GNU/Linux Shell Bash

- commande shell bash
- Programmation Bash-fr.pdf
- Introduction à la programmation en Bash FR
- Guide Bash du débutant FR
- Bash

Gestion des Dates et heures sous Linux avec Bash

1. #!/bin/bash
2. man date # affiche le manuel et permet d'obtenir les commandes suivantes
3. date +%H%M # affiche l'heure sous forme 0804 ==⇒ 08h 04mn
4. date +%Y%m%d # affiche la date sous forme 20180609 année mois jour
5. # pour avoir les variables
6. heure=$(date +%H%M) # insere dans la variable “heure” l'heure sous forme 0804
7. jour=$(date +%Y%m%d) # date est une commande donc je change de nom

Utilisation de la commande date

Pense-bête pour l'utilisation de la commande date en bash / shell.

Base

Retourne la date du jour avec les règles de localisation de la machine, par exemple pour une bécane Franco-française :

date retourne mardi 17 novembre 2009, 06:55:32 (UTC+0100)

l'option -d

Ensuite y'a la paramètre -d assez “marrant” qui permet ce genre de chose

date -d 'now' # retourne mardi 17 novembre 2009, 06:57:53 (UTC+0100)

date -d 'yesterday' # retourne lundi 16 novembre 2009, 06:58:32 (UTC+0100)

date -d “tomorrow”date -d “year” # retourne mercredi 18 novembre 2009, 06:58:55 (UTC+0100)

date -d “days” # retourne mardi 18 novembre 2009, 06:59:15 (UTC+0100)

date -d “week” # retourne mardi 24 novembre 2009, 06:59:30 (UTC+0100)

date -d “month” # retourne jeudi 17 décembre 2009, 06:59:59 (UTC+0100)

date -d “year” # retourn mercredi 17 novembre 2010, 07:03:08 (UTC+0100)

Ensuite on peut utiliser des précisions pour le nombre de jour/semaine/mois/année

date -d “3 days” # retourne date 'now' + 3 jours

date -d “2 week” # Retourne la date dans 2 semaines

Bon etc, ça marche pour jour, semaine, mois année, ensuite on peut ajouter le mot ago pour afficher la date passé.

date -d “1 month ago” # retourne la date il y a un mois

Pareil, ça marche pour les jours, semaines, mois et années Mettre en forme la date

Enfin (je vais terminer la dessus), on peut mettre en forme la date un peu à la manière de date() en php

date “+%Y-%m-%d” # retourn ANNEE-MOIS-JOUR

Notez que la chaine de caractère (pattern de format) doit être rédigé de façon assez précise, elle commence par un '+' et les caractères de substitutions sont toujours précédés d'un '%'.

Pour obtenir par exemple en timestamp au format mysql ça donne

date “+%Y-%m-%d %H:%M:%S”

Voici un petit tour rapide des patterns supportés (les principaux) Année

```
  %Y : Année sur 4 chiffres
  %C : Le siècle (en gros les 2 premiers chiffres de l'année, si elle a 4 chiffres...)
```

Mois

```
  %b : Nom du mois sur 3 lettres
  %B : Nom du mois
  %m : Numéro du mois sur 2 chiffres
```

Jours

```
  %a : Nom du jour de la semaine sur 3 lettres
  %A : Nom du jour de la semaine
  %d : Numéro du jour dans le mois sur 2 chiffres
  %j : Numéro du jour dans l' année
```

Heures

```
  %H : Heures sur 24 heures
  %I : Heures sur 12 heures
```

Heures

```
%M : Minutes sur 2 chiffres
```

Secondes

```
%S : Secondes sur 2 chiffres
```

Raccourcis

```
%F : YYYY-MM-DD
%T : HH-MM-SS
```

Voila pour les principaux patterns, un **man date** vous les détailleras tous

Pour finir

Sachez enfin qu'un cumul de -d “durée” et de “+%PATTERN” est possible et cela s'avère parfois assez pratique :

date -d “2 week” “+%F %T” # Retourne un timestamp MySQL du jour qu'il sera dans 2 semaines

