Sébastien Derriere

M2 astrophysique

# Mise à niveau informatique - Linux - Unix



**1.4**

**Septembre 2017**

UNIVERSITÉ DE STRASBOURG

# Table des matières

# Objectifs

Ce cours passe en revue les principales commandes pour une utilisation optimale d'un système Linux/Unix via un terminal :

- utilisation du shell
- systèmes de fichiers
- droits d'accès
- manipulation de fichiers
- gestion de processus
- réseau
- scripts

# Introduction

Avec des interfaces graphiques toujours plus sophistiquées, tactiles, intuitives, l'utilisation d'un simple terminal pour taper d'obscures lignes de commandes pourrait passer pour une pratique d'un autre âge. Nous allons voir dans ce cours que l'utilisation d'un terminal et la connaissance d'un ensemble de commandes simples peut se révéler à l'usage extrêmement pratique, efficace, faire gagner énormément en productivité, et dans bien des cas remplacer avantageusement un grand nombre d'applications graphiques beaucoup plus lourdes et gourmandes en ressources.

# Généralités



## A. Systèmes d'exploitation Unix, Linux

Le système UNIX remonte à 1969. Il est fortement lié au langage de programmation C, puisque le langage C a été développé en 1971 afin d'écrire UNIX de façon portable. Réécrit en C en 1973, UNIX a depuis divergé en plusieurs familles.

Linux est une implémentation libre du système UNIX respectant les spécifications POSIX. Développé à l'origine par Linus Torvald, on trouve aujourd'hui de nombreuses distributions : Ubuntu, Debian, Red Hat, Mandriva...

*Attention*

Chaque utilisateur d'un système Unix ou Linux est identifié par un *login*, et se connecte à l'aide d'un mot de passe (*password*) qui est strictement personnel. Attention à la robustesse et à la confidentialité de votre mot de passe : l'utilisation d'un système informatique engage votre responsabilité.

Attention également à bien vous déconnecter (`logout`, `exit`, `Ctrl+D`) à la fin de votre session.

## B. Le shell

Le *shell* (coquille en anglais) désigne l'interface avec un système d'exploitation (*operating system* -- OS). Les OS basés sur Unix proposent deux types d'interfaces :

- interface graphique (*Graphical User Interface* -- GUI)
- interface en ligne de commande (*Command Line Interface* -- CLI)

Sous Linux, on dispose de 6 terminaux texte accessibles en appuyant sur les touches **`Alt+Ctrl+F1`** à **`F6`**. Pour revenir à l'interface graphique, il suffit de taper **`Alt+Ctrl+F7`**. On peut aussi ouvrir un terminal (ou console) pour accéder à la CLI depuis l'interface graphique (programmes **`konsole`** sous KDE ou **`xterm`**...).

# 1. Ligne de commande

A l'ouverture d'un terminal, l'utilisateur va voir une invite de commande :

### *Exemple*

```
martin@astromaster:~$
```

L'invite de commande (ou prompt) contient un certain nombre d'informations :

- **`martin`** : nom de l'utilisateur (login)
- **`@astromaster`** : nom de la machine
- **`~`** : emplacement courant dans l'arborescence de fichiers
- **`$`** : type d'utilisateur (ce symbole devient **`#`** pour un super-utilisateur)

Dans un terminal, l'utilisateur saisit des commandes sous forme de lignes de textes qui sont exécutées par l'interpréteur de commandes. Les plus répandus sont :

- Le Bourne Shell, **`sh`**, **`/usr/bin/sh`**. Toujours disponible par défaut.
- Bourne Again Shell, **`bash`**, **`/usr/bin/bash`**, généralement proposé par défaut sous Linux.
- Il y a aussi csh, tcsh, ksh...

Nous utiliserons bash pour ce cours d'introduction.
Les commandes saisies commencent par le nom de la commande, éventuellement suivi d'options, puis d'autres arguments.

### *Exemple*

**martin@astromaster:~$ commande [-options] [arguments]**

On a l'habitude de noter entre crochets les parties optionnelles. Dans la suite, seul le **`$`** du prompt sera conservé dans les exemples.

### *Attention*

Des espaces sont utilisés pour séparer les différentes options et arguments, c'est pourquoi il vaut mieux éviter d'utiliser des espaces dans les noms de fichiers ou de répertoires... tout comme les lettres accentuées ou des caractères spéciaux tels que '/"$() d'ailleurs !

### *Conseil*

Pour faciliter la saisie des commandes, il existe de nombreux raccourcis et astuces :

- La touche **`Tab`** permet l'auto-complétion des noms de commandes, fichiers, etc...
- Les flèches haut et bas permettent de défiler les commandes précédentes.
- **`Ctrl+a`** et **`Ctrl+e`** permettent de positionner le curseur en début et fin de ligne, respectivement.
- **`Ctrl+u`** et **`Ctrl+k`** permettent d'effacer le début ou fin de ligne à partir de la position actuelle du curseur.
- **`Ctrl+r`** permet de rechercher dans l'historique

## 2. Obtenir de l'aide

Linux dispose d'une aide en ligne accessible par la commande **`man`**, permettant d'obtenir les détails sur les commandes, fichiers... Le manuel est généralement divisé en 9 sections :

1. Commandes utilisateur
2. Appels système
3. Fonctions de bibliothèque
4. Fichiers spéciaux
5. Formats de fichier
6. Jeux
7. Divers
8. Administration système
9. Interface du noyau

*Exemple*

**$ man [section] xyz**

Permet d'obtenir la page d'aide sur la commande **`xyz`**

La commande **`which`** permet de savoir où se trouve le fichier exécutable correspondant à une commande donnée. Pour cela, les répertoires du PATH sont explorés (le PATH est une liste de répertoires séparés par **`:`** dans lesquels doit se trouver toute commande utilisée sans indiquer le chemin absolu de la commande).

