autre dans un fichier, ou encore supprimer des chaînes de caractères inutiles.

- `sed 's/occurrence_cherchée/occurrence_substituée/g'fichier`
  Option `s` pour substituer, `g` pour appliquer récursivement la substitution.

```
$ sed "s/ //g" myfile # supprime tous les espaces
totoetgogo
totetgogo
...
$ sed "s/o/AAA/g" myfile #remplace o par AA
tAAAtAAA et gAAAgAAA
tAAAt et gAAAgAAA
```

- Normalisation des espaces

```
$ export chaine="un    et        deux" #création d'une variable
$ echo $chaine | sed "s/  / /g" #normalisation des espaces
un et deux
```