Bash Doc

Ko ou Kio ? Go ou Gio ?

A l’origine fût le bit pourvu d’une simple valeur 0 ou 1, puis ces bits furent agrégés, généralement par 8 pour former des octets (le « byte » anglophone).

Avec un octet on ne fait pas grand-chose, on ne stocke presque rien. Rapidement, les fabricants de matériel ont fourni des mémoires, des disquettes, des disques pouvant contenir des milliers, des millions d’octets. Naturellement, ils ont réutilisé les multiples déjà en vigueur, à savoir le kilo, le méga, etc…

On a vu apparaitre alors le Ko (kilo-octets) et le Mo (méga-octets)…. seulement, les informaticiens aimant les puissances de deux, le kilo-octets (Ko) des informaticiens ne faisait pas 1000 octets mais 1024 octets, car c’est la puissance de deux la plus proche de 1000 (1024 = 2 puissance 10). Dans le même esprit, le méga-octets faisait 2 **20 octets, soit 1.048.576 octets et non pas un million d’octets. Les informaticiens ont donc réutilisé le kilo pour indiquer un multiple de 1024 alors que d’habitude un kilo signifie 1000 quelque chose (kilomètre, kilogramme…). Pourquoi pas ? On s’est bien adapté… Mais en 1998 l’International Electrotechnical Commission (IEC) a décidé qu’il fallait clarifier la situation. Ainsi de nouvelles unités ont été créées : *le Kio qu’on prononce « kibi-octets » fait 1024 octets *le Mio qu’on prononce « mébi-octets » fait 1024 * 1024 = 1.048.576 octets *de même pour le Gio (gibi-octets – on ne rit pas !), le Tio (tébi-octets), etc… Résumons : 1 Ko = 1000 octets alors que 1 Kio = 1024 octets Quel est alors le problème ? Il est double : *tout d’abord, les manuels de vos outils informatiques, de vos commandes (fdisk, parted, …) ont-ils été mis à jour ? Quand vous lisez « Go » dans une page de doc, s’agit-il vraiment du multiple de 1000 ou bien est-ce toujours le multiple de 1024 qui est sous-entendu ? En général, en l’absence de mise à jour, les docs parlent en « multiples de 1024″ *par contre, les constructeurs de disque ne se trompent pas eux : quand vous achetez un disque sur lequel on vous promet 1 To de stockage, il faut bien comprendre 10 puissance 12 octets et non pas 2 puissance 40 octets ! On final, vous vous retrouvez avec 931 Mio environ ! On vous aura prévenu : ne criez pas à l’arnaque…quoique…. ===== Commande Less =====** **Less FR**

Voici les commandes les plus courantes :

Clés	Description
↑ ↓	Déplacer par ligne
ESPACE	Descendre d'une page
Nj	Descendre de N lignes
Nk	Remonter de N lignes
b	Monter d'une page
g ou <	Aller à la première ligne
G ou >	Aller à la dernière ligne
Ng	Allez à la ligne N dans le fichier
Np	Allez à la N pourcentage du fichier dans le fichier
Ng	Aller à la Ne ligne
/motif	Chercher le motif à partir de la position
?motif	Chercher le motif à avant de la position
n	Répétez la recherche précédente
N	Répétez la recherche précédente dans la direction inverse
v	Transfère le fichier vers l'éditeur par défaut
q	Quitter le journal

Les raccourcis clavier de la commande less

=====Changer les droits par défaut, la commande umask===== La commande umask (User Mask) permet de changer les droits attribués par défaut. Elle prend en argument un masque constitué de trois valeurs octales qui détermine les droits à supprimer lors de la création d'un fichier par rapport aux droits qui lui sont attribués par défaut, à savoir 666 pour les fichiers ordinaires et 777 pour les répertoires. Sur la plupart des systèmes le masque par défaut est 022, c'est-à-dire qu'il impose lors de la création d'un fichier de ne pas attribuer le droit d'écriture (2) pour le groupe et pour les autres utilisateurs. Ainsi, lors de la création d'un fichier ordinaire qui par défaut devrait avoir les droits 666 soit rw-rw-rw-, le masque 022 empêche de donner le droit w pour le groupe et les autres utilisateur et le fichier est finalement créé avec les droits rw-r–r– , soit 644. De manière analogue, lors de la création d'un répertoire qui devrait avoir les droits 777 soit rwxrwxrwx, le masque 022 a pour effet de le créer avec les droits rwxr-xr-x soit 755. Plus précisément le système calcule les droits à affecter à un nouveau fichier par une opération logique bit à bit entre le masque et les droits : étant donné un masque M et des droits par défaut D, les droits attribués au fichier créé sont le résultat de l'opération logique bit à bit NOT(M) AND D. Pour visualiser cette opération il est nécessaire d'écrire M et D dans leur représentation binaire, chaque bit indiquant la présence (1) ou l'absence (0) d'un droit. Ainsi en prenant comme masque M = 022 et comme droits par défaut D = 666 pour la création d'un fichier ordinaire, on a : *• la représentation binaire de M : 000 010 010 (qui correspond aux droits ––w- -w- à enlever) *• la représentation binaire de D : 110 110 110 (qui correspond aux droits rw- rw- rw-) *• la négation bit à bit de M : NOT(M) = 111 101 101 *• et les droits attribués sont déterminés par l'opération NOT(M) AND D qui donne 110 100 100 (qui correspond aux droits rw- r– r–) ^ ^ r ^ w ^ x ^ ^ r ^ w ^ x ^ ^ r ^ w ^ x ^ |M|0 |0 |0| | 0| 1| 0 || 0| 1| 0| |NOT(M)|1 |1 |1| | 1| 0| 1 || 1| 0| 1| |D|1 |1 |0| | 1| 1| 0 || 1| 1| 0| |NOT(M) AND D|1 |1 |0| | 1| 0| 0 || 1| 0| 0| En appliquant de manière analogue le masque 022 à des droits par défaut 777 pour la création d'un répertoire on obtient les droits 755, soit rwxr-xr-x. Notez que, pour des raisons de sécurité, avec les droits par défaut 666 pour un fichier,