*Exemple*

**$ which ls**

Devrait renvoyer **`/bin/ls`**

La plupart des commandes sous Linux acceptent une option **`--help`** qui affiche les options et la syntaxe à utiliser.

*Exemple*

**$ ls --help**

# Système de fichiers




## A. Parcours de l'arborescence

Tous les répertoires et fichiers forment une arborescence à partir d'une racine représentée par le symbole `/`.

Un élément peut être accédé par un chemin absolu (à partir de la racine, commençant par `/` donc) ou relatif (à partir de l'emplacement courant).

Il existe différents répertoires utilisés par le système (`/dev`, `/sys`, `/usr`, `/var`, `/tmp`...). Les répertoires des utilisateurs sont en général dans `/home`.

### *pwd*

La commande `pwd` (*Print Working Directory*) affiche le répertoire courant.

### *ls*

`ls` (*LiSt*) utilisé sans option affiche le contenu du répertoire courant.

### *cd*

`cd` (*Change Directory*) permet de changer le répertoire courant.

### *Substitution et wildcards*

Certains caractères spéciaux jouent un rôle particulier dans la manipulation de fichiers :

- `~` représente le *home directory*, base de l'arborescence par défaut d'un utilisateur. Pour un utilisateur martin, c'est souvent équivalent à `/home/martin`
- Le `/` utilisé en début de nom sert à indiquer la racine, mais ce même symbole est aussi utilisé pour indiquer la hiérarchie des répertoires et sous-répertoires.
- `.` représente le répertoire courant. C'est utile si on veut indiquer explicitement qu'on travaille sur des fichiers à l'intérieur du répertoire courant : `./monfichier.dat`

- `..` représente le répertoire un niveau au dessus du répertoire courant
- `~user` représente le *home directory* de l'utilisateur `user`
- `*` remplace n'importe quelle chaîne de caractères. Ainsi `*.dat` correspond à tous les fichiers dont le nom finit par `.dat`
- `?` remplace un caractère unique
- `[abtu]` remplace n'importe quel caractère unique parmi ceux entre crochets. Ainsi `t[ai]ti` correspond à `tati` ou `titi` mais pas `tahiti` ni `toti`.
- `[a-f0-9]` : l'usage d'un tiret permet de définir des intervalles, mais les crochets correspondent toujours à un caractère unique.
- `[!0-9]` le ! est une négation : un caractère unique qui n'est pas dans la liste ou l'intervalle suivant
- `{str1,str2,...}` n'importe laquelle des chaînes entre crochets
- `-` est le répertoire précédent (dans l'historique)
- Le symbole `\` placé devant un caractère spécial permet de l'utiliser comme caractère normal : `\*.dat` correspondrait à un fichier se nommant réellement `*.dat`, et non pas à tous les fichiers finissant par `.dat`

# B. Manipulation

## *mkdir*

`mkdir` (*MaKe DIRectory*) sert à créer un répertoire.

### *Exemple*

**`mkdir foo`** ou **`mkdir ./foo`**

Crée un répertoire `foo` dans le répertoire courant.

**`mkdir ../foo/bar`**

Crée un sous-répertoire `bar` dans le répertoire `foo` (qui se trouve au même niveau que le répertoire courant).

## *rmdir*

`rmdir` (*ReMove DIRectory*) supprime un répertoire (qui doit être vide).

## *touch*

`touch` permet de créer un fichier (vide), ou de changer la date de dernière modification d'un fichier existant.

## *rm*

`rm` (*ReMove*) permet de supprimer un ou plusieurs fichiers (si vous avez les droits pour le faire !).

## *mv*

`mv` (*MoVe*) permet de renommer un fichier, ou de déplacer un ou des fichiers dans l'arborescence.

### *Exemple*

**mv foo.dat bar.dat**

Renomme le fichier `foo.dat` en `bar.dat`

**mv [a-f]*.dat bar/**

Déplace le ou les fichiers satisfaisant à l'expression depuis le répertoire courant vers le répertoire `bar`.

### *cp*

**`cp`** (*CoPy*) permet de dupliquer des fichiers ou répertoires, éventuellement récursivement. L'information est effectivement dupliquée, donc cela prend de la place dans le système de fichiers !

### *Exemple*

**cp fich1.tex fich2.tex**

**cp -rp foo ./bar/**

### *ln*

**`ln`** (*LiNk*) permet de créer des liens entre fichiers. Avec l'option **`-s`**, on crée des liens symboliques : c'est un raccourci vers un fichier, les données ne sont pas copiées ; si on supprime le lien, le fichier original n'est pas détruit.

### *Exemple*

**ln -s /var/local/Aladin.jar ~/**

## C. Recherche

### *Espace occupé*

Il existe des commandes permettant de surveiller l'espace occupé sur le disque.

