CESI
ALUMNI

INFAL51 – Linux
Commandes de bases

15/16/17 décembre 2025

Claude Mourzelas

Introduction à Linux

I. Introduction

Un peu d'histoire / Positionnement sur le marché

L'environnement de travail

L'interpréteur de commandes

Aide et documentation

Introduction à Linux

II. Principes et commandes de base

Système de fichiers

Commandes de base

Liens symboliques et physiques

Droits d'accès

Introduction à Linux

III. Redirections et processus

Entrées/sorties et redirections * Les pipes

Contrôle de tâche et de processus

Introduction à Linux

IV. Introduction au shell

Variables

Substitution de commandes

Alias

La commande for

La commande seq

Introduction à Linux

V. Programmation shell

Script shell

Variables, paramètres et substitutions

Structures de contrôle Fonctions

Commandes utiles

Concepts de base

Qu'est ce que l 'informatique ?

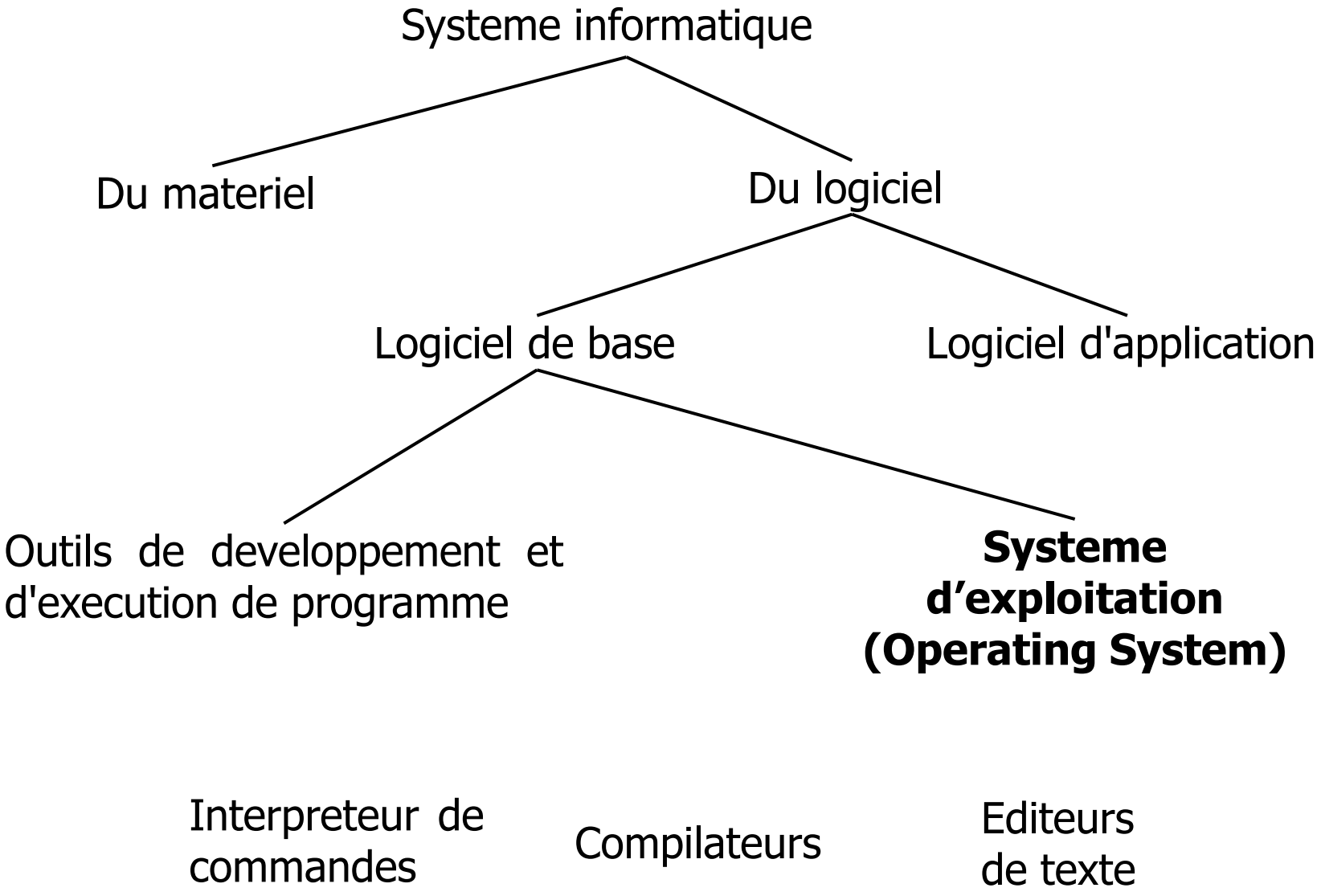
- L'informatique est le traitement automatique de I 'information
- L'information manipule pour :
  - L'université : les étudiants, leurs notes, date de leurs inscriptions, leurs emplois du temps ...
  - une entreprise : les produits, temps de production, prix, les stocks ...
- Le traitement automatique implique un traitement qui suit des régies qui peuvent être identifiées et éventuellement programmées dans un ordinateur

Qu'est ce qu'un système informatique ?

- Un système informatique est un ensemble cohérent de matériels et de logiciels destines a assurer le traitement automatique d'informations

- Un tel système est compose de 4 entités :
  - le matériel (mémoire, processeur, disque, clavier, etc.)
  - le système d'exploitation (Windows, Unix, Linux,...)
  - les programmes duplications (Programmes, Jeux, Réservation d'avion,...)
  - les utilisateurs

Qu’est ce qu'un système informatique ?



Outils de developpement et d'execution de programme

- **L'interpreteur de commandes** (shell): permet d'acceder aux fonctions du systeme a I'aide d'un *langage de commande*

- **Les compilateurs** : sont chargés de traduire des programmes ecrits dans des langages de haut niveau en une suite destructions en *langage machine*

- **Les editeurs de textes** : permettent de saisir et modifier du texte (par exemple des programmes)

Important :

- Ces outils ne font pas partie du système d'exploitation
- Les compilateurs et éditeurs fonctionnent en mode utilisateur, ils peuvent être changes

Qu’est ce qu'un système d'exploitation ?

- Un système d'exploitation est un ensemble de procédures manuelles et automatiques qui permet a un groupe d'utilisateurs de partager efficacement un ordinateur

- Un système d'exploitation est un ensemble de procédures cohérentes qui a pour but de gérer la pénurie de ressources

- Un système d'exploitation est un ensemble de programmes et de fonctions conçu pour faciliter et optimiser l'utilisation des unités physiques de l'ordinateur

- Le seul programme qui tourne constamment dans une machine

- II existe plusieurs systèmes d'exploitation. Ils varient selon :
  - le type de matériel
  - la complexité des taches a effectuer
  - les logiciels qu'ils doivent supporter

Le rôle d'un système d'exploitation

- Fournir a I 'utilisateur l'équivalent d'une machine étendue ou virtuelle plus simple a programmer que la machine réelle
- *Gérer* de manière équitable et optimale I 'allocation des processeurs, de la mémoire et des périphériques aux différents programmes qui les sollicitent

- Exemples de systèmes d'exploitation :
  - Mac-OS (le système Macintosh)
  - Windows (NT, 95, 98)
  - Unix, Linux
  - etc.

Le système d'exploitation Linux

C'est quoi Unix ?

- Unix est ne au début des années 70 dans les laboratoires Bell
- Unix est un système :
  - multi-utilisateurs : plusieurs personnes peuvent partager les ressources de la même machine
  - multitâches : plusieurs programmes ou logiciels peuvent s'exécuter concurremment
- II existe plusieurs versions commerciales :
  - AIX de IBM
  - Sun Solaris de SUN Microsystems
  - HP-UX de Hewlett Packard
  - Tru64 Unix de Compaq
  - etc.
- Plusieurs versions d'UNIX sont nées pour PC :
  - Linux
  - FreeBSD
  - OpenBSD
  - NetBSD ...

Un peu d'histoire

1969 - 1979 ***: les premiers pas universitaires***

■ Été 1969 : Ken Thompson, aux BELL Laboratories, écrit la version expérimentale cTLinux : système de fichiers exploité dans un environnement mono-utilisateur, multi-tâche, le tout étant écrit en assembleur.

■ 1ère justification officielle : traitement de texte pour secrétariat. Puis : étude des principes de programmation, de réseaux et de langages.

