INSTITU SUPERIEURE DES SCIENCE APPLIQUEES ET DE TECHNOLOGIE DE GAFSA

LARI2/LAII2

Ministère de l'enseignement supérieur et de la recherche scientifique

Travail en ligne de commande Linux

Certification Linux professional Institute

LPIC1

Objectif 103.1

Objectifs 103.1 (poids 4)

✓ Utilisation de commandes ou de séquences de commandes pour réaliser des tâches simples en ligne de commande.

✓ Utilisation et modification de l'environnement du shell, en particulier la définition, l'export et le référencement des variables d'environnement.

✓Utilisation et édition de l'historique des commandes.

✓ Exécution des commandes comprises ou non dans le chemin (path) par défaut.

Comprendre les bases de la lignes de Commande

Interface de connexion

- Chaque système d'exploitation possède une interface pour communiquer avec les utilisateurs.

- GNU/Linux possède deux interfaces de connexions:

1) l'interface graphique également appelée mode graphique.
2) L'interface texte également appelée mode commande.

Interface des ligne de commande (ILC)

- Pour l'administration d'un système GNU/Linux on utilise généralement l'interface des lignes de commande.

- L'administrateur ou l'utilisateur dispose alors d'une **invite des commandes** ou un **prompt** qui, dans sa forme la plus simple est représenté par le caractère $ (pour un utilisateur) ou # (pour l'administrateur). Cette interface est appelé en anglais: command line interface (CLI).

Terminal virtuel ou Console

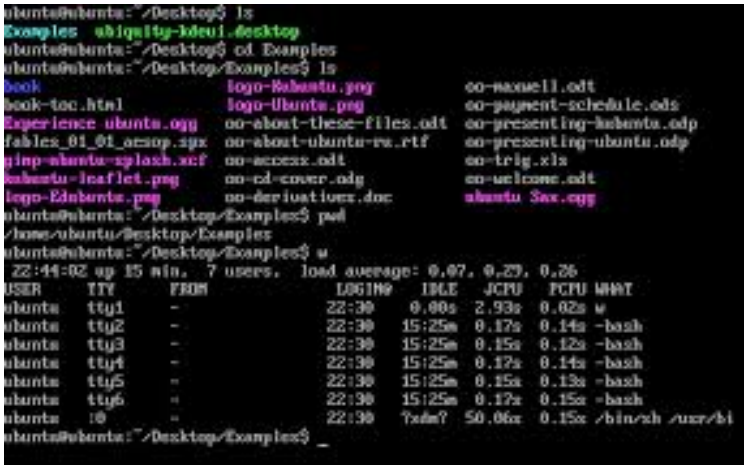
- La ligne de commande fonctionne normalement au sein d'un terminal (écran + clavier), mais aujourd'hui on parle d'émulateur de terminal ou terminal virtuel s'exécutant sur le poste de travail (PC).

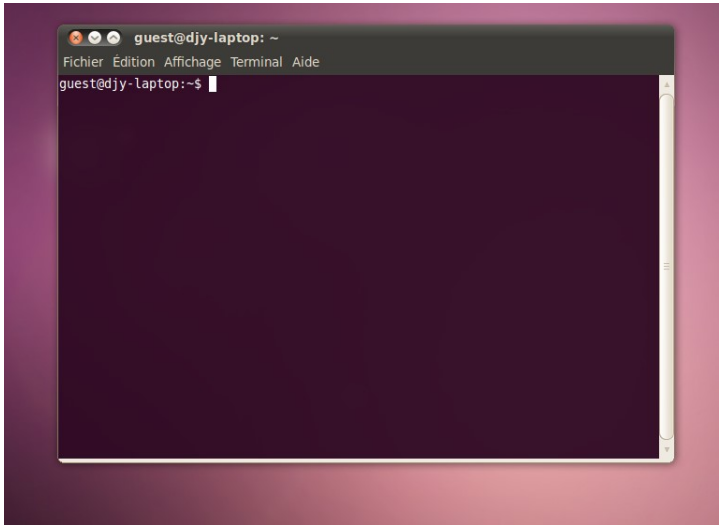
- Il y'a deux types de terminaux virtuels:

1) **Terminal virtuel texte (TVT)**: c'est le terminal par défaut lorsque Linux démarre ou fonctionne sans environnement graphique. (distribution serveur)

Remarque: A partir de l'environnement graphique, on peut ouvrir un TVT avec la combinaison de touche: ALT-CTRL-FX (avec X=1-6).