**`df`** affiche l'état de remplissage des différentes partitions (utile pour vérifier si il y a un problème de place disponible).

**`du`** permet d'afficher la taille d'un ensemble de fichiers ou de répertoires.

### *locate*

**`locate`** affiche tous les emplacements de fichiers satisfaisant à une expression.

### *Exemple*

**locate *.MP3**

Affiche tous les fichiers dont le nom finit par `.MP3` (en majuscules !). Pour ignorer la casse, il y a l'option **`-i`**

### *find*

**`find`** permet aussi de faire des recherches de fichiers, de façon ciblée (recherche par nom mais aussi par date d'accès ou de modification, par taille...),

et d'effectuer des actions directement sur les fichiers trouvés.

*Exemple : Supprimer récursivement tous les fichiers ou répertoires vides dans le home directory*

**find ~ -empty -delete**

# Permissions et droits



## A. Utilisateurs

### *who*

**who** permet de voir qui est connecté sur le système et depuis quel endroit.

**who am i** ou **whoami** affiche l'utilisateur connecté dans la session courante.

Chaque utilisateur est déclaré dans le fichier **/etc/passwd**. Il peut changer son mot de passe à l'aide de la commande **passwd** (ou **yppasswd** pour les systèmes utilisant le *Network Information System* - NIS).

Un utilisateur peut appartenir à plusieurs **groupes**. Les appartenances sont stockées dans le fichier **/etc/group**.

### *Administration*

Il existe un utilisateur spécial disposant de tous les droits sur le système, nommé super-utilisateur ou **root**. On peut passer root à l'aide de la commande **su** (à condition de connaître le mot de passe !).

Un utilisateur peut parfois se voir autoriser l'usage de certaines commandes réservées au super-utilisateur, à condition de précéder ces commandes par **sudo**. Par exemple pour monter un nouveau filesystem : **sudo mount /media/hd_mobile**

## B. Inodes et blocs

Chaque fichier ou répertoire d'un système de fichiers Linux est représenté par :

- un **inode** (*Index NODE*) unique : une structure qui contient toutes les informations relatives à ce fichier, excepté son nom
- des **blocs de données** qui contiennent les données du fichier

Le système de fichiers Linux contient donc d'une part la table des inodes

(identifiés par des numéros) et les blocs de données. Lorsqu'un fichier a plusieurs liens (plusieurs noms), ils pointent sur le même inode.

L'inode d'un fichier ou répertoire contient :

- type de fichier : ordinaire (**-**), répertoire (**d**), lien symbolique (**l**), bloc (**b**), caractère (**c**), pipe (**p**) ou socket (**s**)
- droits des utilisateurs sur le fichier
- nombre de liens
- UID (*User ID*) identifiant de l'utilisateur propriétaire du fichier
- GID (*Group ID*) identifiant du groupe du propriétaire du fichier
- taille du fichier en octets
- atime : date de dernière lecture du fichier
- mtime : date de dernière modification du fichier
- ctime : date de dernier changement de l'inode du fichier

Les données d'un répertoire contiennent une table de correspondance entre les noms des fichiers et répertoires contenus et leurs inodes.

# C. Droits des fichiers

Linux et Unix sont des systèmes multi-utilisateurs. Plusieurs personnes peuvent utiliser la même machine simultanément, et partager des fichiers. Une personne est membre d'au moins un groupe d'utilisateurs. Les droits d'accès aux fichiers et répertoires (lecture, écriture, exécution) peuvent être gérés de façon très détaillée.

### *Définition*

Un utilisateur est identifié par :

1. Un numéro unique : l'UID
2. Un nom d'identificateur unique : le *login*
3. Un mot de passe : *password*

Un groupe est identifié par :

1. Un numéro unique : le GID
2. Un nom unique

## *Utilisateurs et droits*

Lorsqu'un fichier ou répertoire est créé, il se voit associés les UID et GID de l'utilisateur. Les droits peuvent ensuite être définis pour :

- **u** (*User*) : le propriétaire du fichier
- **g** (*Group*) : le groupe
- **o** (*Other*) : tout utilisateur autre que le propriétaire ou les membres du groupe
- **a** (*All*) : tout le monde

Pour chacun des niveaux (`ugo`, on peut ouvrir ou non les droits standards suivants :

| Droit | Fichier | Répertoire |
|---|---|---|
| **r** : lecture | Lire le contenu du fichier | Lire la totalité des entrées. Sans ce droit, on peut tout de même lire un fichier dont on connaît le nom |
| **w** : écriture | Modifier le contenu du fichier | Créer ou supprimer des entrées dans le répertoire |
| **x** : exécution | Exécuter le fichier | Accéder aux entrées du répertoire (cd, pwd, ...) |

*Tableau 1 Droits standards*

Il existe des droits spéciaux, SUID, SGID, Sticky Bit, qui sortent du cadre de cette introduction.

La première colonne renvoyée par la commande `ls -l` correspond aux droits des fichiers.

Un fichier normal ouvert en lecture et écriture pour l'utilisateur et le groupe, mais en lecture seule pour les autres aura des droits représentés par `-rw-rw-r--`.

Un répertoire aura souvent les droits `drwxrwxr-x`.

## Changement des droits

**chmod** est la commande permettant de changer (opérateur ajouter **+**, supprimer **-**, fixer **=**) les droits (**r**, **w**, **x**) à différents utilisateurs (**u**, **g**, **o**, **a**).