Un peu d'histoire

- **Eté 1973 : réécriture du noyau et des utilitaires d'Linux en C.**
- **En 1974 distribution Linux aux Universités (Berkeley et Columbia notamment). Il se compose alors :**

  d'un système de fichiers modulaire et simple,

  d'une interface unifiée vers les périphériques par l'intermédiaire du système de fichiers,
  du multi-tâche,

  et d'un interpréteur de commandes flexible et interchangeable.

- **LINUX est une implantation libre des spécifications POSIX avec des extensions System V (AT&T) et BSD (Berkeley),**
- **En 1991, Linus B. Torvalds (Helsinki) utilise MINIX.**
- **Août 1991 : 1ère version de LINUX 0.01. C'est une réécriture de MINIX, avec des ajouts de nouvelles fonctionnalités et la diffusion des sources sur « Internet »,**

Un peu d'historique

1979 - 1984 : les premiers pas commerciaux

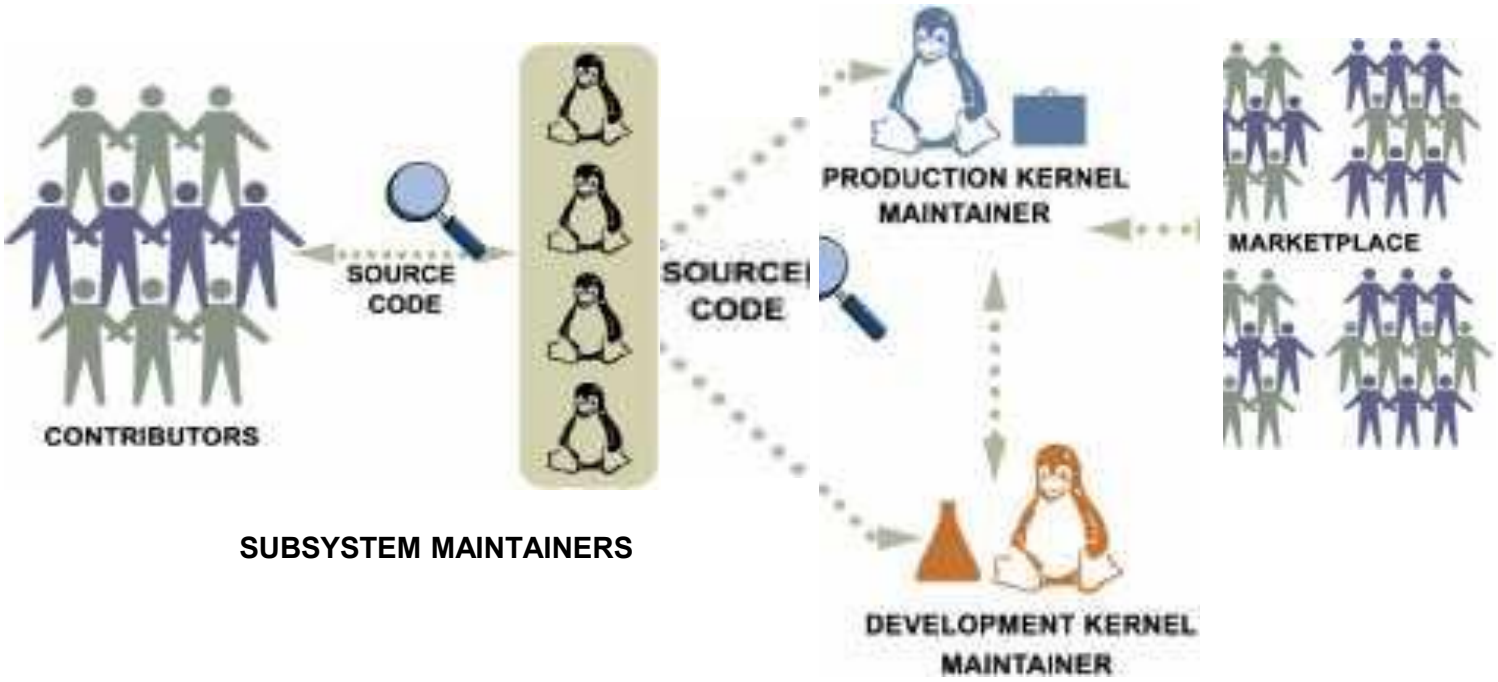
- **En 1979, avec la version 7, Linux se développe commercialement**

1984 - 1993... : la standardisation

- **En 1984 le Système V.2 est adopté comme standard,**
- **En 1985 AT&T publie SVID (System V Interface Definition) qui définit l'interface d'application du Système V.2 et non pas son implémentation,**
- **En 1986, le Système V.3 apporte les Streams, les librairies partagées et RFS (Remote File Sharing),**
- **En 1993, X/Open lance le COSE (Common Open Software Environment). Il s'agit d'accords entre constructeurs pour le développement d'applications dans un environnement commun.**

2. Présentation de Linux

Comment est maintenu le projet Linux ?



SUBSYSTEM MAINTAINERS

- Examen continu par les pairs du code
- Disponible en ligne en permanence pour examen public

Développement du noyau Linux

Si au début de son histoire le développement du noyau Linux était assuré par des développeurs bénévoles, les principaux contributeurs sont aujourd'hui un ensemble d'entreprises, souvent concurrentes, comme Red Hat, Novell, IBM ou Intel.

La licence du noyau Linux est la licence publique générale GNU. Cette licence est libre, ce qui permet d'utiliser, copier et modifier le code source selon ses envies ou ses besoins. Ainsi, quiconque a les connaissances nécessaires peut participer aux tests et à révolution du noyau.

2. Présentation de Linux Le projet GNU

Principe de base : le libre accès au code source accélère le progrès en matière d'informatique car l'innovation dépend de la diffusion du code source

La liberté au sens GNU est définie selon quatre principes (le copyleft GPL) :

- **liberté d'exécuter le programme, pour tous les usages**
- **liberté d'étudier le fonctionnement du programme, de l'adapter à ses besoins**
- **liberté de redistribuer des copies**
- **liberté d'améliorer le programme et de publier ses améliorations, pour en faire profiter toute la communauté**

2. Présentation de Linux

Mode de numérotation

Les numéros de version du noyau sont composés de trois chiffres : le premier est le numéro majeur, le second le numéro mineur.

Avant l'apparition des versions 2.6.x, les numéros mineurs pairs indiquaient une version stable et les numéros mineurs impairs une version de développement. Ainsi, les versions 2.2, 2.4 sont stables, les versions 2.3 et 2.5 sont des versions de développement.

Depuis la version 2.6 du noyau, ce modèle de numérotation stable/développement a été abandonné et il n'y a donc plus de signification particulière aux numéros mineurs pairs ou impairs. Le troisième chiffre indique une révision, ce qui correspond à des corrections de bogues, de sécurité ou un ajout de fonctionnalité.