2) Terminal virtuel graphique: C'est un émulateur de terminal au seins d'un environnement graphique. Exemple: *Xterm, Konsole, Gnome-terminal*

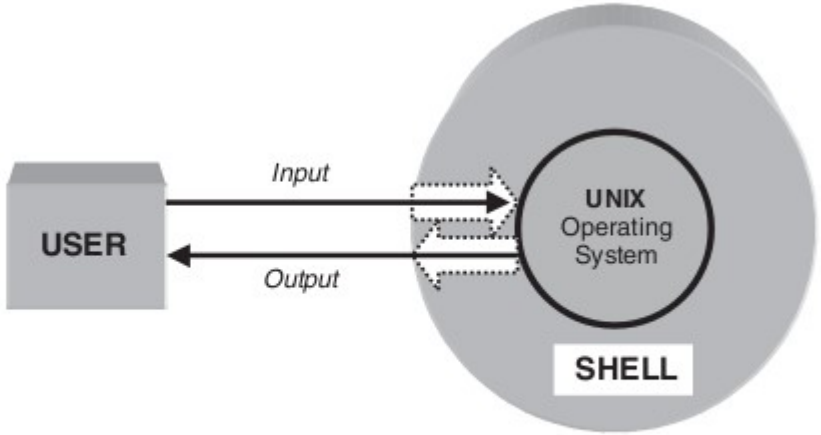




Shell interactif

- C'est le job d'un programme interactif appelé **Shell** de fournir le prompt et d'interpréter les commandes.

- On peut imaginer le Shell (coquille) comme une couche entre l'utilisateur et le noyau du système d'exploitation (S.E).



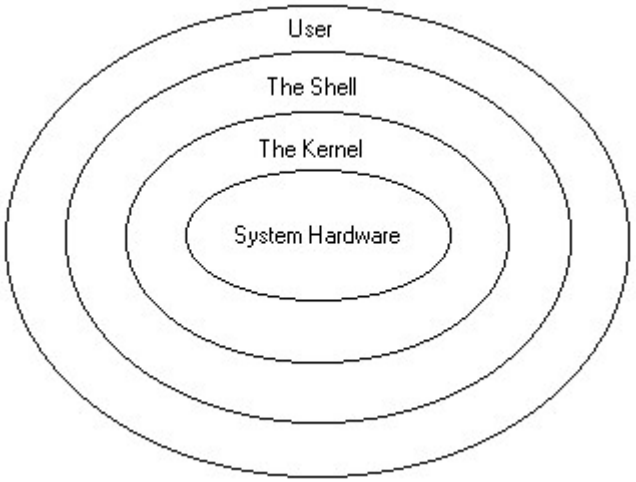


FIGURE 3.1
The user's shell layer.

- shell de login: Si la connexion au S.E est établit en mode commande, alors un premier shell (applé login shell) est lancé automatiquement par le programme **login** après l'authentification sur le système (c.a.d, après avoir saisi à l'invite d'identification un nom d'utilisateur et un mot de passe associé valides).

Les variantes du shell

- Dans une distribution, plusieurs shells sont disponibles et sont plus au moins similaire. La liste des shells disponible se trouve dans le fichier /etc/shells. En voici quelques exemples:

- sh: Le Bourne-shell (**sh**), crée par Stephen Bourne pour Unix. Il est l'ancêtre commun de tous les shells. sh est obsolète en interactif mais encore utilisé en programmation.

- Le C-shell (**csh**) d'origine BSD, et sa version plus récente le tc-shell (**tcsh**), est utilisable en interactif mais non compatible avec le Bourne shell (sh) en programmation.

- **Le Korn-shell (ksh)** est l'interpréteur de commandes le plus répandu dans le monde Unix. Il est très agréable en interactif et compatible ascendant en programmation avec le Bourne-shell.