### Exemple

**$ chmod utilisateur[opérateur]droits fichier**

**$ chmod u+x fichier** rend exécutable fichier pour l'utilisateur

**$ chmod a+r fichier** rend le fichier lisible par tous

Les droits peuvent être exprimés avec une notation numérique, rwx représentant 3 bits, ce qui donne sous forme octale une valeur entre r+w+x=0 et 7.

- r=0 ou 4
- w=0 ou 2
- x=0 ou 1
- Et toutes les combinaisons possibles ! 6 signifie rw mais pas x.

### Exemple

**$ chmod 744 fichier**

rend le fichier ouvert en rwx (7) par l'utilisateur, et seulement en lecture (4) par le groupe ou les autres.

## Changement de propriétaire

**chown** permet de changer le propriétaire ou le groupe d'un fichier.

**$ chown <nouveau proprio> fichier**

```
$ chown :<nouveau groupe> fichier
```

**$ chown <nouveau proprio>:<nouveau groupe> fichier**

Il y a aussi la commande **chgrp** pour changer seulement le groupe.

# Manipulation de fichiers




## A. Affichage

### *wc*

**`wc`** (*Word Count*) **`[-lcw]`** affiche le nombre de lignes, mots, caractères d'un fichier.

### *cat*

**`cat`** affiche un fichier texte à l'écran. Parmi les options utiles, **`-T`** affiche les tabulations, **`-v`** les caractères de contrôle.

**`cat`** permet aussi de concaténer plusieurs fichiers : le résultat est aussi affiché à l'écran.

**$ cat hipparcos.dat**

**$ cat hipparcos.dat tycho.dat**

### *less, more*

**`less`** est une version améliorée de **`more`**, et permet un affichage contrôlé du contenu d'un fichier, avec de nombreuses commandes :

- <Return> : avance d'une ligne
- <Espace> : avance d'une page
- q ou Q : quitte
- *« k »*f (Forward) : avance de *« k »* pages
- *« k »*b (Backward) : recule de *« k »* pages
- /<RegExp> : recherche l'expression régulière <RegExp>
- n (*next*) : prochaine occurrence de l'expression régulière recherchée
- ...

### *Fusions*

**`paste fich1 fich2`** fait la fusion de fichiers ligne à ligne.

**`join`** effectue une jointure de fichiers au sens relationnel.

### Comparaison

**diff** compare deux fichiers et affiche les différences.

**vimdiff** fait la même chose, mais le résultat est affiché dans l'éditeur **vim**.

**comm** fait la comparaison de fichiers préalablement triés.

## B. Sélection

### head, tail

**head** affiche les premières lignes d'un fichier (10 par défaut).

**tail** affiche les dernières lignes d'un fichier.

### grep

**grep** (*Global Regular Expression Print*) permet de rechercher dans un fichier des lignes satisfaisant à une expression régulière. Chaque ligne du fichier contenant l'expression recherchée est affichée en sortie.

**$ grep [options] "RegExp" [fichier]**

Options utiles : **-i** ignore les différences majuscules/minuscules ; **-n** affiche le numéro de ligne ; **-v** affiche les lignes ne contenant pas l'expression régulière.

### Complément : Les expressions régulières

Utilisées par différents utilitaires (grep, sed, awk, less, more, vi...), les expressions régulières permettent de définir des modèles de chaînes de caractères destinés à faire des recherches de chaînes complexes. A ne pas confondre avec les wildcards des fichiers auxquels elles peuvent ressembler, elles utilisent un langage particulier dont voici quelques éléments :

- **^** représente le début de ligne
- **$** représente une fin de ligne
- **.** représente n'importe quel caractère
- **[xyz]** un exemplaire de n'importe lequel des caractères entre crochets. Pour utiliser ], le mettre en début de liste. Pour - ou [, en fin.
- **[a-z]** un caractère unique de l'intervalle
- **[^a-z]** n'importe quel caractère en dehors de l'intervalle
- **a*** zéro ou plusieurs répétitions du caractère a
- **(**RegExp**)** définit une sous-chaîne accessible par la suite par **\i**
- **\i** réutilisation de la sous-chaîne numéro **i**
- ****car ôte le rôle particulier d'un caractère réservé (comme *.[] )

### Opérations sur les colonnes

**cut** extrait des colonnes d'un fichier. Les colonnes peuvent être définies par un nombre de caractères (**-c**) ou correspondre à des champs (**-f**) séparés par un séparateur (défini par **-d**).

**colrm** supprime des colonnes dans un fichier.

### *Tri*

**`sort`** permet de faire des tris de fichiers : sur des lignes complètes, ou pour des champs spécifiques, par ordre alphabétique ou numérique...

**`uniq`** gère les répétitions dans un fichier trié. **`-c`** compte les répétitions, **`-u`** garde un exemplaire unique...