Créer des boîtes de dialogues en Bash

Boites de dialogue avec **Whiptail**

Boite de dialogue en Bash

Utilisation de la commande **dialog** sous bash

Commande Dialog

man dialog en Français

doc dialog en francais

Utilisation de **zenity** sous bash

doc zenity

Cours Linux Bash FR

Shell Bash – Niveau débutant Introduction

Les distributions actuelles de GNU/Linux offrent à l’utilisateur un environnement graphique équivalant aux systèmes d’exploitation propriétaires du marché. Mais cette ergonomie et confort d’utilisation sous Linux pour l’utilisateur débutant n’est réellement présente que depuis quelques années. Les distributions actuelles GNU/Linux rivalisent même

d'environnement graphique très divers et avancés avec l'introduction de la 3D et le support de l'accélération des cartes graphiques. Ceci à permis aux systèmes sous GNU/Linux de se démocratiser dans le grand public avec l'apparition des mini-laptop et sur les postes bureautique pour sa sécurité, Linux étant auparavant souvent considéré pour des « érudits » ou experts en informatique.

L'historique des systèmes sous Linux a, au travers des années, gardé cet attachement auprès d'une certaine communauté d'érudits ou d'experts, on peut y voir cet attachement au travers de ce que l'on appel le « Shell » où l'on voit ces dits experts écrire des lignes de textes dans un langage incompréhensif, tout ça dans une fenêtre noire dénuée de tout esthétisme, accompagné d'une série de manipulation des doigts sur le clavier digne d'un pianiste !

Ce cours est donc fait pour ces personnes débutants sous un système GNU/Linux, car ils seront à coup sûre amenés à utiliser ce fameux « Shell », car certaines choses ne peuvent se faire que via le Shell et parce que Linux à besoin d'être proche de son utilisateur, et son utilisateur proche de son Linux, avec tous les avantages et inconvénients que cela impose 😉.

Ce cours ne va clairement pas vous montrer les arcanes dans la manipulations du shell et ses interactions les plus puissantes avec le « Kernel » Linux, le Shell étant aussi un langage de programmation, des années de pratiques étant nécessaires.

Ce cours va juste permettre aux débutants d'appréhender les principes et les fonctions de bases du Shell en temps que logiciel, pour comprendre un peu mieux ces fameuses lignes textes trouvées sur un forum d'érudits Linuxiens et que vous devrez entrer dans votre « terminal ». Définition

Le « Shell » est ce que l'on appel un « Interpréteur de commandes ». Il date de l'époque d' UNIX, où le seul moyen de communiquer avec sa machine était d'écrire des lignes textes au clavier, dans un langage compréhensible à la fois par l'humain et la machine.

Le rôle de la machine étant d'exécuter les commandes de l'utilisateur et d'afficher le résultat à l'écran.

Le shell c'est un programme qui se trouve dans le répertoire /bin.

Par définition, il doit être léger et rapide, et reste un service d'accès privilégié aux Noyau Linux (Kernel) pour l'exécution de primitives système.

Depuis, cette « interface » avec la machine a perdurée car elle est rapide et fiable pour l'utilisateur qui la maitrise. Le shell a évoluée aux travers des années, plusieurs types de Shell existent :

- le /bin/sh shell Bourne

- le /bin/bash shell Bourne Again SHell

- le /bin/csh C shell

- le /bin/ksh Korn shell

- le /bin/tcsh C shell amélioré

mais le principe de base et toujours resté le même : Les shells sont des interpréteurs, ils lisent chaque commande saisie par l'utilisateur (ou lue à partir d'un fichier), vérifient et traitent la syntaxe

pour l’exécuter.

Ici, nous utiliserons le Shell Bash, l’un des plus couramment utilisé sur les systèmes GNU/Linux. Bash est un logiciel libre publié sous GNU GPL. Les bons côtés du shell

- facilité de mise en œuvre et installé d’office sous tous système GNU/Linux (pas besoin d'installer un autre langage sur votre système).

- le shell manipule essentiellement des chaînes de caractères : pas de structures complexes, pointeurs, etc…

- le langage est adapté au prototypage rapide d’applications : exécutions d’instructions systèmes fiables, rapides et robustes. Le bash se révèle un outil puissant lorsqu’on le maitrise.

- c’est un langage « glu » : exécuter et agglomérer des composants divers écrits dans d’autres langages. Les moins bons côtés du shell

- Le nombre de commandes et la documentation difficile d'accès pour le débutant.

- messages d'erreurs parfois difficiles à exploiter, ce qui rend la mise au point des scripts fastidieuse.

- Temps d’apprentissage : la syntaxe est cohérente mais ardue. De nombreuses années d’utilisation sont nécessaires pour être « à l’aise » avec la ligne de commande. Rappels

Avant de lancer un shell, je pense qu’il est nécessaire de rappeler quelques bases bien utiles. Le shell permet avant tout d'exécuter des commandes, d'explorer l'arborescence du système, de créer, d'éditer et de supprimer des fichiers, etc.