2. Présentation de Linux La licence GPL

La licence GPL (General Public licence) www.gnu.oracopvleft/gpl.html

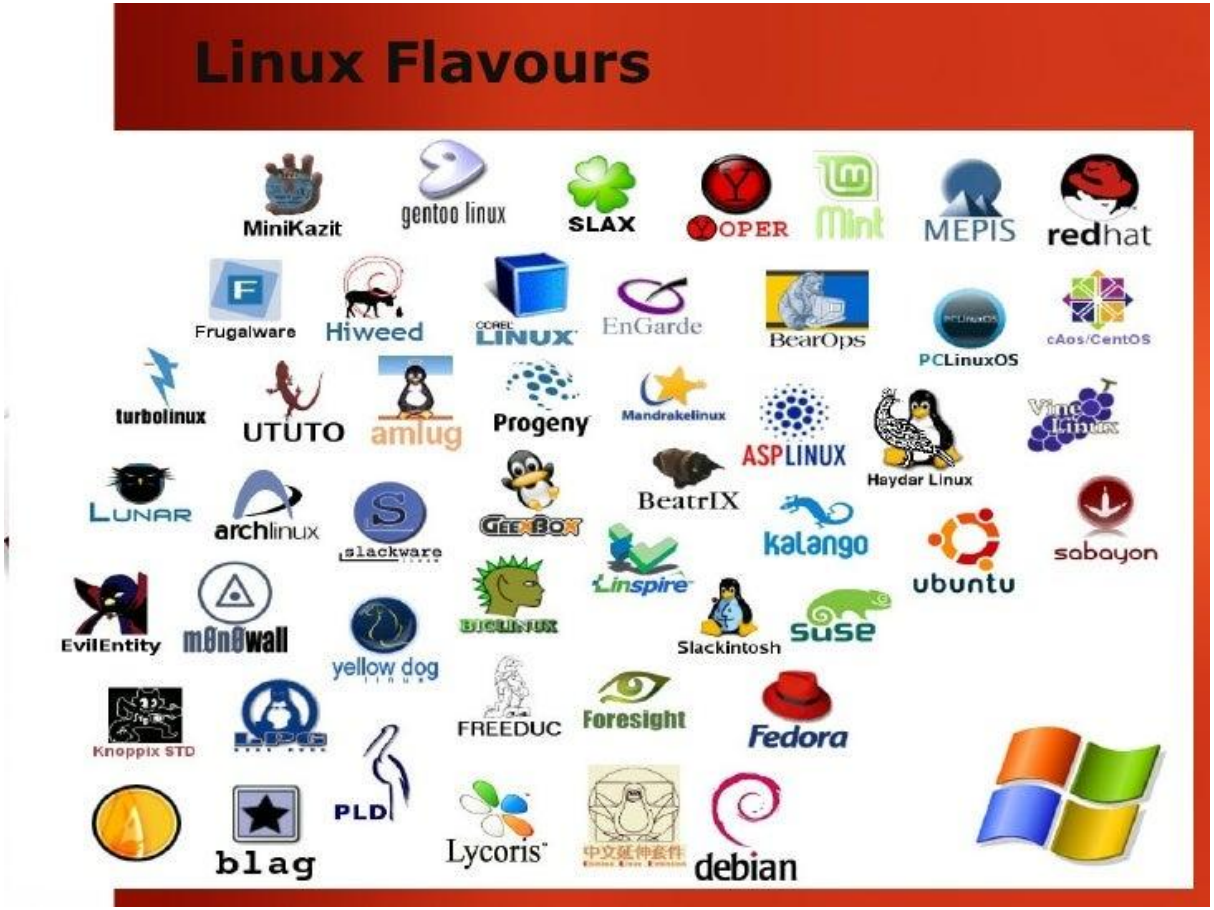
- **Autorise l'utilisateur à copier et distribuer à volonté le logiciel qu'elle protège, pourvu qu'il n'interdise pas à ses pairs de le faire aussi,**
- **Requiert aussi que tout dérivé d'un travail placé sous sa protection soit lui aussi protégé par elle,**
- **Quand la GPL évoque les logiciels libre, elle traite de liberté et non de gratuité (un logiciel GPL peut être vendu),**
- **Remarque : en anglais « free » mélange gratuité et liberté.**

Présentation de Linux - Distributions Linux

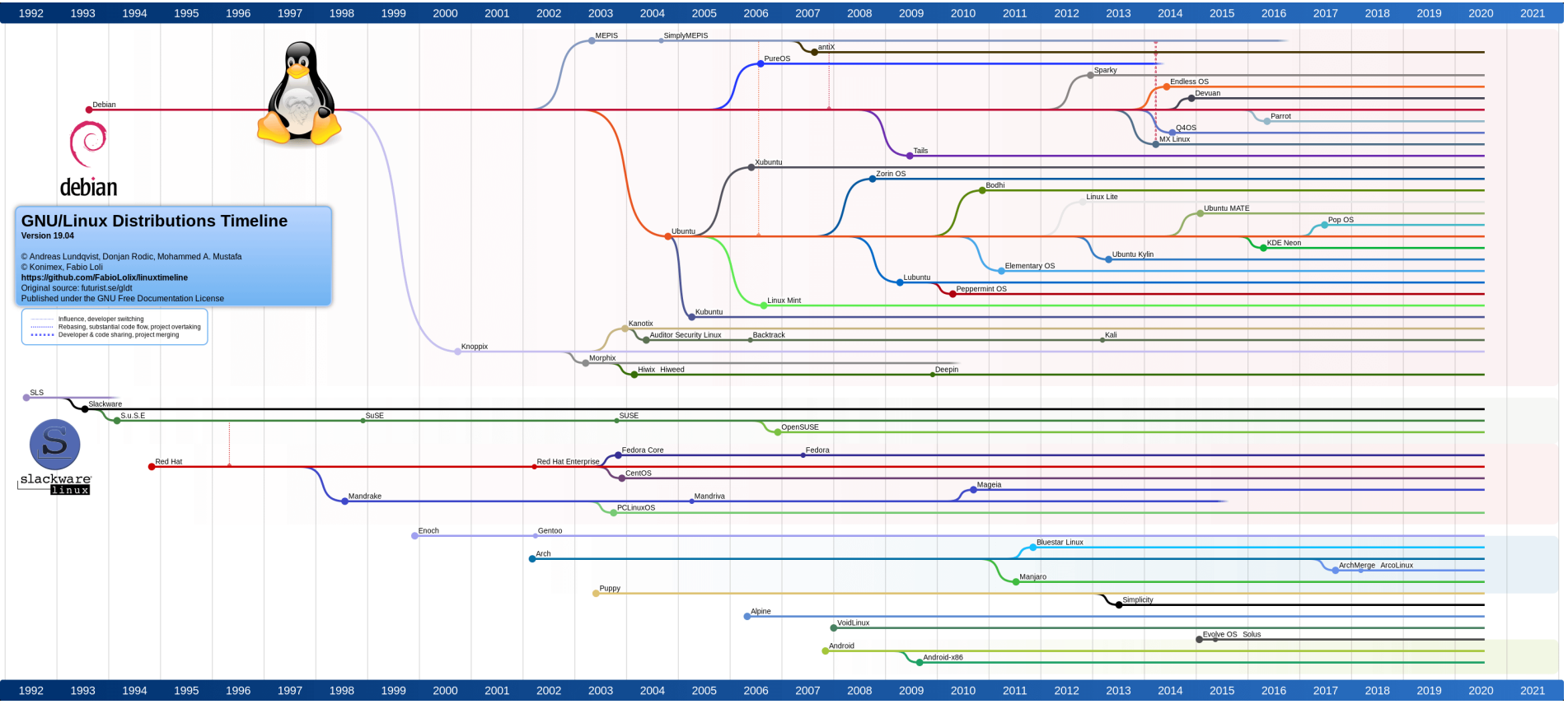
Une distribution est un noyau auquel des logiciels ont été ajoutés

Possibilités de créer des distributions dédiées à un usage particulier

Les distributions les plus connues



GNU/Linux Distributions Timeline



Structure de Linux

□ **Les utilitaires sont des outils d'interfaçage avec le système, de programmation et de communication.**

□ **Les shells les plus connus sont : BOURNE SHELL (sh)**

KORN-SHELL (ksh)

C-SHELL

TC-SHELL

BOURNE AGAIN SHELL (bash)

Le système Unix/Linux : Caractéristiques

- Multi-plateforme
- Multitâches et multi-utilisateurs
- Un système ouvert
- Différents environnements de commandes : les shells
- Système de fichiers hiérarchique
- Gestion hiérarchisée des processus

Organisation du système Unix/Linux

Le système Unix est organisé en couches :

- Noyau : la couche de plus haut niveau, elle assure la communication avec le matériel. Le noyau s'occupe de :
- la gestion de la mémoire,
- L'accès aux périphériques (disque dur, lecteur de CD-Rom, clavier, souris,...),
- la gestion du réseau,...

- Shell : interprète les ordres de I 'utilisateur et les fait exécuter par le noyau. Les ordres peuvent être passés soit directement au clavier, soit en utilisant des outils graphiques de plus haut niveau

- Applications : interagissent avec I 'utilisateur ou avec d'autres applications et communiquent avec le shell ou avec le noyau

C'est quoi Linux ?

- Nous avons I 'habitude de dire que Linux est un système d'exploitation
- Linux n'est en réalité que le noyau du système d'exploitation

  GNU/Linux (GNU = GNU is Not Unix)
- GNU est un projet qui a apporte des tas d'utilitaires au noyau Linux :
  - compilateur gcc,
  - tar, man, bash ...

Linux est un noyau

GNU est un ensemble de programmes utilitaires

GNU/Linux est le système d'exploitation

Quelques liens :

- Page officielle de Linux : http://www.linux.org
- Page officielle de GNU : http://www.gnu.org

Logiciel Libre

- Liberté d'utiliser le logiciel pour n'importe quelle utilisation
- Liberté d'étudier le fonctionnement du logiciel et de l'adapter à ses besoins
- Liberté de redistribuer des copies du logiciel
- Liberté d'améliorer le logiciel et de distribuer publiquement ces
- améliorations pour le bénéfice de toute la communauté

Pourquoi utiliser les logiciels libres ?

