Commandes de base, Shell, processus, redirection et tubes

Gorgoumack SAMBE

Université de Ziguinchor

Version 1.0[1]

[1] Decembre 2012

Objectifs

A l'issue de ce chapitre, l'apprenant doit être capable de :

- distinguer **console**, **terminal**, **prompt** et **shell**.
- lancer proprement une **commande** linux sur le shell.
- passer correctement des **paramètres** à une commande.
- passer correctement des **options** à une commande
- décrire le **fonctionnement du shell**.
- utiliser des **méta-caractères** dans une commande.
- faire la gestion de base des **processus**.
- comprendre et exploiter les **tubes**.
- comprendre et exploiter la **redirection**
- distinguer tube et redirection.

Plan

Plan

La Console Unix

Console

Une **console** Linux est un **écran noir** en attente d'une **instruction**, d'une **commande** shell.

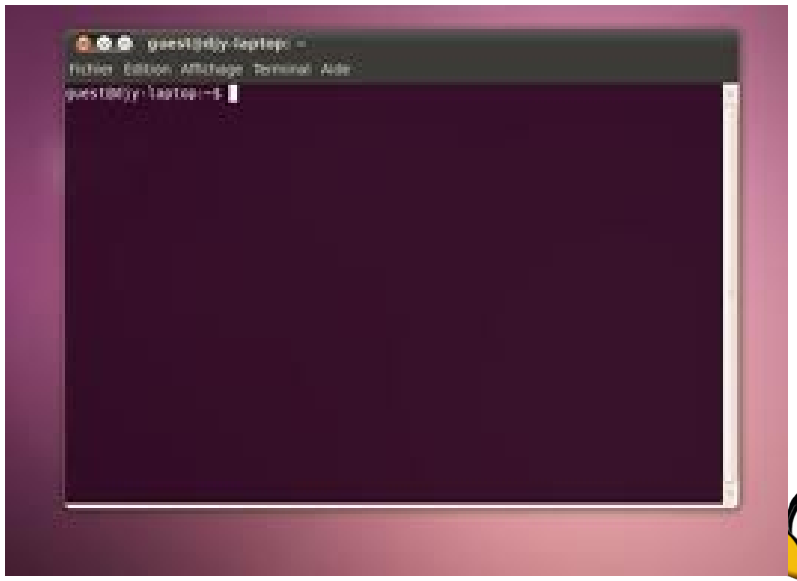
- De manière conventionnelle, Il y a :
  - **6 consoles** disponibles sur la plupart des distributions Linux, elles sont accessible via *Ctrl+Alt+Fx*, x désigne le numéro de la console
  - une ou plusieurs **sessions Xorg(mode graphique)** accessible à partir de F7.

Le terminal

Le terminal

Le terminal est une **émulation** (simulation) de la console **en mode graphique**.

- Vous pouvez lancer un terminal en utilisant la combinaison de touche **Ctrl+Alt+t**.
- Vous pouvez lancer **plusieurs terminaux en même temps**.
- Vous pouvez fermer le terminal actif en utilisant la combinaison de touche **Ctrl+d**.



Invite et interpréteur de commandes

- L'**interpreteur de commande**(CLI pour Command-line interpreter en anglais, **shell**) est le programme qui interprète les commandes passées par l'utilisateur.
  - cf : voir plus loin la partie sur le shell.
- **Invite de commande(prompt)** : ligne de texte indiquant que l'interpréteur est prêt à prendre une nouvelle commande.
  - Exemple d'invite : moussa@machine-moussa:~$
  - est définie par la variable d'environnement **PS1**.

Avantages et inconvénients de la ligne de commande

- **Avantages**
  - **Faible consommation de ressources**, **rapidité** :
    Les interfaces graphiques sont consommatrices de ressources (temps CPU,RAM).
    $\rightrightarrows$ Beaucoup de serveurs (Unix/Linux en particulier) tournent généralement sans interface graphique.
  - **exhaustivité** : Les interfaces graphiques n'affichent pas toujours tous les messages envoyés par le système.
  - **très puissant** avec l'usage des **paramètres** et **options**.
  - **irremplaçable** pour certaines tâches, notamment les **taches d'administration**.
- **Inconvénients**
  - Prise en main difficile ...