### *Divers*

Il existe beaucoup d'outils de modification, parmi lesquels on peut noter :

- **`tr`** (utilisable uniquement avec des redirections ou pipes) transforme des caractères du fichier (suppression ou remplacement).
- **`sed`** est un éditeur rudimentaire non interactif mais assez puissant.
- **`awk`** (ou **`gawk`**) permet de faire de la manipulation programmable de fichiers en mode ligne par ligne. C'est presque un langage de programmation à apprendre pour la manipulation de fichiers, mais cela sort du cadre de ce cours !
- **`vi`** (ou **`vim`** pour une version améliorée, **`gvim`** pour une version fenêtrée) est un éditeur de texte puissant utilisable dans une console. Là encore, son utilisation nécessiterait un manuel complet !
- **`fold -n`** découpe (replie) les lignes d'un fichier en lignes de longueur n caractères.

## C. Compression, archives

### *Archives tar*

Il est possible de stocker un ensemble de fichiers ou toute une portion d'arborescence dans un seul fichier archive sous forme d'un fichier tar. L'extraction du contenu d'un fichier tar permettra de récupérer des fichiers individuels ou de reconstruire l'arborescence.

**$ tar -[options] archive.tar [fichiers]**

- **`$ tar -cf archive.tar foo bar`** : crée un archive contenant les fichiers foo et bar
- **`$ tar -tvf archive.tar`** : affiche le contenu de l'archive.
- **`$ tar -xf archive.tar`** : extrait le contenu de l'archive.

### *Fichiers compressés*

Il existe différentes commandes pour (dé)compresser des fichiers en ligne de commande :

- **`zip`** (**`unzip`**), habituellement extensions en .zip
- **`gzip`** (**`gunzip`**), habituellement extensions en .gz. **`zcat`** décompresse un fichier vers la sortie standard.
- **`bzip2`** (**`bunzip2`**) pour les fichiers .bz2

# Processus, flux et variables




## A. Processus

On peut écrire plusieurs commandes sur une même ligne du shell, en les séparant par des `;`

Une commande qui se termine sans erreur renvoie le code 0 (zéro).

On peut exécuter plusieurs commandes de manière conditionnelle à l'aide des opérateurs `||` ou `&&` :

- `cmd1 || cmd2` : cmd2 est exécutée seulement si cmd1 se termine en erreur (statut de sortie non nul)
- `cmd1 && cmd2` : cmd2 est exécutée seulement si cmd1 se termine avec un statut de sortie égal à zéro

Tout programme qui s'exécute est un process géré par le système, et identifié par un numéro, son PID (*Process IDentifier*).

`ps` est une commande qui affiche des informations sur les process en cours d'exécution sur la machine.

`$ ps -ux` affiche les process de l'utilisateur.

`$ ps -ef` affiche tous les process sur la machine

`$ ps -ejH` indente les process sous forme d'un arbre

La commande `top` permet de surveiller l'utilisation des ressources par les process, et de les trier selon divers critères. (touche q pour quitter)

Un process peut mettre du temps à s'exécuter (ou bien nécessiter une action explicite de l'utilisateur dans une autre fenêtre pour se terminer) et dans ce cas l'utilisateur n'a plus de prompt dans l'invite de commande. Un tel processus interactif courant peut être :

- Interrompu par `Ctrl+c`. Cette interruption peut être inhibée par le process.
- Stoppé par `Ctrl+z`. On peut reprendre le process par la suite.
- Tué par la commande `kill -9 PID` (cette interruption est prioritaire et

ne peut pas être inhibée par le process)

Un process peut être lancé explicitement en tâche de fond (background) en terminant la ligne de commande par le symbole **&**.

Les process lancés en background ou stoppés se voient attribués des numéros. La commande **jobs** permet de voir ces process. Pour le job numéro n, on peut effectuer les actions suivantes :

- **bg %n** : continue l'exécution du job en background
- **fg %n** ou **%n** : reprend l'exécution du job en interactif (foreground)
- **kill -9 %n** : tue le job spécifié
- **stop %n** : stoppe le job (équivalent à **Ctrl+z**)

On peut aussi tuer des process à l'aide des commandes **pkill** (à l'aide du nom du process et non de son PID), ainsi qu'avec **xkill** (pour tuer une fenêtre graphique).

**history** est une commande affichant les dernières commandes exécutées dans le shell courant.

On peut utiliser l'historique pour répéter et modifier des commandes :

- **!!** : désigne la commande précédente
- **!n** : désigne la commande numéro n
- **!-n** : désigne la n-ième commande précédente
- **!str** : désigne la commande la plus récente commençant par "str"
- **!?str** : désigne la commande la plus récente contenant str
- **!*** : l'ensemble des arguments de la dernière commande
- **!$** : le dernier argument de la dernière commande
- **^str1^str2** : ré-exécute la dernière commande en substituant str1 par str2

Les raccourcis commençant par ! peuvent être complétés par des modificateurs :

- **:p** ré-affiche la commande sans l'exécuter (ainsi **!!:p** affichera le texte de la dernière commande)
- Pour les mots considérés comme des noms de fichiers (ex `/d1/d2/fich.ext`)
  - **:h** ne garde que le nom du répertoire (`/d1/d2`)
  - **:t** ne garde que le nom du fichier, sans le chemin (`fich.ext`)
  - **:r** ne garde que la racine du nom, sans l'extension (`/d1/d2/fich`)
  - **:e** ne garde que l'extension (`.ext`)

Tout process s'exécute avec une certaine priorité. La priorité d'une commande peut être réduite lors de son lancement en la précédant de la commande **nice**. On peut aussi abaisser la priorité d'un processus déjà lancé avec la commande **renice**.