- C’est le domaine des technologies de l’information qui possède une histoire cohérente et continue
- Ce sont les systèmes alternatifs qui permettent une analyse comparative critique
- C’est un moyen d’acquérir des outils et des méthodes originales
- C’est un moyen de s’approprier les nouvelles technologies de façon autonome et indépendante

LINUX : STATISTIQUES ET TENDANCES 2024

Analyse des évolutions clés et des données actuelles

Introduction à Linux et son importance

Importance et adoption

Domination des serveurs

Linux détient 45 % du marché mondial des serveurs, assurant robustesse et performance pour les infrastructures critiques.

Présence mobile

Android, basé sur Linux, équipe 85 % des smartphones, soulignant l'importance de Linux dans la mobilité.

Hébergement web dominant

Linux alimente 96,3 % des serveurs web parmi le million les plus visités, démontrant son omniprésence en ligne.

Communauté et distributions

Une communauté dynamique soutient Linux avec des distributions populaires comme Ubuntu et Debian, favorisant la sécurité et la scalabilité.

Croissance et marché

Expansion économique et projections

Croissance du marché Linux

Le marché mondial de Linux devrait atteindre 15,64 milliards de dollars en 2027 avec un taux de croissance annuel de 19,2 %.

Domination dans le cloud computing

Linux alimente plus de 90 % des charges de travail dans le cloud public, renforçant sa position stratégique mondiale.

Part de marché dans les conteneurs

Linux détient 85 % du marché des conteneurs, soulignant son rôle clé dans les environnements virtualisés modernes.

Impact sur le marché mobile

Android, basé sur Linux, représente environ 70 % des systèmes d’exploitation mobiles, confirmant son influence majeure.

Popularité et communauté

Adoption par les développeurs et contributions

Popularité auprès des développeurs

Près de 47 % des développeurs professionnels utilisent Linux pour sa flexibilité et ses fonctionnalités avancées.

Communauté de contributeurs active

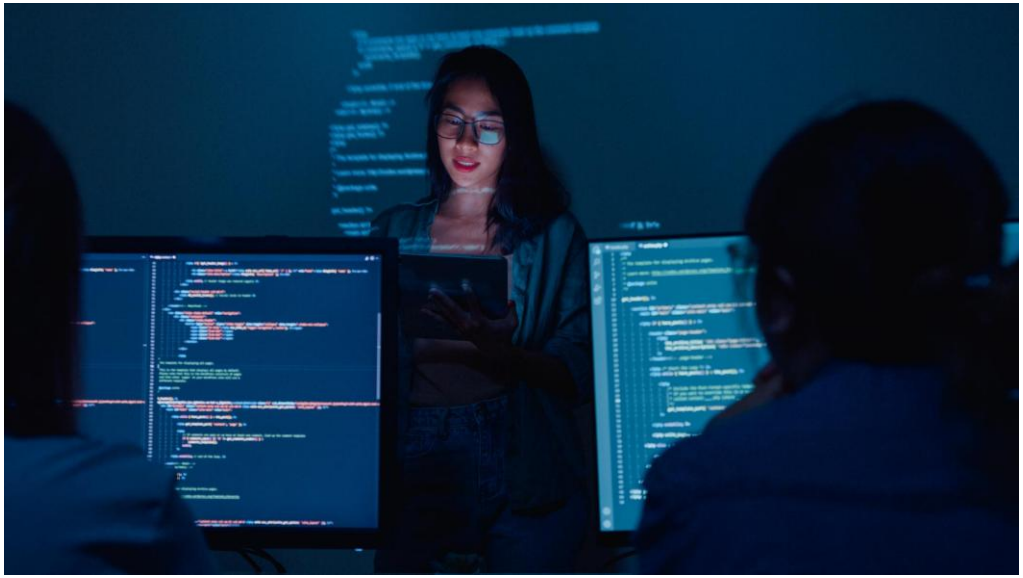
Plus de 25 000 contributeurs participent au noyau Linux, ajoutant environ 10 000 lignes de code chaque jour.

Évolution et améliorations continues

La dynamique communautaire favorise de nouvelles distributions et améliore la sécurité ainsi que la performance de Linux.

Utilisation professionnelle diverse

Linux est reconnu pour sa stabilité et son adaptation aux serveurs, systèmes embarqués et autres environnements professionnels.



Tendances futures

Innovations et perspectives

Linux à l'Edge et IoT

Plus de 75 % des appareils Edge fonctionneront sous Linux d'ici 2025, renforçant son rôle dans l'IoT et les systèmes décentralisés.

IA et apprentissage automatique

L'intégration de l'intelligence artificielle et du machine learning dans Linux offre des capacités avancées pour l'analyse de données.

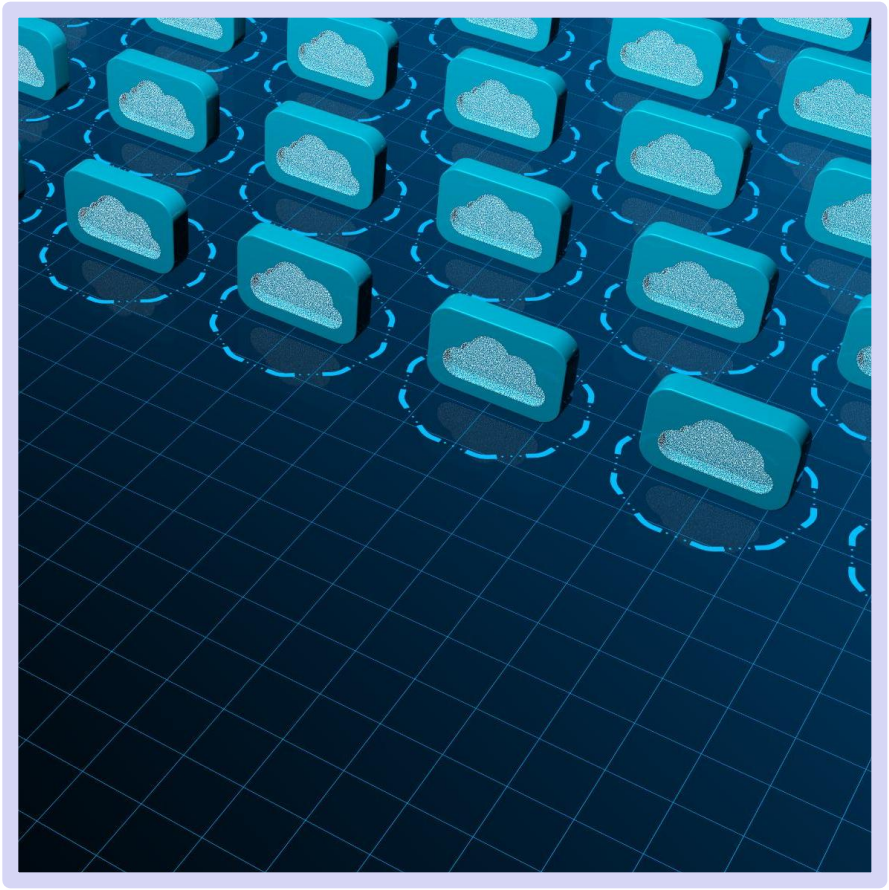
Croissance du gaming Linux

Le gaming sous Linux progresse grâce à des initiatives améliorant la compatibilité des jeux, comme SteamOS et Proton.

Sécurité et conteneurisation

Les distributions immuables et les technologies de conteneurisation comme Docker et Kubernetes offrent plus de sécurité et stabilité.

Conclusion et points clés



Synthèse des éléments majeurs

Présence dominante de Linux

Linux détient une forte part de marché dans les serveurs, les smartphones et l'hébergement web mondial, soulignant son importance technologique.

Atouts de Linux

Open-source, sécurisé et économique, Linux est la solution privilégiée pour les entreprises et développeurs innovants.

Rôle stratégique futur

Linux joue un rôle clé dans l'IA, le Edge computing et le gaming, avec une croissance soutenue par une communauté active.

LINUX

STATISTIQUES ET TENDANCES EN 2024

IMPORTANCE ET ADOPTION

45% part du marché des serveurs

85% des smartphones

600+ distributions actives

100 % des superordinateurs

CROISSANCE ET MARCHÉ

15,64 Mrds $ d'ici 2027

19,2% Croissance de 19,2% par

70% du marché mobile

85% des conteneurs

POPULARITÉ ET COMMUNAUTÉ

47% des développeurs

80 millions de lignes de cle

25 000 contributeurs

25 000 contributeurs

TENDANCES FUTURES

AI

AI et Edge computing

Jéux vidéo

Sécurité

Sècurité

Installation Linux

1. Téléchargement et installation de VirtualBox https://www.virtualbox.org/
2. Téléchargement de image Ubuntu - https://www.linuxvmimages.com/images/ubuntu-2404/
3. Chargement de l'image - https://ubuntu.com/tutorials/how-to-run-ubuntu-desktop-on-a-virtual-machine-using-virtualbox#1-overview
4. Démarrage de l'installation
5. Login

La première fois !