Qu'est ce qu'une commande

- Une **commande** est un fichier **exécutable** que l'on lance en tapant son **patronyme** sur une ligne de commande et qui est **destiné à réaliser une action précise**.
  Exemples :
  - *pwd* : affiche le chemin absolu du répertoire de travail.
  - *date* : affiche la date et l'heure.
  - *ls* : affiche le contenu d'un répertoire.
  - *firefox* : lance le navigateur firefox.
- **/bin** : contient les **commandes de base** Linux (*man, cd, ls, pwd, cp, mv, mount, umount*...)
- **/sbin** : contient les **exécutables système**(comme *adduser*).
- Lorsqu'une commande est lancée, elle est cherchée parmi les répertoires situés dans la **variable d'environnement PATH**.

Structure générale d'une commande

nomCommande *options* `paramètre1 ...paramètreN`

- Une commande est généralement constituée d'un **nom** suivi d'une liste d'(éventuelles) **options** et d'une liste d'(éventuels) **arguments**.
- séparateurs : espaces et tabulations.

Exemples :

- **ls**
- **ls** *-a*
- **ls** *-i -l*
- **ls** *-ial*
- **ls** /bin /usr
- **ls** *-ial* /bin /sbin

Paramètres d'une commande

- Les **paramètres** désignent généralement les éléments auxquels s'appliquent la commande : fichiers, répertoires, utilisateurs, commandes, . . .
  Exemples :
  - **ls** /usr/bin : liste le contenu du répertoire passé en paramètre.
  - **firefox** www.univ-zig.sn : ouvre la page avec cette url sur le navigateur.
  - **touch** monfic : modifie la date de modification du fichier.
    - crée le fichier s'il n'existe pas
  - **cat** monfic : affiche le contenu de monfic.
  - **sudo** ls : exécute la commande ls en tant que root.
  - **adduser** assane : crée le compte utilisateur de login assane
  - **man** whereis : affiche l'aide de manuel de la commande whereis.

Options d'une commande

- Les options permettent de **modifier le fonctionnement** d'une commande.
- Elle se place **à la suite de la commande** et commencent généralement par un **tiret**.
- Certaines options peuvent à leur tour **prendre des arguments**.
- La majorité des options, sous Unix, est composée d'un tiret et d'une seule lettre : -l -a -i
- Il est possible de **combiner les options** : -ia, -il, -aux, -ax.
- certaines options à **nom long** sont en général précédés de **deux tirets** : –version

Options d'une commande

Exemples

- *ls -l* : affichage détaillé(long)
- *ls -ai* : afficher tous les fichiers(all) avec leurs inodes(inoeud)
- *date –version* : affiche les informations de version de la commande date.
- *ps -u emilie*: affichage des processus de l'utilisateur emilie(user).
- *df -HT* : espace disque libre en MO/GO(Human readable) avec les types de systèmes de fichiers(T)
- *cal -y* : affiche le calendrier annuel.
- *cal -y 1990* : affiche le calendrier de l'année 1990.

Quelques commandes utilitaires

- *df* : usage de l'espace disque des différents systèmes de fichiers.
- *cal,ncal* :affichage du calendrier
- *whoami* : affiche le nom de l'utilisateur connecté.
- *clear* : efface l'écran
- *date* : affiche la date et l'heure
- *echo* : affiche le texte passé en paramètre.
- *id* : affiche les informations sur l'utilisateur.
- *nomcommande –version* : Version d'une commande.
- Aide sur les commandes :
  - *nomcommande –help* : Aide courte sur les commandes.
  - *man nomcommande*: Aide sur les commandes et leurs options.

Quelques commandes sur les fichiers

- *cat* : Affiche à l'écran le contenu d'un fichier
- *more* et *less* : Affiche à l'écran le contenu d'un fichier, page par page.
- *cp* : Copie de fichiers.
- *rm* : Suppression de fichiers
- *mv* : Déplacement de fichiers. Renommage de fichiers.
- *touch* : change l'heure de dernière modification du fichier
  - **crée le fichier s'il n'existe pas**.

Quelques commandes sur les répertoires

- *pwd* : Affiche le chemin absolu du répertoire de travail
- *cd* : Permet de changer de répertoire de travail.
- *mkdir* : Pour la création de répertoires
  - option -p : pour créer en même temps les dossiers parents
- *rmdir* : Pour supprimer des **répertoires vides**.
- *rm -r* : Suppression récursive de répertoires.
  - -f: sans demande de confirmation
- *ls* : Liste le contenu d'un répertoire
  - option *-a* affiche les fichiers cachés
  - option *-l* pour un affichage long
  - option *-i* pour afficher les inodes

Plan

Le Shell ou interpréteur de commandes

- Le Shell (coquille): programme qui **interprète** les **commandes** passées par l'utilisateur.
- Une commande n'est **pas transmise directement au noyau**. L'interpréteur analyse la ligne, **réalise certaines tâches** avant d'exécuter.
- Le Shell permet :
  - la **substitution de noms** de fichiers et répertoires.
  - la gestion des **variables d'environnement**.
  - l'**exécution des commandes**.
  - la **redirection** des entrées et des sorties.
  - la gestion des **tubes**(pipe).
  - la réalisation de **scripts**(chapitre 5).