umask ne permet pas de créer des fichiers ordinaires exécutables. Après leur création, il vous faudra ajouter explicitement les droits en exécution pour chaque fichier. C'est la raison qui explique pourquoi les droits par défaut sont à 666 et non à 777 comme ceux des répertoires. Dans l'exemple suivant, on crée un fichier ordinaire vide par la commande touch.

```
$ touch fichier1
$ ls -l
-rw-r--r--    1 john    python     0  2017-02-12 10:09 fichier1
$ mkdir rep1
$ ls -l
-rw-r--r--    1 john    python     0  2017-02-12 10:09 fichier1
drwxr-xr-x    2 john    python  4096  2017-02-12 10:10 rep1/
$ umask 027
$ touch fichier2
$ ls -l
-rw-r--r--    1 john    python     0  2017-02-12 10:09 fichier1
-rw-r-----    1 john    python     0  2017-02-12 10:10 fichier2
drwxr-xr-x    2 john    python  4096  2017-02-12 10:10 rep1/
$ mkdir rep2
```

```
$ ls -l
-rw-r--r--    1 john    python     0  2017-02-12 10:09 fichier1
-rw-r-----    1 john    python     0  2017-02-12 10:10 fichier2
drwxr-xr-x    2 john    python  4096  2017-02-12 10:10 rep1/
drwxr-x---    2 john    python  4096  2017-02-12 10:12 rep2/
```

On peut voir sur cet exemple que le fichier fichier1 et le répertoire rep1 se voient attribuer les droits par défaut déterminés par le masque 022. Après l'exécution de la commande umask 027, les nouveaux fichiers et répertoires ont respectivement les droits 640 et 750, ce qu'on observe pour fichier2 et rep2. Remarquons aussi que les droits des fichiers précédemment créés ne sont pas modifiés. En effet, la commande umask n'a pas d'effet rétroactif. ====== Editeur vim====== editeurvim ====== Raccourci clavier Linux ====== ===== Déplacements ======= CTRL + a ⇒ place le curseur au début de la ligne CTRL + e ⇒ place le curseur à la fin de la ligne (End) CTRL + b ⇒ recule d'un caractère (Backward) CTRL + f ⇒ avance d'un caractère (Forward) Alt + b ⇒ recule d'un mot i.e. place le curseur sur la première lettre du mot sur lequel se trouve le curseur Alt + f ⇒ avance d'un mot i.e. place le curseur après la dernière lettre du mot sur lequel se trouve le curseur ===== Couper / Coller===== Dans les raccourcis suivants, la chaîne de caractères coupée est stockée dans un presse-papier. CTRL + k ⇒ coupe la chaîne depuis le curseur jusqu'à la fin de la ligne (Kill) CTRL + u ⇒ coupe la chaîne depuis le début de la ligne jusqu'au caractère qui précède le curseur CTRL + w ⇒ coupe la chaîne depuis le caractère qui précède le curseur jusqu'au début du mot (si le curseur est placé à la fin d'un mot, coupe le mot) Alt + ←- ⇒ identique à CTRL + w Alt + d ⇒ coupe la chaîne depuis le caractère situé sous le curseur jusqu'à la fin du mot (si le curseur est placé au début d'un mot, coupe le mot) CTRL + y ⇒ colle la chaîne du presse-papier juste avant la position du curseur ===== Modification===== CTRL + t ⇒ inverse la position des deux caractères situés avant le curseur (pratique quand on tape par exemple, sl au lieu de ls) Alt + t ⇒ inverse la position des deux mots situés avant le curseur (pratique lorsqu'on a inversé deux arguments d'une commande) Alt + c ⇒ met en majuscule la lettre située sous le curseur et déplace le curseur à la fin du mot (en plaçant le curseur au début d'un mot, met la première lettre en majuscule) Alt + l ⇒ met en minuscule toutes les lettres depuis la position du curseur jusqu'à la fin du mot Alt + u ⇒ met en majuscule toutes les lettres depuis la position du curseur jusqu'à la fin du mot (en plaçant le curseur au début d'un mot, met le mot en majuscule) CTRL + _ ⇒ annule la dernière modification ===== Autres Raccourcis ===== CTRL + l ⇒ Permet d'effacer le contenu du terminal. (L minuscule) CTRL + c ⇒ En cours de