Au travers du Shell, vous touchez donc à des parties très importantes et sensibles de votre système d’exploitation, et exécuter des commandes que l’on ne comprend pas peut vraiment avoir des conséquences « catastrophiques » sur ce dernier ! raison de plus pour faire quelques rappels et d’être à l’aise avec ces commandes.

La recommandation principale dans l’utilisation du shell est la « prudence » ! executer des commandes en « root » ne doit s’effectuer que pour des tâches bien specifiques d’administration que l’utilisateur maitrise parafaitement.

Les commandes effectuées dans ce cours peuvent s’effectuer avec un simple compte utilisateur, pas besoin d’être en « root » ! L’arborescence sous Linux

Sous Linux, on ne le rappellera jamais assez, tout est fichier !

Chacun de ces fichiers est placé quelque part en dessous de la racine / (« root »)

Dans /, il faut un ensemble de répertoires systèmes dont la présence de certains est impérative (*), et pour d'autre, préférable.

• /dev : C'est ici que les périphériques (réels et virtuels) sont accessibles (partitions, disques, cartes son, ports SCSI, ports USB, etc.), mais en mode “données brutes”. Il est souvent nécessaire d'utiliser des programmes pour interpréter ces contenus (par exemple, en montant la partition /dev/hda1 dans /mnt/hda1 pour accéder aux fichiers).

• /mnt : C'est en général à cet endroit qu'on accède aux autres systèmes de fichiers

(autres partitions, CD/DVD, clés USB, serveurs de fichiers)

• /etc(*) : Ici sont regroupés tous les fichiers de configurations des différents logiciels installés sur la machine ainsi que des fichiers de configuration système utilisés au démarrage de la machine.

• /media : Certaines distributions montent les périphériques amovibles à cet endroit.

• /var : Fichiers dont le contenu varie o /var/log : On trouve ici les logs des différents logiciels et serveurs. Cela permet de voir ce qui s'est passé quand quelque chose ne va pas.

o /var/spool : Fichiers en cours de traitement (file d'impression, mails en cours d'envoie)

o /var/tmp : Fichiers temporaires (voir aussi /tmp).

• /home : Chaque utilisateur possède son propre répertoire pour y stocker ses fichiers personnels et la configuration des programmes.

• /usr(*) : Répertoire contenant les fichiers du système partageables en réseau et en lecture seule.

• /opt : Répertoire contenant les applications complémentaires (dites : add-on) n'appartenant pas à la distribution installée.

• /tmp : Sont stockés ici les fichiers temporaires (fichiers créés pendant le fonctionnement des logiciels et supprimés à la fin). (Voir aussi /var/tmp)

• /boot(*) : Sont stockés ici les fichiers de démarrage du système (noyau du système, etc.). On y trouve aussi certains fichiers de configuration (GRUB)

• /lib(*) : Ce sont des bibliothèques utilisées par divers programme (C'est l'équivalent des DLL Windows). Par exemple, permet à tous les programme de lire et écrire des fichiers JPEG.

• /sbin(*) : Ce répertoire contient les programmes systèmes et les outils d'administration (par exemple les outils permettant de formater un disque).

• /bin(*) : Ici sont situés les programmes utilisés à la fois par les utilisateurs et les administrateurs. Ensemble de fichiers exécutables représentant les commandes. Lancer un Shell (mode console ou terminal)

Pour accéder à la ligne de commande, il est possible d'utiliser un terminal (xterm, kterm, gterm) ou encore une console virtuelle.

Pour ce faire, plusieurs possibilités sont offertes :

• utiliser le menu du bureau (Gnome, Kde, etc.). Il s'agit de la méthode conseillée.

• utiliser le menu lancer une application. Dans la fenêtre ainsi ouverte, taper le nom de terminal et valider. La fenêtre lancer une application peut être ouverte avec Alt+F2

• Utiliser les consoles virtuelles (il y en a 6). La console virtuelle

La console virtuelle (tty1 à tty6) est un écran noir où une invite de commande apparaît, de la forme login :

Depuis l'interface graphique, il est possible de se connecter à une console virtuelle en utilisant la combinaison de touches Ctrl+Alt+FN, où N est un chiffre de 1 à 6.