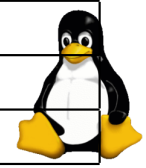
Les différents Shell

- Il existe de **nombreux Shells**, avec des **syntaxes** et **richesses** différentes
  - **sh (Bourne Shell)**: Toujours présent, moins d'efficacité.
  - **bash (Bourne Again Shell)**: sh amélioré. Installé par défaut sous Linux.
  - **csh (C. Shell)**: Proche du langage C en programmation
  - **tcsh (Turbo C. Shell)**: Extension du C.shell sous Linux.
- Le Shell interprète les commandes en gérant des **caractères spéciaux(méta-caractères)**.

Le Shell et les méta caractères

- Les Caractères spéciaux.

Caractères	signification
Retour chariot($\backslash r$)	**fin de ligne** de commande.
Espace-tabulation ($\sqcup$, $\backslash t$)	**séparateur** des options et arguments.
$*?\sim$ [][^]	Caractères (**jokers**) de substitution des noms de fichiers
$	**Référence** au contenu d'une variable d'environnement
;	Séparateur de commandes
$\sim$	Chemin absolu du répertoire de travail de l'utilisateur
!	Caractère utilisé pour la gestion du **rappel de commandes**.
" ' `	Délimiteurs
&	**Exécution en arrière plan** d'une commande
< >	**Redirection** des entrées/sorties.
|	**Tubes**
(),{}	Délimiteurs
\	Caractère de **dé-spécialisation**.



Substitution des noms de fichiers et répertoires

Les joker

- Les joker sont interprétés avant exécution de la commande.

Caractères	signification
?	N'importe quel caractère.
*	N'importe quelle suite de caractères
∼	Chemin absolu du répertoire de travail de l'utilisateur
[]	N'importe quel caractère parmi ceux spécifiés.
[^]	N'importe quel caractère n'apparaissant pas dans les crochets.

- *echo ab** : affiche tous les fichiers dont le nom commencent par ab
- *echo [ab]cd**: affiche tous les fichiers dont le nom commence par a ou b suivi de cd et se terminant par n'importe quelle suite de caractères.
- echo [^f]cd??: tous les fichiers dont le nom commence par autre chose que f suivi de cd et de deux caractères quelconques.
- echo $[AabB]c[^q - z] * .d?c$: ???

- Un caractère précédé d'antislash \ n'est pas interprété.
- A l'intérieur des cotes (' ') aucun caractère n'est interprété.
- A l'intérieur des guillemets(" "),seuls $,‘ et ! sont interprétés .

Variables d'environnement

- L'**environnement** désigne le **contexte** d'exécution d'un programme, c'est un ensemble de **paramètres**(**variables**) contenant des valeurs de type **chaîne de caractères**.
- Exemple du **shell** : couleur de terminal, shell utilisé, invite de commande, langue, path, ...
- le shell utilise en réalité **deux environnements** différents :
  - son propre environnement, qui contient les variables d'environnement locales à la session du shell en cours ;
  - l'environnement d'exécution, dont les **variables d'environnement** sont **transmises aux programmes** que le shell lance.
- Le shell possède en réalité un **langage de script**. vous pouvez définir **vos propres variables**.

Variables d'environnement courantes

Nom	Signification
HOME	Chemin du répertoire personnel de l'utilisateur.
USER,LOGNAME	Nom de login de l'utilisateur.
TERM	Type de terminal utilisé.
SHELL	Chemin sur le fichier de programme du shell actuellement utilisé.
PATH	Liste des répertoires dans lesquels les programmes à exécuter seront recherchés.
LANG	Nom de la locale à utiliser par défaut pour les paramètres d'internationalisation des applications.