- Une fois connecté, le système nous connaît,
  ouvre une session à notre nom et attend nos instructions via un programme spécial:
- Le Shell = interpréteur de commandes
  - interface utilisateur “de base” (interlocuteur avec le système)
  - interprétation ligne à ligne
  - plusieurs shells: sh, csh, tcsh, bash, ksh, zsh, ...
  - configurable: fichiers d’environnement (commençant par un “`.`”)
    - “**`.login`**”
    - “**`.logout`**”
    - “**`.bashrc`**”
  - langage de programmation

- shell par défaut : bash

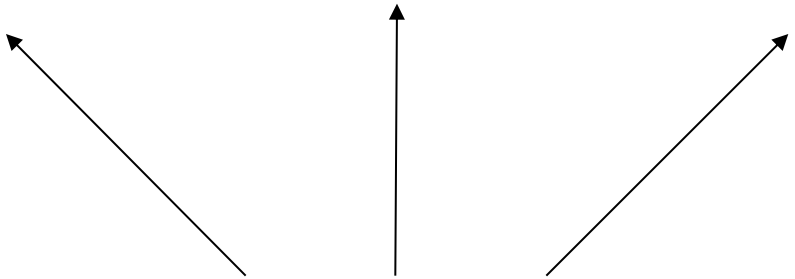
Les utilisateurs et les groupes

- Linux est multi-utilisateurs
- Chaque utilisateur a un login = nom identifiant unique
- Un utilisateur particulier : root
- Rôle du root :
  - Administrer la machine
  - Configurer les comptes utilisateurs
  - Résoudre les problèmes systèmes
- Les utilisateurs sont organisés dans des groupes
  - s'ils partagent les mêmes fichiers
  - S'ils ont les mêmes responsabilités
  - S'ils appartiennent au même service

Initiation au shell

- Format général des commandes:

commande [-option(s)] [argument(s)]



Exemple : **ls –a *.txt**

Exemples

- **`date`**
- **`touch fic`**

  crée un fichier fic
- **`hostname`**
- **`whoami`**

  affiche le nom de login de l'utilisateur connecté
- **`echo`**

  affiche un message (`echo "bonjour !"`)
- **`ls`**

  liste le contenu d'un répertoire
- **`man <cde>`**

  manuel en ligne de la commande <cde>

Syntaxe des commandes (1)

- Une commande Unix peut avoir 0, un nombre fixe ou un nombre variable d'arguments, plus un certain nombre d'options. Une option commence en général par un tiret (-) et deux tirets à la suite (—) indiquent la fin des options sur une ligne de commande

[hal@home/bob] date

Une commande sans paramètre qui affiche la date et l'heure courante

[hal@home/bob] cal 2006

Une commande avec un paramètre (une année) qui affiche le calendrier complet de l'année en question

[hal@home/bob] cal 9 2006

La même commande avec deux paramètres, le mois et l'année

[hal@home/bob] echo

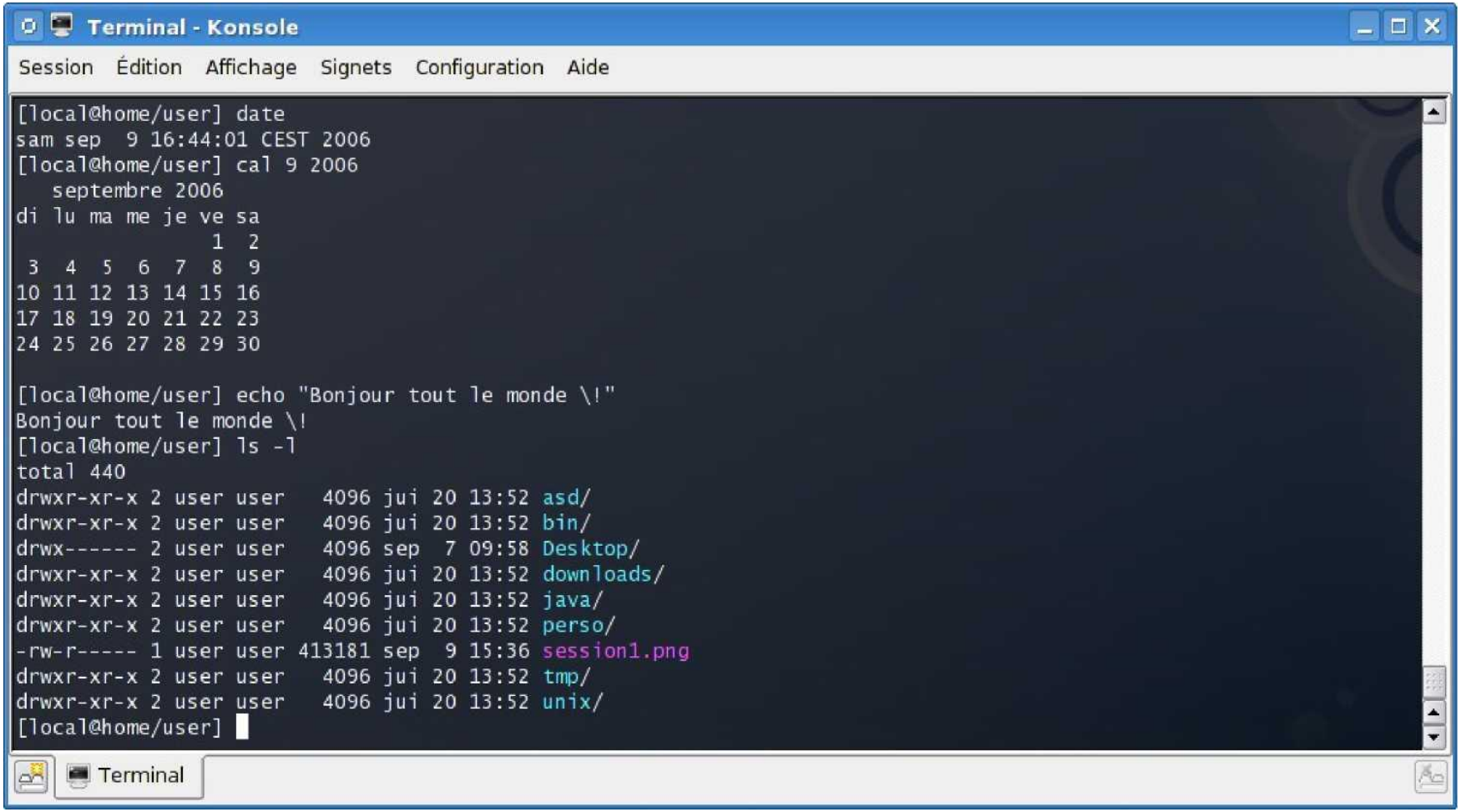
Sans argument cette commande affiche une ligne vide

[hal@home/bob] echo Bonjour tout le monde

Affiche les quatre arguments Bonjour, tout, le et monde, et va à la ligne

[hal@home/bob] echo -n Bonjour tout le monde L'option **-n** empêche le passage à la ligne

Syntaxe des commandes (2)



Exercices

Testez quelques commandes simples

1. Affichez la date courante avec la commande date
2. Affichez le calendrier de l'année courante avec cal 2012
3. Que fait la commande cal 9 2008 ? la commande cal ?
4. Testez la commande echo en évaluant echo Bonjour tout le monde
5. Que fait la commande echo sans argument ?
6. Essayez d'affichez le message 3 > 2 est vrai à l'aide la commande echo
7. Recommencez en faisant précéder le caractère > du caractère \ *(anti-slash)*
8. Recommencez en utilisant le caractère ' pour délimiter la chaîne de caractères
9. Recommencez en utilisant le caractère ” pour délimiter (d'une autre manière) la chaîne de caractères
10. Affichez le message précédent mais sans aller à la ligne (utilisez l'option -n)

Edition des commandes (1)

L'édition d'une commande sous **bash** est possible pendant la saisie même de la commande, à l'aide de caractères spéciaux qui sont identiques à certains raccourcis clavier de **Emacs/XEmacs**. Lors de la saisie d'une commande, on est automatiquement en mode insertion, et on dispose des fonctionnalités suivantes :