Pour revenir au mode graphique depuis une console virtuelle, utiliser la combinaison de touches ALT+F7. Les utilisateurs

Linux a été pensé dés l'origine comme étant sécurisé, multi-tâche et donc multi-utilisateurs (gestion de droits multiples).

Rappelez-vous qu'à une certaine époque, les premiers utilisateurs ne disposaient que d'un seul « ordinateur » sur lequel chacun se partageait des « ressources » (temps processeur, espace disque, mémoire, etc...). Les utilisateurs se connectaient alors via des « terminaux » (écran, clavier, même pas de souris !) pour accéder à l'ordinateur « central » à distance via un « proto-reseau » (au début, une simple liaison série suffisait !), d'où l'interêt d'authentifier chaque utilisateur et processus qui s'y executait.

C'est pourquoi lorsqu'un utilisateur veut utiliser le système sous GNU/Linux, la première étape est tout d'abord de s'authentifier auprés du système, se dernier lui attribuant les droits, des espaces propres qui lui ont été prédefinis et attribués lors de la configuration du système : administrateur, utilisateurs, invité. Connexion dans une console virtuelle

A l'invite login:, saisir l'identifiant (login) de l'utilisateur, puis valider en appuyant sur Entrée, l'invite Password: s'affichera. Saisir le mot de passe de l'utilisateur et valider.

• le mot de passe n'est pas affiché à l'écran pour des raisons de sécurité. Le fait de ne pas voir à l'écran ce que vous tapez, ne doit pas vous inquiéter.

• Si le mot de passe est valide, une invite de commande comme suite doit s'afficher, indiquant que la connexion a réussie. L'invite de commande du shell après la connexion

Suite à la connexion, l'invite de commande (ou prompt) du shell apparaît et a en général la forme suivante :

utilisateur@machine ~ $

• utilisateur - représente l'identifiant ou le nom de l'utilisateur connecté

• machine- représente le nom de la machine sur laquelle l'utilisateur est connecté

• ~ est un raccourci qui signifie le répertoire personnel /home/utilisateur • $ signifie que vous êtes connecté en tant qu'utilisateur

utilisateur@machine ~ #

Si au lieu de $ le signe # apparaît, alors vous êtes connecté en tant que « superutilisateur » (root).

Gardez à l'ésprit que les systèmes GNU/Linux utilisent par convention # pour root et $ pour un utilisateur autre que root.

Ce comportement peut être changé en modifiant la variable d'environnement PS1, mais cela est vivement déconseillé!

Rappel : « root » à tous les droits, celui aussi de casser votre système lors de l'execution d'une commande hazardeuse... pensez donc à rester le maximum avec vos droits utilisateur, la simple

commande exit devrait vous permettre de vous reloguer sous votre utilisateur. Changer d’identité

Depuis votre shell vous avez la possibilité de prendre l'identité d'un autre utilisateur existant sur votre système, y compris l'utilisateur “root” (utilisateur qui dispose de tous les privilèges).

Pour faire cela vous avez à votre disposition la commande su ou su - Regarder utiliser la commande su pour plus de détails. L'utilisation des commandes

Le principe d'exécution des commandes repose sur un principe de fonctionnement simple. L'exécution est séquentielle, les commandes sont exécutées les unes à la suite des autres. Les

commandes peuvent se suivre sur la même ligne mais il faut les séparer par des ; L'execution de la commande s'effectue pas l'appuie de la touche « entrée ».

Exemples:

$ commande1 ; commande2

La suite de commande : % commande1 ; commande2 ; commande3 correspond à la séquence :

$ commande1

$ commande2

$ commande3

Exécution en tache de fond :

$ commande &

Exécution asynchrone :

$ commande1 & commande2

Nous étudierons plus en détails l’utilisation de caractère & dans la partie Pipeline et parallélisme du cours. Quelques commandes de base

Le catalogue de quelques commandes de bases est déjà assez fournit comme vous allez pouvoir le constater. L'intérêt étant d'en connaître le maximum sinon la commande man sera toujours là pour vous aider!