# B. Flux et pipes

## *Fichiers standards*

Tout programme UNIX ou Linux a trois fichiers standards :

- le fichier standard input (**`stdin`**) : pour l'entrée de données. Par défaut c'est une entrée au clavier.
- le fichier standard output (**`stdout`**) : pour la sortie de résultats. Par défaut, sortie sur l'écran.
- le fichier standard erreur (**`stderr`**) : pour la sortie des messages d'erreur. Par défaut, sortie sur l'écran.

Ces trois fichiers standards peuvent être **redirigés** depuis ou vers un autre fichier :

- **`cmd < fich`** : le fichier `fich` sera pris comme entrée standard de la commande
- **`cmd > fich`** : la sortie standard de la commande est redirigée vers le fichier fich
- **`cmd >> fich`** : le fichier standard output est concaténé à la fin de fich
- **`cmd < fich1 > fich2`** : redirige à la fois stdin depuis fich1 et stdout vers fich2 (c'est l'usage de la commande **`tr`**)
- **`cmd >& fich`** : redirige stdout et stderr dans fich (attention pas d'espace entre > et & sinon & est pris comme fin de commande)

## *Pipes*

Deux process peuvent communiquer par l'intermédiaire de leurs fichiers standards d'entrée et de sortie au moyen de *pipes* (tuyaux en anglais). L'opérateur du *pipe* est **`|`**

- La sortie standard de la première commande est envoyée en entrée standard de la suivante : **`ls | wc -l`**
- On peut faire un pipe combinant stdout et stderr avec l'opérateur **`|&`**
- On peut combiner redirections et pipes : **`cmd1 < fich1 | cmd2 |& cmd3 > fich2`**
- La commande **`tee`**, utilisée avec des pipes, permet de rediriger son entrée standard à la fois vers un fichier et sur la sortie standard
  **... | tee trace.log | ...**

# C. Variables

## *Variables d'environnement*

Lors de l'exécution d'un shell, un certain nombre de variables d'environnement sont définies. On peut afficher leur nom et leur valeur par la commande **`env`**, **`set`**, ou **`printenv`**.

La commande **`echo`** permet d'afficher du texte simple, mais on peut l'utiliser avec des variables, en précédant leur nom du symbole **`$`** : elles seront substituées par leur valeur à l'affichage.

**$ echo "Je suis $USER et je travaille dans $HOME"**

Certaines variables d'environnement sont fort utiles. Par exemple, PATH indique quels répertoires sont parcourus pour rechercher les commandes à exécuter. SHELL indique quel est le shell courant. Etc.

On peut ajouter des variables d'environnement de deux façons dans le bash :

`VAR=valeur` (utiliser des guillements simples si c'est un texte avec des espaces `VAR='ma valeur'`, ou des guillemets doubles pour substituer des variables `VAR="$AUTRE_VAR"`), puis

**export VAR**

ou de façon plus compacte en une seule commande

**export VAR=valeur**

On peut ensuite afficher son contenu, et la variable sera accessible aux sous-process du shell : `echo $VAR`

On peut supprimer une variable par `unset VAR` ou bien encore `env -u VAR`

Attention sous d'autres shells comme csh, la commande à utiliser est `setenv`, avec une syntaxe différente !

## *Autres variables*

Il existe des variables spéciales qui pourront être utiles en particulier dans les scripts :

- `$0` : le nom de la commande en train d'être exécutée
- `$#` : le nombre d'arguments
- `$1` à `$9` : chacun des arguments de la ligne de commande
- `$*` : chaîne contenant tous les arguments
- `$$` : le numéro du process courant. Utile pour définir des noms de fichiers temporaires uniques `... > /tmp/cmd$$.log`
- `$?` : le statut de la dernière commande exécutée (0 si il n'y a pas eu d'erreur)

# Réseau




## A. Connexions vers d'autres machines

### *Connexions non sécurisées*

On peut se connecter à distance sur des machines, soit pour y ouvrir une session, soit pour faire des transferts de fichiers. On préfère en général se connecter par des protocoles sécurisés, mais il existe d'autres moyens d'accès comme :

- **`rlogin`** (*Remote LOGIN*) pour ouvrir rapidement un terminal à distance
- **`telnet`** : fonctionne entre des machines hétérogènes
- **`rcp`** : copie de fichiers entre machines distances
- **`ftp`** : transfert de fichiers

Pour pouvoir se connecter sur une machine distante, il faut que celle-ci accepte les connexions, c'est à dire que les ports de communication correspondants doivent être ouverts, et qu'un programme doit tourner sur le serveur distant pour répondre aux demandes de connexions. De plus en plus de machines n'autorisent que des connexions via des systèmes sécurisés.

### *Remarque : ftp anonyme*

Certains serveurs offrent un ftp anonyme, qui permet de déposer ou de récupérer des fichiers sans posséder de compte sur la machine. Dans ce cas, l'identifiant à utiliser est `anonymous` (ou `ftp`), et le mot de passe est votre adresse email complète.

### *Connexions sécurisées*

Le protocole le plus courant pour des connexions sécurisées est SSH, qui utilise un système d'encryption. Il existe plusieurs commandes associées.