Variables spéciales

- $$: **Identifiant** de processus(PID) du **shell courant**.
- $! : **Identifiant**(PID) de la **dernière tâche** lancée **en arrière plan**.
- $? : **Code retour** de la **dernière commande**.
  - 0: succés
  - un nombre strictement positif : Échec

Variables locales

- **Accès au contenu** d'une variable: $NOM_VARIABLE.
  - echo $SHELL $PS1
  - echo mon répertoire de travail : $HOME
- **Affectation** d'une valeur à une variable :
  - var=val
  - chaine='une chaîne,pas de problème avec les métacaractères($,`,|,...)'
  - bienvenue=Bonjour\ $USER,\ bienvenue.
- ${#*var*}: **Longueur** du contenu de la variable var
- l'anticote (`) est un **délimiteur de commande** :
  - echo `date`
  - bienvenue="Bonjour $USER, nous sommes `date`"

Plan

Processus

- Un **processus** est une **instance d'exécution** d'un **programme**.
- Tout **processus** est identifié par un **numéro unique**: le **PID**.
- Tout processus est enfant d'un autre processus.
- Le processus **init** est l'**ancêtre** de tous les processus. Il est lancé par le noyau au démarrage du système.
- Le processus lancé au login est le père de tous les processus lancés par la suite par l'utilisateur, et il ouvre trois fichiers:
  - L'entrée standard, la sortie standard et la sortie erreur standard.
- Un utilisateur ne peut arrêter que les processus qui lui appartiennent.
- Seul l'administrateur système(root) a le droit d'arrêter un processus ne lui appartenant pas.

Quelques commandes : ps,top et kill

- La commande ps (process status) permet de lister les processus actifs
  - Options: -*e*: tous les processus.
    -*f* :affichage détaillé. Voir l'aide man.
    $> ps -ef

UID	PID	PPID	C	STIME	TTY	TIME	COMMAND
root	1	0	0	Dec 6	?	1:02	init
...							
jean	319	300	0	10:30:30	?	0:02	/usr/dt/bin/dtsessio
olivier	321	319	0	10:30:34	ttyp1	0:02	csh
olivier	324	321	0	10:32:12	ttyp1	0:00	ps -ef

  - **UID**: nom de l'utilisateur qui a lancé le processus. **PID**: numéro du processus. **PPID**: numéro du processus père. **C**: facteur de priorité. **STIME**: heure de lancement du processus. **TTY**: nom du terminal. **TIME**: durée de traitement du processus. **COMMAND**: nom du processus.
- la commande top affiche les tâches avec une mise à jour régulière de l'affichage.
- la commande kill permet d'arrêter un processus
  - kill numéro_PID

Processus en série, Processus en arrière-plan

- Enchaînement de **processus en série**(Exécution séquentielle).
  - Un processus s'exécute et lorsqu'il termine, le suivant démarre.

    Ex: date ; who ; ls ; ps
- Lancement de processus en **arrière plan**
  - Le **shell n'est pas bloqué** durant l'exécution du processus.
  - Possibilité de **lancer plusieurs programmes** à partir d'une même console.
    Ex: gedit&
  - Le système répond en indiquant le **numéro du processus** créé.
  - Mêmes entrée et sortie erreur standard que son processus père.
  - Ce processus ne peut plus lire de données à partir de l'entrée standard
- Lancement d'une série de processus en arrière plan
  Ex: (gedit ; echo 'Fin de l'édition avec gedit')&

Contrôle des tâches

- Ctrl-C : arrête le processus courant (kill).
- Ctrl-Z : met en arrière plan le processus courant.

- jobs : affiche les tâches qui sont en arrière-plan.
- bg : continue l'exécution en arrière plan d'une tache qui est dans l'assiette(Ctrl-Z)
- fg : continue l'exécution en avant plan d'une tâche qui est dans l'assiette(Ctrl-Z)

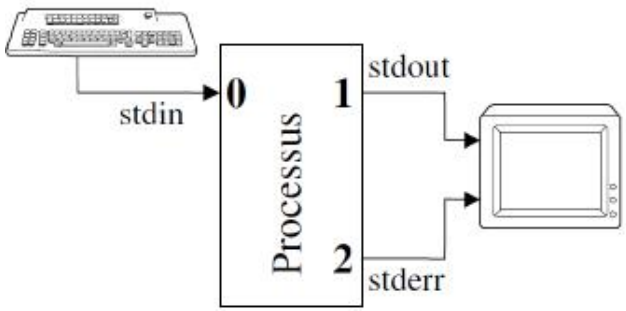
Plan

Entrée/sortie Standard, sortie d'erreur

- **Convention**: Tout processus doit lire ses données sur l'**entrée standard** et renvoyer ses résultats sur la **sortie standard**.