Toutes ces commandes sont « sensibles à la case » (majuscule/miniuscule), le caractère d'espacement servant de séparateur.

cat : Lit (concatène) un ou plusieurs fichier(s), affichage sur la sortie standard

cd : ChangeDirectory, change de répertoire. 'cd' seul permet de revenir dans le home directory chmod : CHangeMODe - change le mode d'accès (permissions d'accès) d'un ou plusieurs fichier(s)

chown : CHangeOWNer - change le propriétaire d'un ou de plusieurs fichier(s) cp : copier des fichiers crontab : planification de tâches

cut : Retire des parties précises de texte dans chaque ligne d'un fichier date : Affiche la date selon le

format demandé

dd : DevicetoDevice - Recopie octet par octet tout ou partie du contenu d'un périphérique (habituellement de stockage) vers un autre peripherique.

df : affichage de la quantité d'espace libre disponible sur tous les systèmes de fichiers du : DiksUsage - l'utilisation de disque

echo : Affiche du texte sur la sortie standard (à l'écran) exit : arrête l'exécution du shell find : recherche de fichiers

fsck : FileSystemChecK - vérification d'intégralité de système de fichiers grep : recherche dans un ou plusieurs fichiers les lignes qui correspondent à un motif groupadd : Ajouter un groupe d'utilisateurs gunzip : décompression de fichiers gzip : compression de fichiers

head : affiche les premières lignes (par défaut 10) d'un fichier help : affiche une aide sur les commandes internes de bash history : affiche l'historique des commandes déjà utilisées kill : envoyer un signal à un processus less : programme d'affichage à l'écran ln : création de liens ls : liste le contenu des répertoires man : Une des plus importantes! affiche les pages de manuel de la commande qui suit.

mkdir : MaKeDIRectory - crée un répertoire mkfs : MaKeFileSystem - création de systèmes de fichiers more : programme d'affichage à l'écran mount : monter un système de fichiers

mv : déplacer, renommer un fichier ps : affiche les processus en cours d'exécution

pwd : Print name of current/working directory - affiche le chemin complet du repertoire courant

rm : suppression de fichiers

rmdir : Remove empty directories - suppression d'un dossier vide

tail : affiche les 10 dernières lignes d'un fichier tar : création d'archives

su : Substitute User identity ou Switch User - prendre l'identité d'un utilisateur uname : Affiche des informations sur le système.

useradd : ajouter un utilisateur whereis : localiser une commande (équivalent à which). Les « man » pages

Comme on l'a vu, le nombre de commandes est assez importante, et encore, nous n'en avons vu qu'une toute petite partie ! Accéder à la documentation en ligne de commande se révèle donc « Essentielle » à tout utilisateur du shell, même à l'expert.

La syntaxe pour appeler la documentation d'une commande est simple : man commande man n commande

n - le numéro de la page man (vous le verrez en haut à gauche)

$ man crontab

En règle générale dans la partie SEE ALSO d'une page de "man", vous trouverez la liste des commandes qu'il est conseillé de consulter ayant un rapport direct avec la commande dont vous lisez le manuel.

Regardez la partie SEE ALSO et vous remarquerez les pages qui sont conseillées de consulter. Cela veut dire qu'on peut taper :

$ man 5 crontab

$ man 8 cron

la touche “q” pour quitter la page man et revenir à l'invite de commande (prompt)

Pour obtenir la description succincte d'une commande, on va utiliser l'option “-f” man -f commande whatis commande

Pour connaître les rubriques qui contiennent dans leur présentation un mot clé, l'option “-k” : man -k commande Manipuler des variables

Après connexion, l'utilisateur est connecté dans son environnement. Cela signifie que le shell met à sa disposition des variables d'environnement, c'est-à-dire un conteneur mémoire dans lequel des données propre à chaque utilisateur ou au système sont stockées. Pour afficher le contenu d'une variable d'environnement, la commande echo $NOM_VARIABLE peut être utilisée.

Le nom des variables d'environnement est par convention en majuscules, il est donc nécessaire de respecter la case pour ces variables. Variables d'environnement à connaître

HOME, USER, GROUPS, UID, PWD, SHELL, PATH, HOSTNAME

• HOME contient le répertoire d'utilisateur

• USER contient le login d'utilisateur

• PWD contient le répertoire courant

• SHELL contient le nom du shell de connexion

• PATH contient la liste des répertoires où se trouvent les commandes que l'utilisateur peut exécuter

• HOSTNAME contient le nom de la machine

• HISTSIZE contient la taille maximale des commandes exécutées contenues dans le fichier historique

• PS1 contient les paramètres d'affichage de l'invite de commande du shell

• PS2 contient le prompt

Exemples:

Récuérer les variables globales d’environnement :

$ env

Récuérer toutes les variables d’environnement + variables locales :

$ set

Donne l'invite de commande du shell :

$ echo $PS1

Donne l'invite du prompt :

$ echo $PS2 Variables locales

Ici, nous allons juste voire comment bash peut se transfromer en petite calculatrice ou executer de petites commandes très simples. Cela nous donnera un aperçu de ce qu'est une variable locale et l'ouverture que cela propose lorsque nous étudierons les « scripts ».