Déplacement du curseur

- déplacement du curseur à gauche d'un caractère : flèche gauche ou <ctrl-b>
- déplacement du curseur à droite d'un caractère : flèche droite ou <ctrl-f>
- déplacement du curseur d'un mot à gauche : <meta-b>
- déplacement du curseur d'un mot à droite : <meta-f>
- déplacement du curseur en début de ligne : <ctrl-a>
- déplacement du curseur en fin de ligne : <ctrl-e>

Edition des commandes (2)

Effacement de caractères

- effacement du dernier caractère saisi (à gauche du curseur) : <backspace>

  effacement du caractère courant (sous le curseur) : <suppr> ou <ctrl-d>

  effacement du mot courant (à partir du curseur) : <meta-d>
- effacement du texte du début de ligne jusqu'au curseur: <ctlr-u>
- effacement de la ligne du curseur à la fin de la ligne : <ctlr-k>

Insertion des caractères supprimés

Les caractères supprimés par les commandes précédentes peuvent être insérés à droite du curseur par <ctlr-y>

Edition des commandes (3)

Historique des commandes

Le shell **bash** gère un historique des commandes saisies et on peut se déplacer dans cet historique

ré-édition de la commande précédent la commande courante dans l'historique : flèche haut ou <ctrl-p>

ré-édition de la commande suivant la commande courante dans l'historique : flèche bas ou <ctrl-n>

Complétion

Le shell **bash** comprend un mécanisme de complétion pour les noms de commandes comme pour les noms de fichiers

complétion simple : <tab> à la suite des caractères déjà saisis

complétion multiple : <tab><tab> à la suite des caractères déjà saisis

Exercices

Testez à nouveau et séquentiellement les commandes suivantes, mais cette fois en utilisant **intensivement** l'édition en ligne des commandes :

1. Affichez la date courante avec la commande date
2. Affichez le calendrier de l'année courante avec cal 2012
3. Que fait la commande cal 9 2008 ? la commande cal ?
4. Testez la commande echo en évaluant echo Bonjour tout le monde
5. Que fait la commande echo sans argument ?
6. Essayez d'affichez le message 3 > 2 est vrai à l'aide la commande echo
7. Recommencez en faisant précéder le caractère > du caractère \ *(anti-slash)*
8. Recommencez en utilisant le caractère ' pour délimiter la chaîne de caractères
9. Recommencez en utilisant le caractère " pour délimiter (d'une autre manère) la chaîne de caractères
10. Affichez le message précédent mais sans aller à la ligne (utilisez l'option -n)

Aide sur les commandes

La plupart des commandes de Linux proposent au moins un paramètre d'aide

-h

(- est surtout utilisé pour introduire des options en 1 caractère)

—help

(— est toujours utilisé pour introduire l'option "longue" correspondante, qui rend les scripts plus faciles à comprendre)

Les commandes affichent souvent un court résumé des options disponibles quand vous utilisez un argument invalide.

Pages de manuel

man <mot_clé>

Affiche une ou plusieurs pages de manuel pour <mot_clé>

man man

La plupart des pages de manuel disponibles concernent des commandes Unix, mais aussi des fonctions, entêtes ou structures de données de bibliothèques C, ou même des fichiers de configuration du système

man stdio.h

man fstab (pour /etc/fstab)

Les pages de manuel sont recherchées dans les répertoires spécifiées par la variable d'environnement MANPATH

Exercices

1. A l'aide d'internet, trouvez le nom de la commande Linux qui permet de compresser des fichiers en utilisant l'algorithme de compression de Burrows Wheeler
2. Affichez la page de manuel de cette commande
3. Trouvez le numéro de la version de cette commande actuellement installée sur votre ordinateur

Recherches sur Internet

Résolution de problèmes

La plupart des archives des forums et des listes de discussion sont publiques, et indexées très régulièrement par Google

Si vous recherchez la cause d'un message d'erreur, copiez-le mot à mot dans le formulaire de recherche, entour é par des guillemets. Il est très probable que quelqu'un d'autre aura déjà rencontré le même problème

N'oubliez pas d'utiliser Google Groups: http://groups.google.com/ Ce site indexe plus de 20 années de messages de groupes de discussion

Maintenant, il y a bien sur l'usage de l'IA qui est devenue incontournable (chatGPT, Copilot, gemini, etc…)

Exercices

Testez l'aide disponible sur les commandes en effectuant les opérations suivantes :

1. Affichez l'aide de la commande cal
2. Affichez le calendrier de l'année 2000
3. Affichez la page de manuel de la commande date
4. Utilisez la commande touch pour créer un fichier vide de nom vide dans votre répertoire de travail linux

Principes et commandes de base

- Système de fichiers
- Commandes de base
- Liens symboliques et physiques
- Droits

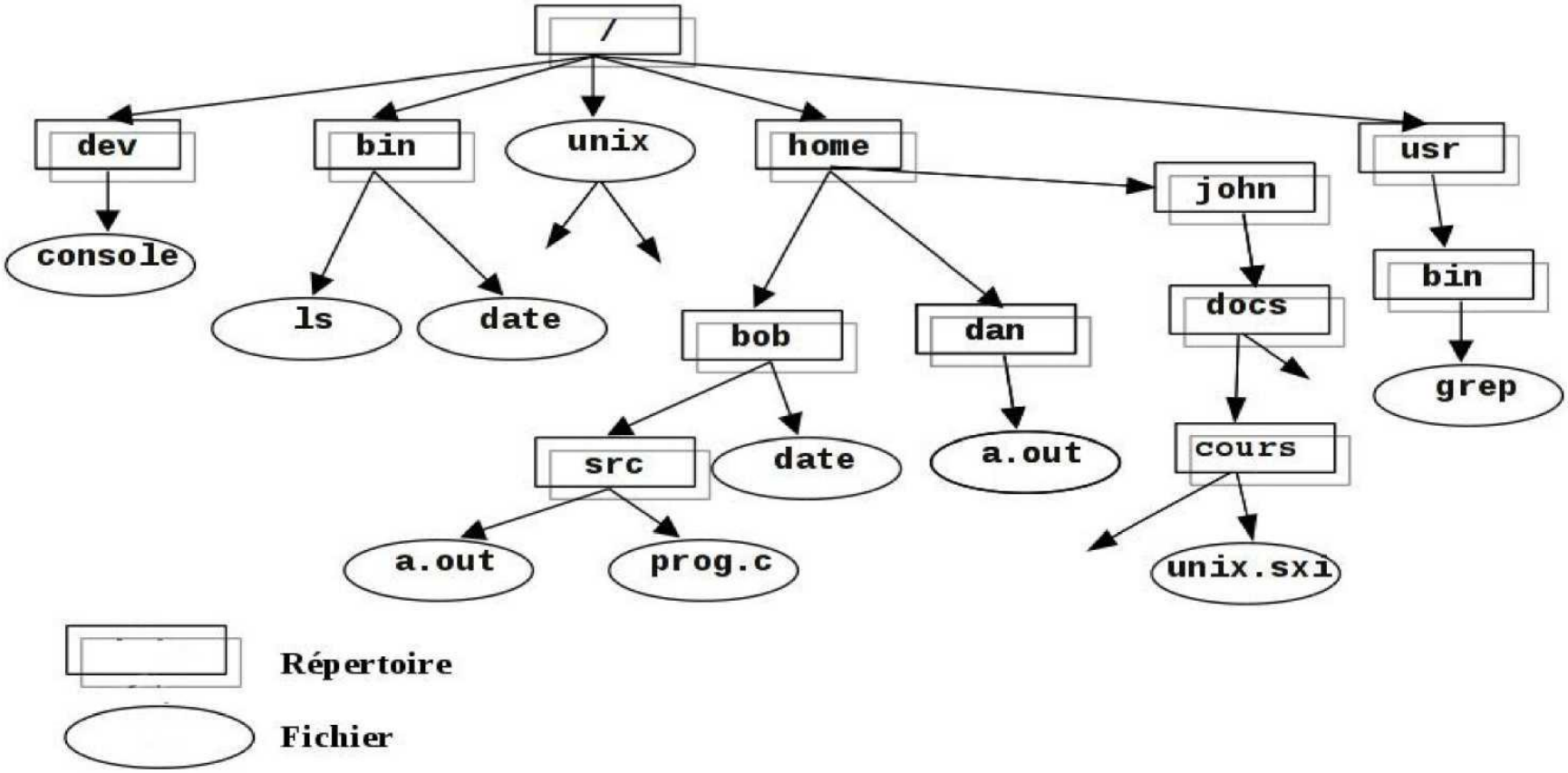
Fichiers et répertoires

Presque tout dans Unix / Linux est un fichier

- Même les répertoires sont des fichiers, contenant une liste de fichiers et de répertoires.
- Depuis le début d'Unix, aucune limitation majeure quant à la longueur d'un nom de fichier. Tout caractère (les espaces en particulier) peut être utilisé dans le nom, et les extensions sont facultatives. Les différences de majuscules ou de minuscules constituent des fichiers distincts.
- Un *chemin* («path») est une séquence de répertoires imbriqués, avec un fichier ou un répertoire à la fin, séparés par le caractère /
- Chemin relatif: docs/cours /unix. Html : Relatif au répertoire courant
- Chemin absolu: /home/john/docs/cours/unix.html : Chemin depuis le répertoire racine du système (/)

Système de fichiers Linux (1)

Structure arborescente



Système de fichiers Linux (2)

Rien d'imposé par le système. Peut varier d'un système à l'autre, même entre deux installations de Linux!