			Langages			
Fichier spécial	Correspondance	Descripteur	Shell	C++	C	Java
stdin	Entrée standard	0	read	cin	scanf	System.in
stdout	Sortie standard	1	echo	cout	printf	System.out
stderr	Sortie erreur	2	echo	cerr	stderr	System.err

- Par défaut : stdin = clavier; stdout = écran; stderr = écran.



Redirection

Ces 3 entrées/sorties peuvent être redirigées afin que la lecture/écriture se fasse à partir/vers d'autres fichiers.

- Redirection de l'entrée standard: $<$
  - *commande* $<$ *fichier*

    La commande va lire ses données à partir du fichier
- Redirection de la sortie standard: $>$,$>>$
  - *commande* $>$ *fichier* :

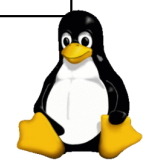
    Si fichier n'existe pas il est créé. Sinon il est écrasé. Le résultat de la commande est mis dans fichier.
  - *commande* $>>$ *fichier* :

    Si fichier n'existe pas il est créé. Sinon le résultat est ajouté à la fin de fichier (mode append).
- Redirection de la sortie d'erreur: 2 $>$, 2 $>>$
  - commande 2 $>$ fichier

    Si fichier n'existe pas il est créé. Sinon il est écrasé. Les messages d'erreur de la commande sont mis dans fichier.
  - commande 2 $>>$ fichier ???

Exemples

Commande Shell	Description
sort	Trie les lignes tapées au clavier; écrit le résultat trié à l'écran.
sort < f1	Trie les lignes du fichier f1; écrit le résultat trié à l'écran.
sort > f2	Trie les lignes tapées au clavier; écrit le résultat trié dans le fichier f2. (si f2 existait avant, il est effacé et recréé).
sort >> f2	Trie les lignes tapées au clavier; ajoute le résultat trié à la fin du fichier f2 (si f2 n'existait pas avant il créé).
sort 2 > f3	Trie les lignes tapées au clavier; écrit le résultat trié à l'écran. Les messages d'erreur éventuels sont écrits dans le fichier f3.
sort <f1 >f2 2>f3	???
sort <f1 2>>f3	???
wc <f2	compte le nombre de lignes/mots/caractères du fichier f2
wc -l <f2	compte le nombre de ligne du fichier f2



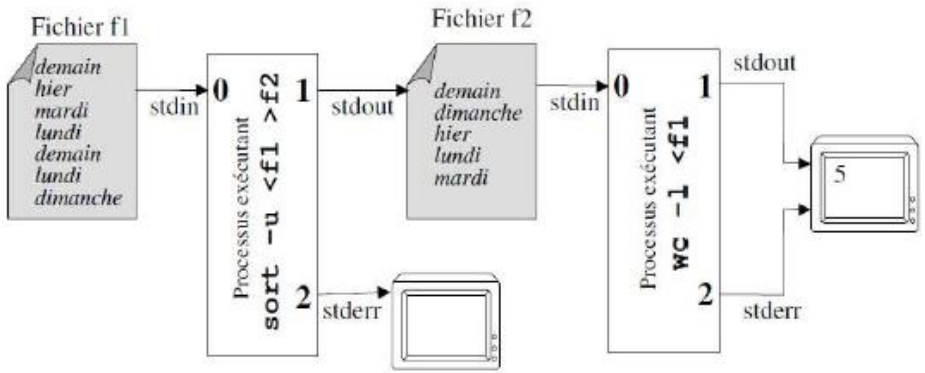
Périphériques virtuels

- Des Zéros et des Nulls: deux fichiers pratiques
  - Utilisation de /dev/null
    Un "trou noir". Un fichier en écriture. Tout ce qui y est écrit disparaît à jamais. Toute tentative de lecture n'aboutira à rien. Peut être très utile en ligne de commande et dans certains script.
  - Utilisation de /dev/zero Pseudo périphérique spécial qui renvoie une infinité de caractères null (ASCII NUL, 0x00) lors d'une lecture. Souvent utilisé pour fournir un flux de données (écraser des informations ou initialiser un fichier).

Tubes (pipe)

- Problème posé : compter le nombre de mots uniques dans une liste .

```
sort -u <f1 >f2    # Trie l'entrée standard en laissant
                   # une occurrence unique de chaque ligne.
wc -l <f2          # word count : affiche sur la sortie standard
                   # le nombre de lignes dans l'entrée standard
```



Tubes (pipe)

- Mécanisme permettant de relier la sortie standard d'un processus à l'entrée standard d'un autre, sans création de fichier intermédiaire.
- sort -u <f1 — wc -l