Ici, nous effectuerons juste un affichage « Hello world » et l'addition de 2 variables A et B :

Exemples:

Pour faire afficher au terminal sur l'écran la chaîne « Hello world » au travers d'une variable locale :

$ var='Hello world'

$ echo 'bonjour, ' $var

Maintenant le calcul de A + B :

$ A=1 ;B=2

$ echo "$A+$B"

ne donne pas ce à quoi l'on s'attend… normalement, bash traite les valeurs des variables comme des chaînes de caractères. On peut effectuer des calculs sur des nombres entiers, en utilisant la syntaxe $[1] pour délimiter les expressions arithmétiques:

$ echo $[2] Fichiers de configuration du shell

Au moment de la connexion, dans une console virtuelle ou à l'ouverture d'un terminal en mode graphique, le shell utilise des informations qui se trouvent dans certains fichiers (.bashrc, .bash_profile, etc)

Le comportement du shell peut être modifié en éditant ces fichiers.

Le fichier .bashrc est par exemple utilisé dans le chapitre sur les alias.

Pour ce qui est de la configuration de votre shell vous devez attendre un peu, ça ne sera pas pour tout de suite. Vous apprendrez à le faire avec le temps, donc patience… De la navigation à l'utilisation Naviguer dans l'arborescence du système de fichiers

Dans les systèmes de la famille Unix, la racine représente le sommet de l'arborescence des répertoires

Elle est représentée par le caractère / (slash) et signifie "root" (racine en français).

Tous les répertoires de votre système sont liés à la racine de façon directe ou indirecte.

Concernant les syntaxes à utiliser pour naviguer dans l'arborescence du système de fichiers :

Où suis-je ?

Une chose très importante à savoir quand on est connecté dans un shell, c'est de savoir où l'on se trouve dans l'arborescence :

La commande pwd (PrintWorkingDirectory) affiche votre localisation dans l'arborescence.

$ pwd Le chemin absolu

Le chemin absolu représente l'arborescence complète de fichiers, en partant de la racine Exemple :

Le fichier b.txt se trouve dans /home/user/doc/text

Vous vous trouvez dans /home/user/ascii

Le chemin absolu vers b.txt est donc

Quelque soit votre localisation dans l'arborescence l'utilisation du chemin absolu est le moyen le plus sûr pour accéder au fichier désiré. Le chemin relatif

Le chemin relatif correspond à l'accés à un fichier dans l'arborescence rapporté à votre localisation dans le shell.

On utilise les notations . et/ou ..

. nous permet de descendre dans l'arborescence du répertoire courant : indique le répertoire courant.

.. nous permet dans un 1er temps de monter en arborescence dans le but d'atteindre d'autres répertoires : indique le répertoire parent.

$ cd .. Se deplacer dans l'aborescence

La navigation dans l'arborescence revient donc à utiliser les commandes vues ci-dessus. Mais l'écriture des chemins absolus peut être fastidieux et souvent des erreurs se glissent dans la syntaxe, c'est là que la « complétion de commande » nous vient en aide. Complétion de commandes

Tapez une commande dans un terminal ou ne serait-ce que de se rappeler de sa syntaxe n'est pas une chose toujours facile.

Malgré ça, il existe une touche très pratique sous le shell qui vous permet la « complétion des commandes ».

Supposons que j'ai a taper /usr/bin/tail

La complétion s'obtient en utilisant la touche TAB Pour ça on va commencer avec le 1er caractère

• Je tape /u et j'appuie sur TAB o Le shell va compléter et il va écrire /usr/ o A ce moment j'ajoute un b donc je suis avec /usr/b o J'appuie de nouveau sur TAB et j'aurai /usr/bin/ o A ce moment j'ajoute ta, donc j'aurai /usr/bin/ta

o J'appuie 2 fois sur TAB

Le shell sur mon système trouve 4 correspondances tac tack tail tasksel

Je vais continuer et je vais ajouter un i donc j'aurai /usr/bin/tai J'appuie de nouveau sur TAB et j'obtiens /usr/bin/tail

Ainsi, la complétion nous permet de faire des économies en ce qui concerne l'écriture de la commande et dans le même temps la sûreté de la syntaxe. Donc à utiliser sans modération.
Historique des commandes

Les commandes exécutées sont enregistrées dans un historique.