- Le Bourne Again Shell (**bash)** est l'interpréteur de commandes le plus répandu dans le monde Linux. Le Bash est compatible avec le sh. Il inclut des caractéristiques du ksh et du csh. Il offre des améliorations aussi bien en interactif qu'en programmation.

Note: ***bash est le shell sur lequel se base la certification LPI.***

Lancement d'un shell

- Le lancement d'une console virtuelle en mode graphique dépend de la distribtion installée. Par exemple: sous Ubuntu taper **ALT-CTRL-T**, ou taper dans la barre de recherche gonme-terminal.
- Il est possible aussi d'ouvrir un terminal virtuel texte (**TVT**) en tapant **ALT+CTRL+une touche de fonction**, Exemple: **ALT+CTRL+F1**
- Il y a d'habitude 6 consoles virtuelles allant de F1 à F6 (Ubuntu) et F2 à F7 (Fedora)
- La commande **ALT+CTRL+F7 ou F1** sert à revenir à l'environnement graphique. (**GUI**)

- On basculer d'un shell à l'autre simplement en tapant la commande correspondant à l'abréviation du nom du shell choisit. Exemple:

$ sh

$ csh

$ exit ← permet de revenir au shell précédent

- La commande **exit** termine le shell courant. Lorsqu'elle est saisi au niveau du login shell, elle le termine est provoque la déconnexion.

Syntaxe générale des commandes

- La structure d'une commande est:

Commande [paramètres] [arguments]

- ✓ Une commande peut avoir ni paramètre ni arguments.

- ✓ Un paramètre est une option de la commande qui modifie son fonctionnement.

- ✓ une option est généralement une lettre ou un nombre précédé par un tiret -. Il y aussi des options avec un libellé plus long. Elles sont alors précédés par un double tiret --, et ont généralement une version courte associée.

Exemple:

$ ls -all

est équivalent à

$ ls -a

Remarque: Plusieurs options peuvent être aligner ensembles. Exemple:

$ ls –a –i –l

est équivalent à

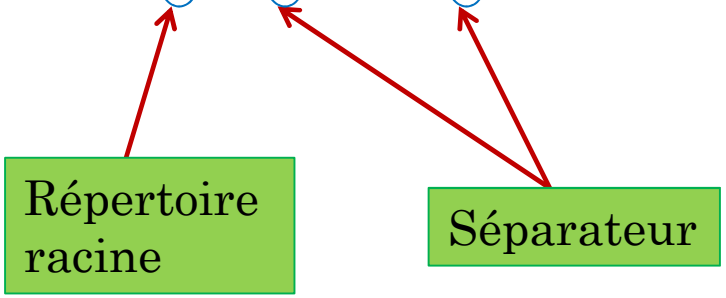
$ ls –ail

Remarque: l'option **--help** affiche toutes les options de certaine commandes.

- Les arguments sont les chemins d'accès de fichier.
- Un fichier peut être localisé dans l'arborésence du système de fichier en utilisant l'un des chemins d'accès suivant:

1. Le chemin d'accès absolu (c.a.a): Il est définit par rapport au répertoire racine /, par exemple:

   /usr/share/doc/latex/readme



2. Le chemin d'accès relatif (c.a.r): Il est définit par rapport au répertoire courant de travail.

Rq: Deux fichiers sont ici important représentés par

- ✓ . : Le répertoire courant
- ✓ .. : Le répertoire parent du répertoire courant.

Commandes internes

- Les commandes **internes (built-in)** du bash sont ceux qui sont intégrées dans le code binaire du shell et se trouvent alors toujours dans la mémoire centrale. Elles ne peuvent être détruites.

- Une liste non exhaustive des cmds internes est: **., enable, pwd, cd, set, alias, echo, exec, exit, export, history, jobs, kill, newgrp, source, type, umask, unalias, unset.**

- Deux exemples de commandes internes importantes sont:

1. **pwd**: (print working directory) permet d'afficher sur l'écran le répertoire dans le quel l'utilisateur se trouve.
2. **echo:** Affiche le texte donné en argument sur l'écran.