### *Méthode : ssh*

**`ssh`** est un client SSH permettant un accès sécurisé à une machine distante. Lors d'une première connexion à une machine distante, un échange de clés a lieu entre les machines. Ces informations sont stockées dans un fichier,

généralement **~/.ssh/known_hosts**

On se retrouve sur la machine distante dans un shell, dans lequel on peut effectuer les commandes habituelles. A noter la commande **hostname** qui affiche le nom de la machine.

Si un changement de système changeant les clés a eu lieu sur l'une ou l'autre des machines, un avertissement sera affiché, et la connexion ne sera plus possible. Il faudra supprimer la ligne correspondante dans le fichier, puis recommencer l'opération.

### *sftp*

**sftp** est une version sécurisée de FTP (*File Transfer Protocol*). On se connecte avec son login et mot de passe (sauf pour un ftp anonyme). On dispose ensuite d'un jeu de commandes réduites, parmi lesquelles :

- **pwd** : affiche le répertoire courant de la machine distante
- **cd** : change de répertoire sur la machine distante
- **ls** : liste le contenu du répertoire distant
- **bin** : passe en mode binaire pour les transferts (le défaut auquel on peut revenir par la commande **ascii** est de faire les transferts en mode texte, ce qui endomage les fichiers binaires)
- **get** `fichier` : tranfert du fichier de la machine distante vers la machine locale
- **put** `fichier` : transfert de la machine locale vers la machine distante
- **delete** `fichier` : supprime un fichier de la machine distante
- **mget** `fich*` : transferts *m*ultiples, on peut utiliser les wildcards dans les noms de fichiers
- **mput** `fich*` : idem dans l'autre sens
- **prompt** : bascule pour activer/désactiver la confirmation lors des transferts multiples
- **bye** ou **quit** : fin de session FTP
- **lcd** : change le répertoire sur la machine locale

### *scp*

**scp** permet de faire des copies entre machines distantes en utilisant ssh. La syntaxe ressemble à celle de **cp**, mais les fichiers d'origine ou de destination peuvent être précédés du nom d'une machine hôte (et d'un login si les noms d'utilisateurs sont différents de l'utilisateur courant) :

**$ scp [-rp] [[user1@]host1:]fichier1 [[user2@]host2:]fichier2**

Il faudra bien entendu s'identifier avec un mot de passe sur la machine distante.

# B. Internet

### *Machines et adresses*

Toute machine connectée au réseau INTERNET peut être identifiée de deux façons : par son nom (composé de 2 à 5 champs : `eso.org`,

`astro.unistra.fr`, `astromaster.u-strasbg.fr`) ou par son adresse IP (composée de 4 champs numériques entre 0 et 255 dans le protocole IPv4 : `130.79.128.16`).

La correspondance entre noms et numéros IP est assurée par des serveurs de noms (DNS).

On peut tester si une machine est accessible à l'aide de la commande **`ping`** `{nom,IP}`.

D'autres utilitaires permettent de trouver les correspondances entre un nom de machine et l'adresse IP (**`dig`**, **`nslookup`**), ou de visualiser le trajet réseau vers une autre machine (**`traceroute`**).

Les noms des machines ne doivent pas être confondus avec les adresses électroniques des mails, qui sont attribuées à des utilisateurs.

Comme le nombre total d'adresses IP disponibles est limité à 4 294 967 296 en IPv4, un nouveau protocole IPv6 a été mis au point, offrant plus de 281 000 milliards d'adresses. La migration vers ce nouveau protocole se fait doucement...

## *Web et téléchargement*

Les navigateurs internet offrent un moyen convivial de naviguer dans des documents hyper-texte. Mais ils ne sont qu'une sur-couche sur des protocoles et des opérations de plus bas niveau. Et ils ne sont pas forcément adaptés au téléchargement de données qui peuvent être accessible via le web.

Divers outils permettent d'effectuer des requêtes http en contrôlant finement les échanges ou de sauvegarder le contenu de documents web sans avoir besoin de recourir à un navigateur.

**`wget`** permet de sauvegarder le contenu d'une page dans un fichier local.

**wget -O hipparcos.info "http://cdsarc.u-strasbg.fr/vizier/ftp/cats/I/239/ReadMe"**

Les guillemets autour de l'URL sont bien utiles pour la traiter comme une seule chaîne de caractères et éviter les symboles spéciaux.

**`curl`** est un utilitaire très puissant pour récupérer le contenu d'une URL, supportant de nombreux protocoles (HTTP, FTP, IMAP, POP, SCP, ...) avec beaucoup d'options.

Pour l'anecdote, il existe un navigateur internet en mode texte parfois installé sur les machines : **`lynx`**, pour explorer les pages web depuis un terminal.

# Scripts

VII



## A. Scripts bash

On peut écrire dans un fichier texte un ensemble de commandes pour les exécuter ensuite : on parle de script.

Un script nommé `script` peut être exécuté de deux manières :

- interprétation directe, en lançant **`$ bash script [params]`**
- interprétation indirecte : **`$ ./script [params]`**

  à condition que le script ait été rendu exécutable (permission **`x`**).

Les commandes du script sont interprétées par un shell. Un script est exécuté par défaut avec le Bourne Shell (sh). Il doit commencer par le symbole **`#`** pour être interprété avec le login shell de l'utilisateur (bash pour vous). En pratique, il est d'usage d'indiquer explicitement sur la première ligne quel shell on veut utiliser pour interpréter les commandes, par exemple dans le cas du bash on écrira en début de fichier :

```
#!/bin/bash
```

### *Variables*

Vous avez accès dans un script à toutes les variables d'environnement du shell.