La variable HISTSIZE contient le nombre maximal des commandes à enregistrer. Vous pouvez accéder à l'historique avec la commande history

$ history n | less

• n - l'option n permet d'afficher les n dernières commandes (facultatif)

• less - la commande “less” vous permet de naviguer dans l'historique o les flêches haut et bas vous permettent de naviguer dans l'historique o !n - permet d'exécuter la commande correspondant au numéro “n” dans la liste sans avoir à la retaper Les raccourcis clavier

cd : revenir dans le répertoire personnel

cd - : revenir dans le répertoire précédent (uniquement si vous avez exécuter un cd)

Ctrl+l : effacer l'écran

Ctrl+c : arrêt d'une commande

Ctrl+z : suspendre(mettre en pause) une commande

CTRL+t : corréction d'une erreur de frappe en inversant 2 lettres

Ctrl+a : aller au début de ligne

Ctrl+e : aller à la fin de ligne

Ctrl+s : interruption de la sortie de terminal (masquer la saisie) Ctrl+q : annuler l'interruption de la sortie (afficher la saisie)

Ctrl+u : efface tout à gauche du curseur

Ctrl+w : efface le mot à gauche du curseur

Ctrl+k : efface le mot à droite du curseur

Ctrl+y : coller la saisie précédente

Ctrl+d : efface le caractère courant, si la ligne est videdeconnexion

Alt+b : se déplacer en avant, mot par mot dans la ligne de commande

Alt+f : se déplacer en arrière mot par mot dans la ligne de commande

Alt+d : efface le mot suivant

Alt+t : échange le mot courant avec le mot précédent

Alt+c : met en majuscule la lettre courante, tout le reste dut mot courant en minuscules, puis se deplace au mot suivant

Alt+l : met en majuscules à partir de la lettre courante jusqu'à la fin de mot, puis se deplace au mot suivant

Alt+u : met en minuscules à partir de la lettre courante jusqu'à la fin de mot, puis se deplace au mot suivant

Alt+Backspace : effacer le mot précédent (équivalent Ctrl+w) Les entrées/sorties et redirection Redirections standards

Une partie qui peut se reveler obscure mais qui est vraiement très pratique lorsqu'on la maîtrise est la notion de « redirection » des entrées / sorties standard.

Pour comprendre, il faut visualiser l'utilisateur devant son clavier et son écran. Il a donc a sa disposition :

• Son clavier : qui lui permet d'insérer des commandes, dans le jargon informatique cela correspond à stdin

• Son écran : qui lui permet de voire s'afficher les commandes traités par son terminal, et qui dans le jargon informatique correspond à stdout, voir même les erreurs eventuelles suite au traitement de commandes par le terminal, correspond à stderr.

Ensuite, chacun de ces 2 périphériques possède un numéro de « fichier descripteur » de 0 à 2

Stdin clavier 0

Stdout ecran 1

Stderr ecran 2

La redirection consiste à rediriger un flux d'information ou une commande vers le bon « descripteur de fichier » en utilisant le symbole > ou < suivant le sens de la direction du flux ou en utilisant d'autres destinations que les descripteurs standards.

Pour réaliser une redirection on utilise la syntaxe :

commande > fichier - redirection en mode écriture vers le fichier, le fichier est créé s'il n'existe pas et son contenu sera remplacé par le nouveau si le fichier existe déjà.

commande » fichier - redirection en mode ajout vers le fichier, le fichier est créé s'il n'existe pas et le résultat sera ajouté à la fin de fichier.

commande < fichier - la commande lit depuis le fichier

Combiné à des commandes de base, ceci permet de réaliser des traitements étonnants ! mais necessite aussi une longue pratique… mais des commandes simples peuvent se révéler déjà très

utiles pour faire un peu de tout.

2>fichier et/ou 2»fichier : pour rediriger la sortie d'erreur standard vers fichier.

> &2 : redirige la sortie standard vers la sortie d'erreur.

2>&1 : redirige la sortie d'erreur standard vers la sortie standard. n>&m : tout ce qui écrit sur le descripteur n sera envoyé vers le descripteur m.