frappe, permet d’arrêter la saisie de la ligne de commande et de revenir à l’invite avec une ligne vierge. ======= commande Echo ======= =====La commande echo===== <WRAP info >Un signe $ précède les commandes qui ne nécessitent pas de droits administrateur ; un signe # précède celles qui nécessitent des droits administrateur (ces signes ne font PAS partie des commandes). Les lignes qui ne commencent pas par un signe $ ou # correspondent au résultat de la commande précédente.
Les touches utilisées sont indiquées entre crochets, exemple [ctrl] **pour la touche “contrôle”</WRAP> <WRAP tip>Comme de nombreuses commandes basiques, echo peut venir de plusieurs paquets. Elle peut par exemple être incluse dans votre shell (bash par exemple), ou bien venir d'un paquet extérieur (core-utils par exemple), mais grosso modo, elle aura le même comportement dans une utilisation basique.</WRAP> ====Utilisation basique==== La commande** echo **permet simplement d'afficher une ligne. Son utilisation est plutôt simple : <code>$ echo “Ma jolie phrase est belle.”</code> Ceci aura pour conséquence d'afficher sur votre console “Ma jolie phrase est belle.”. Vous pouvez utiliser les guillemets pour contrôler quelque peu le résultat de cette commande : * Avec un guillemet simple de chaque côté du texte, rien ne sera modifié ou interprété par la commande echo : <code>$ echo 'ca va ? oui !' ca va ? oui !</code> * Avec un guillemet double de chaque côté du texte, le texte sera interprété par la commande echo ou par votre shell, voici ce que j'obtiens (la commande n'arrive même pas à se lancer correctement) : <code>$ echo “ca va ? oui !” bash: !“: event not found</code> * Notez que vous pouvez aussi utiliser la commande sans guillemets, mais d'autres interprétations pourront être effectuées (je rajoute par exemple le caractère** * **à mon texte, le contenu de mon répertoire sera affiché : <code>$ echo ca va ? * oui ! ca va ? Desktop Documents fichier1 fichier2 fichier3 oui ! </code> ====Quelques utilisations pratiques==== * Pour écrire à la fin d'un fichier sans en écraser le contenu, on utilise les signes** >> **: <code>$ echo “Ma jolie phrase est belle.” » monfichier</code> * Pour écraser un fichier, en effaçant tout son contenu, on utilise le signe** > **: <code>$ echo “Ma jolie phrase est belle.” > monfichier</code> * Si vous utilisez “sudo” pour obtenir les droits root (comme sur ubuntu par exemple), et que vous vouliez utiliser “echo” pour écrire dans un fichier qui appartient à root, vous devrez ruser car le “sudo” ne survivra pas à la redirection. Ça sera plus clair avec un exemple : Ne fonctionnera PAS :**
<code>$ sudo echo 'www.nouveau_dépôt' » /etc/apt/sources.list</code> Vous obtiendrez un refus avec “bash: /etc/apt/sources.list: Permission non accordée” comme motif. Pour que ça fonctionne il faut utiliser un autre type de redirection, par exemple avec la commande “tee**” qui sert justement à ça.**
Vous pouvez utiliser :

<code>$ echo 'www.nouveau_dépôt' | sudo tee -a /etc/apt/sources.list</code>
L'option “-a**” indique à “tee” d'ajouter la ligne en fin de fichier, sinon le comportement par défaut de “tee” est de remplacer le fichier cible. Ne pas l'oublier !**
*** Pour écrire plus d'une ligne avec “echo”, vous pouvez utiliser un saut de ligne, noté “**\n**”. Pour indiquer à “echo” que ce symbole doit être interprété comme un saut de ligne, il faut utiliser l'option “**-e**” :**

```
$ echo -e '#ceci est un commentaire \nma deuxième ligne' >>
/home/tux/test.txt
```

Vous obtiendrez dans le fichier ”/home/tux/test.txt“ le résultat suivant:

```
#ceci est un commentaire
```

```
ma deuxième ligne
```

Pratique pour ajouter une ligne d'option dans un fichier de configuration, et un commentaire explicatif en même temps.

From:
http://chanterie37.fr/fablab37110/ - **Castel'Lab le Fablab MJC de Château-Renault**

Permanent link:
http://chanterie37.fr/fablab37110/doku.php?id=start:linux:bash:doc&rev=1676817241

Last update: **2023/02/19 15:34**