Vous pouvez de plus définir des variables locales dans votre script, simplement en leur affectant une valeur :

```
var=valeur
```

Une variable peut ensuite être utilisée en faisant précéder son nom du symbole $. On mettra le nom de la variable entre {} pour l'isoler du reste d'une chaîne de caractères (la concaténation se faisant de manière implicite).

```
echo $var
echo "${var}s importantes"
```

De plus, il existe des variables prédéfinies :

- **`$0`** est le nom du script en cours d'exécution
- **`$$`** est le numéro du process du shell
- **`$n`** est l'argument numéro n passé au script

- `$@` est la liste des arguments

Il est aussi possible de manipuler des variables de type tableau.

## Lecture et affichage

bash possède deux commandes standard pour lire des données et les afficher.

**`read`** permet de saisir une entrée de l'utilisateur au clavier.

**`echo`** permet d'afficher sur la sortie standard. On peut aussi faire un affichage formaté avec **`printf`**.

## Structures conditionnelles

On peut faire des tests conditionnels dans un script bash :

**if [commande_test] ; then**

... commandes ici ...

**else**

... autres commandes ...

**fi**

Il existe un tas d'expressions primaires qui peuvent être utilisées dans le test, par exemple pour vérifier les propriétés d'un fichier, comparer des chaînes de caractères ou des expressions arithmétiques.

**`if [-f $FILENAME]`** renvoie vrai si le fichier existe

**`if [arg1 -eq arg2]`** vrai si égalité des deux entiers arg1 et arg2

...

## Boucles

Il existe des instructions pour effectuer des boucles :

**`for`** `var` **`in`** `liste` **`;`** **`do`** `commande` **`;`** **`done`**

On peut utiliser directement les wildcards pour définir une liste, ou bien une sous-requête.

**`for i in ex* ; do cp $i $i.bak ; done`** : copie tout les fichiers dont le nom satisfait ex* dans un nouveau fichier dont le nom est suffixé par .bak.

La commande **`seq`** permet d'afficher une séquence de nombres :

**`seq 0 5 100`** : affiche les nombres de 0 à 100 avec un incrément de 5.

## Autres structures

D'autres structures existent en bash : **`case`**, **`while`**, **`until`**, **`break`**, **`continue`**, **`shift`**... vous pouvez vous reporter à une documentation complète pour en apprendre davantage sur la syntaxe du bash.

## A savoir

Une commande placée entre backquotes (``) est exécutée dans un sous-shell. Le résultat de l'exécution de cette commande, sa sortie standard, est décomposée en mots, tous sur la même ligne (les sauts de ligne sont remplacés par des espaces).

Essayez **`rep_courant=`pwd``** puis **`echo $rep_courant`**

Le backslash (**`\`**) s'applique au caractère suivant en enlevant son caractère spécial. **`\\`** permet d'utiliser le caractère backslash !

Les apostrophes simples (`''`) sont utilisables pour délimiter une chaîne de caractères, mais la substitution de variables n'est pas faite.

Les guillemets doubles (`""`) permettent aussi de délimiter les chaînes de caractères, avec substitutions des noms de variables et de fichiers.

Comparez `echo "Je travaille dans $HOME"` et `echo 'Je travaille dans $HOME'`.

On peut insérer des commentaires dans un script en faisant commencer des lignes par `#`.

# B. Divers

## *Quelques dernières astuces pour la route*

`clear` efface le contenu du terminal courant.

## *Alias*

Lors du démarrage du shell, un fichier de configuration nommé `~/.bashrc` est exécuté. On peut éditer ce fichier pour y ajouter des commandes, comme la définition de variables, ou d'`alias`. Un alias est la définition d'une nouvelle commande à partir de commandes existantes.

ex : `alias ll='ls -alF'`

La commande `unalias` permet de supprimer un alias existant.

ex :`unalias ll`

## *Source*

On peut exécuter des commandes se trouvant dans un fichier en tapant

`source fichier`.

## *Calendrier*

`date` permet d'afficher la date courante dans de nombreux formats différents selon les options. `cal` affiche un calendrier. Ex : `cal 09 2013`

## *Imprimantes*

Des commandes permettent de gérer les imprimantes réseau.

`lpq` affiche l'état de la queue d'impression sur l'imprimante par défaut.

`lpq -Plaserdea` affiche la queue d'impression de l'imprimante laserdea.

`lprm -Plaserdea N` permet de supprimer le job d'impression numéro N de cette queue.

## *Crontab*

On peut définir des tâches répétitives programmées à certaines heures, tous les jours, certains jours de la semaine, etc. Pour cela, il faut éditer un fichier cron, à l'aide de la commande `crontab -e`.

## *Locale*

Les applications Linux peuvent utiliser des variables d'environnement pour

s'adapter aux paramètres régionaux (*locale* en anglais) ou préférences linguistiques des utilisateurs. Cela peut poser des problèmes avec certains programmes, par exemple des nombres décimaux non reconnus car écrits 1.234 au lieu de 1,234 à la française.

On peut voir les préférences en cours avec la commande **`locale`**.

On peut changer temporairement les préférences en modifiant la variable `LC_ALL`, par exemple :

```
export LC_ALL=POSIX
```