Exemples:

- envoyer le contenu de fichier1 dans le fichier2. Si le fichier2 existe son contenu d'origine sera supprimé, le fichier2 est créé s'il n'existe pas

$ cat fichier1 > fichier2

- envoyer le contenu de fichier1 dans le fichier2 - mode ajout

si le fichier2 existe, le contenu du fichier1 est ajouté à la fin de fichier2, si le fichier2 n'existe pas, il sera créé

$ cat fichier1 » fichier2

- Recherche dans la racine le fichier appelé , les erreurs au lieu d'être envoyées sur STDERR (à l'écran) sont envoyées dans /dev/null (sorte de poubelle sans fin)

$ find / -name ‘’ 2>/dev/null

- Recherche dans la racine le fichier appellé , les erreurs au lieu d'être envoyées sur

STDERR (à l'écran) sont envoyées dans le fichiers

$ find / -name ‘’ 2>

- On veut afficher un message d'erreur, donc utiliser la sortie d'erreur.

echo “$0 : vous n'avez pas le droit d'écrire dans le fichier $1”>$2 Pipeline et parallélisme

Donc en plus des redirection, il existe aussi les commandes pipées ou « pipe » (tune en anglais), enchaînées par un | et des commandes executées en parallèle &. Le shell crée les processus associés à chaque commande, et détourne la sortie du premier sur l'entrée du second.

commande1 | commande2 - le résultat de la commande1 est utilisé par la commande2 commande1 & commande2 - les commandes sont exécutées simultanément, commande1 s'exécutant en arrière-plan

commande1 && commande2 - si la commande1 réussi la commande2 est executée commande1 || commande2 - la commande2 s'exécute seulement si la commande1 échoue commande1;commande2 - les commandes sont exécutées dans l'ordre

Le pipe :

Par exemple, wc est une commande qui calcule le nombre d'octets, de mots, de lignes de son

$ cat *.pas | wc -l

Sans le savoir, cat affiche le contenu sur stdout, mais pas à l'écran, sur le stdin de wc. Sans le savoir, wc lit son stdin et cumule les retours à la lignes \n. Il affiche alors sur stdout le résultat, qui est le nombre de ligne de tous les fichiers .pas d'un répertoire. C'est simple, et pas facile à faire dans d'autres environnements.

Le parallélisme et la dépendance :

On peut lancer une commande en parallèle pendant que la principale continue son execution :

$ echo a & echo b

Pour la dépendance des commandes :

$ commande1 && commande2 && commande3

La commande2 s'exécute ssi le code de sortie de commande1 vaut 0. De même commande3 s'exécute ssi le code de sortie de commande2 vaut 0.

$ commande1 || commande2 || commande3

La commande2 s'exécute ssi le code de sortie de commande1 est diffèrent de 0. De même commande3 s'exécute ssi le code de sortie de commande2 est diffèrent de 0.

Exemple :

$ echo si ok && echo reussi

avec une erreur :

$ ech si ok && echo reussi

Vous remarquez que dans le 1er cas les 2 commandes s'exécutent.

En revanche dans le 2ème cas j'ai fait volontairement une erreur de syntaxe pour la 1ère commande.

Le shell ne regarde même pas le 2ème commande et il s'arrête en nous disant que ech n'est pas une commande connue.

Pour l’alternative :

$ echo si ok || echo reussi

Dans le 1er cas vous remarquez que seulement la 1ère commande s'exécute.

Dans le 2ème cas le shell affiche une erreur pour la 1ère commande mais il exécute quand même la 2ème. Les scripts

Les shells ne sont pas seulement des interpréteurs de commandes mais également de véritables langages de programmation.

Un script correspond à une suite de commandes écrite dans un fichier. Ceci permet d’automatiser

certaines tâches répétitives ou de faire de petits programmes adaptés aux besoins de l'utilisateur.

Ici, nous créerons un script pour le shell bash. 3 étapes suffisent :

• Créer un fichier contenant comme première ligne la syntaxe suivante:

#! /bin/bash

• Rendre ce fichier executable en utilisant la commande:

$ chmod u+x script

• Executer le script en utilisant la commande:

$ script (ou le path s'il ne se trouve pas dans $PATH) ou

$.\script

Exemple :

Créer le fichier test (touch Progtest) et insérer le contenu suivant :

#! /bin/bash# interpréteur commande1 commande2_# même chose que : commande1; commande2

Si on ne veut pas que le fin de ligne soit interprété, il suffit de mettre un "\ " en fin de ligne.

#! /bin/bash commande\

Suite de la commande très longue

Ressources documentaires :

?id=389

#exemple

1)

2)

$A+$B