Exemple:

$ pwd

$ echo *je suis étudiant en informatique*

Commandes externes

- Les autre commandes (externes), telle que **ls** sont stockées sur disque dans des fichiers exécutables (programmes binaires), généralement rangés dans les répertoire **/bin** et /**usr/bin**. Les fichiers qui représente ces commandes peuvent être détruits.

- Note: On peut distinguer les cmds on utilisant la cmd interne **type**. Exemple:

$ type ls

$ type pwd

Les alias de commande

- Les alias permettent à chaque utilisateur de créer ces propres cmds a partir de cmds existantes.

- Donc un alias peut être utiliser pour créer un raccourcis pour une cmd complexe ou longue.

- Deux cmds gèrent les alias:

$ alias *nom_alias='commandes'* ← définit ou modifie un alias dont le nom est nom_alias.

$ alias ← affiche la liste des alias déjà définis.

$ unalias *nom_alias* ← supprime l'alias dont le nom est nom_alias.

Exemple:

$ alias rm=‘ rm –i’

$ alias

$ unalias ls

Note bien: Lorsqu'une commande est saisie, le shell recherche dans l'ordre: un alias, une cmd interne (builtin), une cmd externe. La première trouvée est exécutée.

Les variables d'environnement

- Le bash dispose de plusieurs variables modifiables par l'utilisateur. Leur rôle est primordial car elles contiennent les information utiles aux cmds et logiciels exécutés durant la session de travail.

- Une variable du bash joue le même rôle qu'une variable dans un langage de programmation.

- Les variables du bash sont toutes du type « chaines de caractères ».

- Pour afficher les variables d'environnement du bash on utilise la cmd **printenv**.

$printenv ← affiche toutes les variables d'environnement du bash

Exemple de variables d'environnement sont:

- PATH: contient une liste de répertoires dont lesquels le shell recherche les exécutables.
- HOME: Chemin d'accès du répertoire personnel.
- PS1: invite de commande numéro 1

- En Bash, un utilisateur crée ou modifie une variable en affectant directement une valeur à cette variable par l'opérateur « = ».

$ classe=LAII2 ← classe est le nom de la variable alors que LAII2 est sa valeur

N.B: Il n'y aucun espace dans cette ligne de cmd

- Pour afficher la valeur d'un variable on utilise la cmd **echo**.

$ echo $classe ← Affiche la valeur de la variable classe

N.B: Pour utiliser une variable on doit la précéder par le signe $

Exportation d'une variable locale

- Une variable définit dans un environnement de travail (shell) bash n'est disponible que dans celui-ci. On dit que c'est une variable locale.

$ bash ← exécuté un nouveau bash (processus enfant).

$ echo classe ← la variable classe n'est pas définit dans le processus enfant.

- Pour rendre une variable (locale) visible aux autre environnement de travail on doit **l'exporter**.

- La commande qui exporte une variable est:

$ **export *variable***

Par exemple:

$ export classe ← Exporte la variable classe. La variable locale classe est devenu une variable d'environnement.

Remarque: Si vous fermez l'environnement de travail (shell) vous perdez les variables précédemment définis. Pour qu'ils deviennent permanent, il faut les définir dans les fichiers de configuration du shell. Ceci est aussi valable pour les alias.