/	**Répertoire racine**
/bin/	**Commandes de base du système**
/boot/	**Images, initrd et fichiers de configuration du noyau**
/dev/	**Fichiers représentant des périphériques**
/dev/hda	**premier disque dur IDE**
/etc/	**Fichiers de configuration du système**
/home/	**Répertoires utilisateurs**
/lib/	**Bibliothèques de base du système (partagées)**

Système de fichiers Linux (3)

/lost+found	Fichiers détériorés que le système a essayé de récupérer
/mnt/	Systèmes de fichiers montés /mnt/usbdisk/, /mnt/windows/ ...
/opt/	Outils spécifiques installés par l'administrateur. Souvent remplacé par /usr/local/
/proc/	Accès aux informations du système /proc/cpuinfo, /proc/version ...
/root/	Répertoire utilisateur de l'administrateur
/sbin/	Commandes réservées à l'administrateur
/sys/	Contrôle du système et des périphériques (fréquence du processeur, gestion de l'alimentation des périphériques, etc.)

Système de fichiers Linux (4)

/tmp/ **Fichiers temporaires**

/usr/ **Programmes utilisateurs ordinaires, non essentiels au système : /usr/bin/, /usr/lib/,…**
Outils spécifiques installés par l'administrateur. (alternative : /opt/)

/var/ **Données utilisées par le système ou ses serveurs /var/log/, /var/spool/mail (courrier entrant), /var/spool/lpd (travaux d'impression)...**

Répertoires spéciaux (1)

Le répertoire relatif . /

- **Le répertoire courant. Utile pour les commandes qui ont un répertoire comme argument. Egalement utile parfois pour lancer des commandes dans le répertoire courant (voir plus tard)**

Ainsi . /lisezmoi. txt et lisezmoi. txt sont équivalents

Le répertoire relatif ../

Le répertoire parent. Fait partie toujours partie du répertoire . (voir ls -a).

Unique référence au répertoire parent.

- **Utilisation la plus courante: cd . .**

Répertoires spéciaux (2)

Le répertoire absolu ~ /

Pas vraiment un répertoire spécial. Les interpréteurs de commande le remplacent juste par le répertoire utilisateur de l'utilisateur courant

Ne peut pas être utilisé dans la plupart des programmes, car il n'est pas un vrai répertoire (voir la variable HOME)

Le répertoire absolu ~bob/

De façon analogue, remplacé par les shells par le répertoire utilisateur de l'utilisateur bob

Chemins absolus et relatifs

Un fichier peut être désigné de façon absolue ou relative

* **Un chemin absolu identifie de manière unique un fichier ou un répertoire dans l'arborescence :**

  /home/bob/src/a.out /usr/bin/grep

* **Un chemin relatif identifie un fichier ou un répertoire en fonction du répertoire courant (on est sous ~bob) :**

  src/a.out ../dan/a.out

La commande Is

Affiche la liste des fichiers dans le répertoire courant, en ordre alphanumérique, sauf ceux qui commencent par le caractère “. ”

ls -a («all»: tous)
Affiche tous les fichiers (y compris les fichiers . *)

ls -l (long)
Affichage en format long (type, date, taille, propriétaire, permissions)

ls -t (temps)
Affiche les fichiers les plus récents en premier

ls -S (“size”: taille)
Affiche les fichiers les gros en premier

ls -r («reverse»: inversé)
Affiche en ordre inverse

ls -ltr (les options peuvent être combinées)
Format long, les fichiers les plus récents à la fin

La commande ls possède une cinquantaine d’options !

Exercices

Testez la commande **ls** en affichant, depuis votre répertoire personnel initial *(home directory*), la liste de **tous** vos fichiers et sous-répertoires :

1. de cinq façons différentes (faites varier le paramètre de **ls**)
2. sous un format *condensé*
3. sous un format *long* (donnant le propriétaire, les permissions, la taille, ...)
4. en affichant aussi les fichiers cachés (dont le nom commence par un point)
5. avec un format long et en affichant les fichiers cachés, mais dans l'ordre alphabétique inverse
6. avec un format long et en affichant les fichiers cachés, mais du plus récent au plus ancien
7. avec un format long et en affichant les fichiers cachés, mais du plus ancien au plus récent

Les commandes cd et pwd

- **cd rep** ***(change directory)***

Fait de rep le nouveau répertoire courant

- **cd**

Sans argument, fait du répertoire utilisateur le nouveau répertoire courant

- **cd — (le caractère tiret : '-')**

Fait du répertoire précédent le répertoire courant (le dernier répertoire depuis lequel on a fait cd) le nouveau répertoire courant

- **pwd** ***(print working directory)***

Affiche le chemin absolu du répertoire courant

Les commandes mv et cp

mv ancien_nom nouveau_nom *(move)*

Change le nom du fichier ou du répertoire donné

mv -i *(interactive)*

Si le fichier existe déjà, demander confirmation à l'utilisateur

cp fichier_orig fichier_dest

Crée une copie d'un fichier d'origine

cp fichl fich2 fich3 . . . rep

Copie tous les fichiers vers le répertoire de destination (dernier argument)

cp -i *(interactive)*

Si le fichier de destination existe déjà, demander confirmation à l'utilisateur

cp -r rep_orig rep_dest *(recursive)*

Copie du répertoire tout entier

Les commandes rm et rmdir

rm fichl fich2 fich3 . . . *(remove)*

Supprime les fichiers donnés

rm -i fichl fich2 fich3 . . . *(interactive)* Demande toujours à l'utilisateur de confirmer les suppressions

rm -r repl rep2 rep3 . . . *(recursive)*

Suppression des répertoires donnés et de tout leur contenu

rm -f fichl fich2 fich3 . . . *(force*)

Suppression des fichiers donnés sans message ni question

rmdir repl rep2 rep3 ... *(remove directory)* Suppression des répertoires donnés seulement s'ils sont vides

La commande mkdir

mkdir rep *(make directory)*

Fabrique rep, un nouveau répertoire, dans le courant. Si rep existe déjà, provoque une erreur

mkdir repi rep2 ... repN Fabrique séquentiellement plusieurs répertoires

mkdir repi/rep2

Il faut que le répertoire repi existe déjà, sinon provoque une erreur

mkdir -p repi/rep2/.../repN *(parent)*

Fabrique les répertoires repi rep2 ... repN-1 si besoin, et enfin le répertoire repN

Exercices

Testez les commandes élémentaires relatives aux fichiers et des répertoires :

1. Créez un répertoire system sous votre répertoire de travail linux, puis un répertoire tpi sous system
2. Effacez le répertoire system avec la commande rmdir. Que constatez-vous ?
3. Après avoir effacé les répertoires tpi et system, créez à l'aide d'une seule commande les répertoires system, system/tpi, system/tp2
4. Renommez le répertoire system en test
5. Copiez un fichier de votre choix du répertoire /bin dans le répertoire test/tpi :
   a) depuis le répertoire /bin
   b) depuis le répertoire test/tpi
   c) depuis votre *homedir,* en utilisant des chemins absolus
   d) depuis votre *homedir,* en utilisant des chemins relatifs
6. Effacez à l'aide d'une seule commande les répertoires test/tpi et test/tp2

Substitutions des noms de fichiers

ls *txt

L'interpréteur remplace d'abord *txt par tous les noms de fichiers et de répertoires finissant par txt (y compris . txt), sauf ceux commençant par . et enfin exécute la ligne de commande l s

ls -d .*

Affiche tous les fichiers et les répertoires commençant par .
-d indique à ls de ne pas afficher le contenu des dossiers . *

ls ?.log

Affiche tous les fichiers dont le nom commence par un caractère et finit par . log

rm -r log*

Efface TOUS les fichiers et sous-répertoires du répertoire courant dont le nom commence par log

Caractères génériques

Plus généralement :