Fichier de configuration du bash

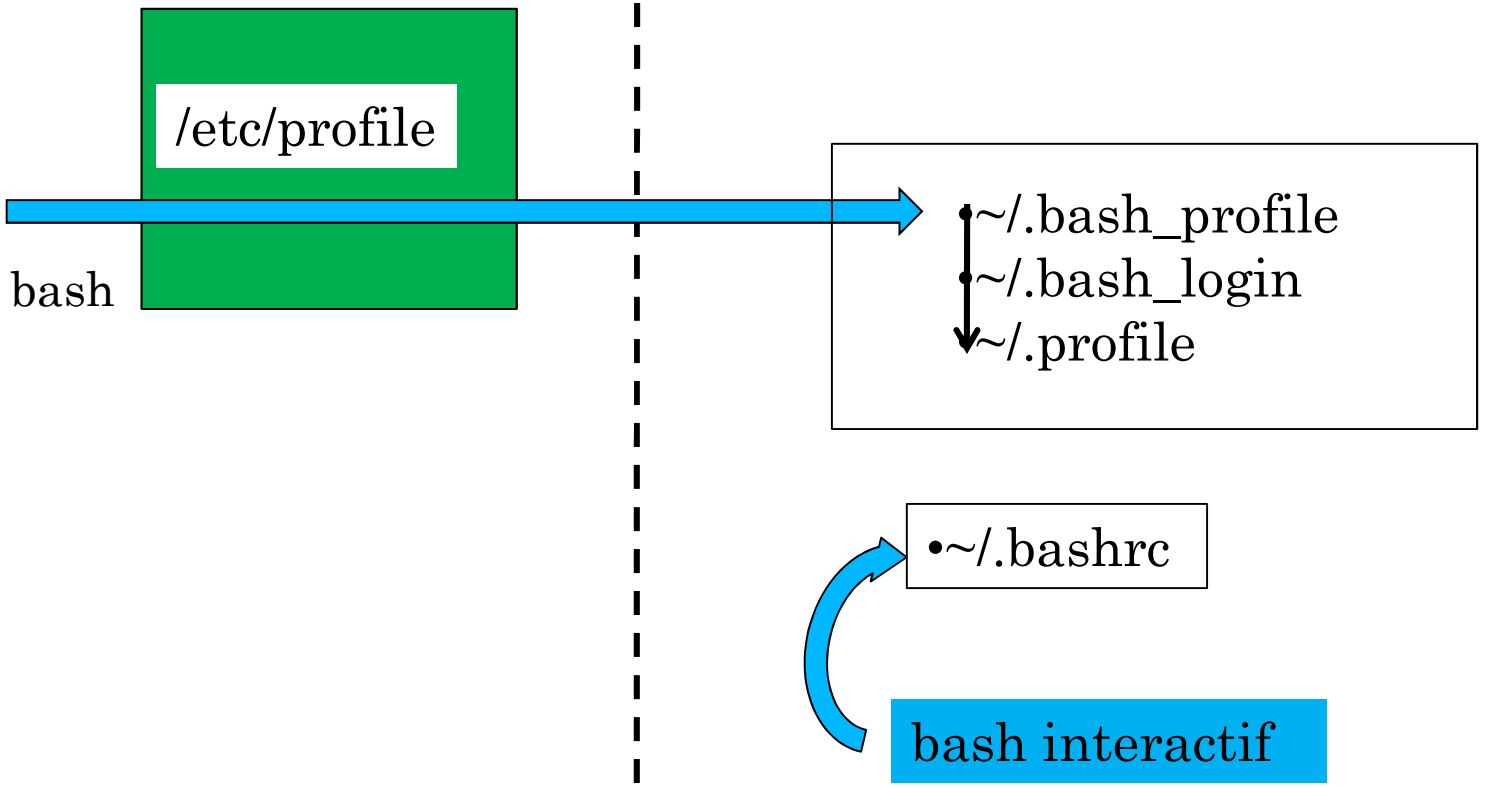
- Lors de l'authentification en mode commande le shell de login exécute dans l'ordre le script se trouvant dans les fichiers de configuration **/etc/profile** puis **~/.bash_profile** s'il existe sinon **~/.bash_login**. Si **~/.bash_login** n'existe pas le shell interprète le fichier **~/.profile**.

- En ouvrant un shell interactif, ce dernier exécute les fichiers de configuration suivant: **~/.bashrc**.

Fichiers système

Fichiers utilisateur

Connexion en mode commande



Exécution d'une commande non définit dans le chemin d'exécution

- Si une cmd se trouve dans un répertoire définit dans la variable PATH alors son exécution se fait simplement par l'invocation de son nom. Sinon l'utilisateur doit l'invoquer en utilisant son chemin d'accès.

Exemple:

$ /bin/ls ← permet l'exécution de la cmd ls en utilisant son chemin d'accès.

Exécution d'une séquence de commande

- Une séquence de cmd peut être exécuter en séparant les cmds par le caractère « ; ».

$ cmd1;cmd2;cmd3……

Exemple: la succession de cmds suivante

$ date

$ pwd

Peut être remplacer par :

$ date ; pwd ← séquence de deux cmds

Les séparateurs conditionnels de commandes

- Il est possible de contrôler la séquence d'exécution de commandes en utilisant des séparateurs conditionnels:

1. Le séparateur && permet d'exécuter la commande qui le suit si et seulement si la commande qui le précède a été exécutée sans erreur (code retour du processus nul).

2. Le séparateur || permet d'exécuter la commande qui le suit si et seulement si la commande qui le précède a été exécutée avec erreur (code retour du processus différent de 0.

Exemple:

$ cd projet1 || mkdir projet1 ← Si le répertoire *projet1 n'existe pas, alors il sera créé par la commande mkdir.*

$ cd projet1 && touch fich1 ← création d'un fichier nommé fich1 si la commande cd projet1 a été correctement exécutée.

Commandes de l'historique et édition

- L'interface d'édition de la ligne des commandes du shell possède deux fonctionnalités très puissantes:

1. L'historique des commandes: Lorsque le shell est utilisé, un mécanisme mémorise les dernières cmds lancé par l'utilisateurs à partir d'un terminal.
2. L'édition des commandes en ligne: Ceci est possible pendant la saisie de la cmd ou après une recherche d'une ligne de cmd dans l'historique.

Mécanisme d'historique

- Le nombre de lignes gérées par le mécanisme d'historique est défini par la variable HISTSIZE (1000 par défaut). Il est ainsi possible de rappeler ces commandes, de les modifier et de les rejouer.

La cmd **history** permet l'utilisation de ce mécanisme.

$ history 8 ← invoque la liste des 8 dernières cmds de l'historique.

- Lorsque l'on quitte la session, l'historique est sauvegardé dans un fichier dont le nom est défini dans la variable HISTFILE

- L'intérêt de ce mécanisme d'historique est de permettre de rechercher puis modifier une ancienne ligne de commandes. Il existe deux modes de recherche : le mode incrémental et le mode non-incrémental.

1. Dans le mode incrémental, la ligne de commandes recherchée est affichée au fur et à mesure de la saisie du ou des caractères de recherche:

$ <Ctrl-R> *motif* ← recherche incrémentale en arrière, à partir de la ligne courante de l'historique, de la dernière commande contenant le motif. <Ctrl-R> continuera la recherche.

2. Dans le mode non-incrémental, la ligne de commandes recherchée est affichée après la saisie de la chaîne de caractères de recherche:

$ <meta-p>motif<return> ← recherche non-incrémentale en arrière, à partir de la ligne courante de l'historique, la dernière commande contenant le motif. La ligne de commandes de l'historique contenant motif sera affichée après le <return>.

<meta> key : <Alt> ou <Esc>

Mécanisme de relance d'une commande

!! Relance la dernière cmd exécutée.

!10 Relance la cmd numéro 10

!-10 Relance la nième cmd avant la dernière cmd, sachant que la dernière et celle que l'on saisit.

!ls Relance la dernière commande commençant par *ls*

!?motif? Renouvelle la dernière commande qui contient la chaîne de caractères motif

Edition de la ligne de commande

- L'édition d'une ligne de commandes est possible pendant la saisie de la commande ou après une recherche d'une ligne de commandes dans l'historique. En voici quelques astuces qui permettent une édition rapide de ligne de cmd

1. Déplacement du curseur: En plus des flèches droite et gauche qui permettent de déplacer le curseur d'une position dans votre ligne de commandes, il existe des commandes qui permettent un déplacement plus rapide:

<ctrl-a> déplacement du curseur au début de la ligne de commandes.

<ctrl-e> déplacement du curseur à la fin de la ligne de commandes.

<meta-f> déplacement du curseur d'un mot vers la droite.

<meta-b> déplacement du curseur d'un mot vers la gauche.

2. Fonction couper-coller :

Les touches permettant de supprimer une partie de votre ligne de commandes sont :

<ctrl-k> suppression du texte de la position du curseur jusqu'à la fin de ligne.

<ctrl-u> suppression du texte de la position du curseur jusqu'au début de la ligne.

<meta-d> suppression du texte de la position du curseur jusqu'à la fin du mot.