*** dénote 0, 1 ou plusieurs caractères**

***.c** **doc*unix.html**

? dénote un (et exactement un) caractère quelconque

?nix

[. . .] dénote un caractère appartenant à un ensemble

[Uu]nix *.sx[ci]

[^ . . .] dénote un caractère n'appartenant pas à un ensemble

•Tco] **[[:upper:]]**

Exercices (1)

Placez-vous dans le répertoire /bin puis affichez la liste de tous les fichiers

1. dont le nom commence par la lettre t
2. dont le nom termine par la lettre h
3. dont le nom a exactement trois lettres
4. dont le nom contient la sous-chaîne un
5. dont le nom contient au moins un chiffre

Créez dans votre répertoire de travail linux (avec la commande touch) les fichiers de noms exact fich1, fich2, fichll, fich12, fichla, fichai, .fich1, . fich2, afich et toto, puis affichez :

6. les fichiers dont le nom commence par fich
7. les fichiers dont le nom commence par fich suivi d'un seul caractère
8. Les fichiers dont le nom commence par fich suivi d'un seul caractère qui est un chiffre
9. les fichiers dont le nom commence par le caractère . (point)
10. les fichiers dont le nom ne commence

Exercices (2)

Placez-vous dans le répertoire /etc, puis listez tous les fichiers et les noms de répertoires :

0. de configuration (i.e. dont le nom termine par le suffixe .conf)
1. dont le nom commence par la chaîne cron
2. dont le nom ne termine pas par le caractère d
3. Placez-vous dans votre répertoire linux , videz-le de son contenu, puis placez-y deux fichiers de noms toto et titi contenant respectivement les lignes toto et titi (utilisez un éditeur)
4. Devinez ce que va faire la commande cp t* et vérifiez en l'évaluant.
5. Construisez le répertoire tata (toujours sous le répertoire linux) puis devinez ce que va faire très exactement la commande mv t* tata . Vérifiez en l'évaluant.
6. Déplacez tous les fichiers placés sous tata dans linux
7. Fabriquez un fichier (avec la commande touch) de nom -i , puis effacez-le
8. Fabriquez un fichier de nom t*a , puis effacez-le (et n'effacez que lui !)
9. Fabriquez un fichier de nom toto<espace>titi puis effacez-le (et n'effacez que lui !)

Afficher le contenu de fichiers

Plusieurs façons d'afficher le contenu de fichiers

cat fichl fich2 fich3 . . . (concaténer)

Met bout à bout et affiche le contenu des fichiers donnés

more fichl fich2 … (plus de détails)

A chaque page, demande à l'utilisateur d'appuyer sur une touche pour continuer. Peut aussi aller directement à la première apparition d'un mot-clé (commande " / " suivie du mot-clé)

less fichl fich2 fich3 . . . (moins)

Un more plus puissant (le nom est un jeu de mot...) !

Ne lit pas le fichier entier avant de commencer à afficher Permet de remonter en arrière dans le fichier (commande " ? ")

Les commandes head, tail et cat

head [-n N] fichier *(tête)*

Affiche les N premières lignes (ou 10 par défaut) du fichier donné.
N'a pas besoin d'ouvrir le fichier en entier pour le faire

tail [-n N] fichier *(queue)*

Affiche les N dernières lignes (ou 10 par défaut) du fichier donné Ne charge pas tout le fichier en mémoire. Très utile pour les gros fichiers.

tail -f fichier *(follow*)

Affiche les 10 dernières lignes du fichier donné et continue à afficher les nouvelles lignes au fur et à mesure qu'elles sont rajoutées en fin de fichier. Très pratique pour suivre les rajouts à un fichier de journal ("log")

cat [-n] fichier

Affiche le contenu du fichier donn é. Avec l'option -n numérote chaque ligne affichée de 1 à n (le nombre de lignes)

La commande wc

w wc fichier *(word count)*

Affiche le nombre de lignes, de mots et de caractères du fichier fichier *

wc -c fichier

Affiche le nombre d'octets contenus dans le fichier fichier w wc -m fichier

Affiche le nombre de caractères contenus dans le fichier fichier w wc -l fichier

Affiche le nombre de lignes contenues dans le fichier fichier w wc -w fichier

Affiche le nombre de mots contenus dans le fichier fichier

La commande sort

sort fichier

Trie les lignes du fichier selon l'ordre des caractères et les affiche.

sort -r fichier (*reverse*)

Idem, mais en ordre inverse

sort -ru fichier *(unique)*

Idem, mais ne produit qu'une seule fois les lignes identiques.

Plus de possibilités seront abordées plus tard

Exercices

0. Affichez la première ligne du fichier nanorc de le répertoire */etc*
1. Affichez les 10 dernières lignes du fichier nanorc de le répertoire **`/etc`**
2. Visualisez le fichier /etc/fstab successivement avec les commandes cat, more et less
3. Visualisez le fichier /etc/fstab avec la commande less et sautez directement aux lignes contenant la chaîne de caractères ANSI (en utilisant / et n)
4. Testez la commande wc et ses différentes options sur le fichier /etc/fstab
5. Créez un fichier de texte (dans votre répertoire linux) contenant quelques caractères sur deux lignes mais ne terminant pas par un *newline* (ligne vide en fin de fichier) et comptez le nombre de caractères du fichier avec la commande wc
6. Recommencez en ajoutant cette fois un *newline* en fin de fichier
7. Testez la commande sort sur plusieurs fichiers comme /usr/lib/helix/README ou /var/www/icons/README
8. Testez les commandes cat, wc et sort sans arguments et en saisissant des lignes interactivement (tapez <ctrl-d> pour terminer)

La commande grep (1)

grep motif fichiers

Parcourt les fichiers donnés et affiche les lignes qui correspondent au motif spécifié.

grep erreur *.log

Affiche toutes les lignes contenant erreur dans les fichiers * . log

grep -i erreur *.log

Idem, mais indifférent aux majuscules et minuscules

grep -ri erreur .

Idem, mais récursivement dans . et ses sous-répertoires

grep -v info *.log

Affiche toutes les lignes des fichiers, sauf celles qui contiennent info

La commande grep (2)

Le motif (le premier paramètre) de grep est une expression régulière qui utilise des caractères semblables aux caractères génériques du *shell* mais dont l'interprétation diffère

grep 'es*ai' fichier

Affiche les lignes qui contiennent e suivi d'un ou plusieurs s et terminant par ai

grep 'es.ai' fichier

Affiche les lignes qui contiennent es suivi d'un caractère quelconque et terminant par ai

grep ' [0 - 9] ' fichier

Affiche les lignes qui contiennent au moins un caractère numérique

grep ' J e ' fichier

Affiche les lignes qui commencent par Je

grep ' \ . $ ' fichier

Affiche les lignes qui termine par le caractère .

grep -v ' [; : 0 - 9] ' fichier

Affiche les lignes qui ne contiennent ni chiffres, ni ; ou :

La commande find

Plus facile à expliquer par quelques exemples

`find . -name "*.pdf" [-print]`

Recherche tous les fichiers `*.pdf` dans le répertoire courant (`.`) et ses sous-répertoires. Vous devez utiliser les guillemets pour empêcher le shell de substituer le caractère `*`

`find docs -name "*.pdf" -exec xpdf {} \;`

Recherche tous les fichiers `*.pdf` dans le répertoire `docs` et les affiche l'un après l'autre

De nombreuses possibilités existent et il faut lire la page de manuel !

Cependant, les 2 exemples ci-dessus couvrent la plupart des besoins

La commande locate

Outil de recherche à base d'expressions régulières, essentiellement pour trouver l'emplacement de fichiers ou de répertoires

locate clef

Affiche tous les fichiers sur votre système contenant clef dans leur nom

locate "*.pdf"

Affiche tous les fichiers *.pdf existant sur votre système.

locate "/home/frigo/*mousse*"

Affiche tous les fichiers *mousse* dans le répertoire indiqué (chemin absolu)

locate est très rapide grâce à l'indexation de tous les fichiers dans une base de données dédiée, qui est mise à jour régulièrement.

locate est beaucoup moins utile que find qui est beaucoup plus adapté à la recherche fine de fichiers, et qui permet en outre d'effectuer des actions sur les fichiers trouvés.

Exercices

0 Affichez les lignes du fichier /etc/gai.conf qui contiennent le caractère

1. Affichez les lignes du fichier /etc/gai.conf qui commence par le caractère #

2. Affichez les lignes du fichier /etc/gai.conf qui contiennent des nombres séparés par des points (comme par exemple 3.1 ou 10.2.1)

3. Affichez les lignes du fichier /etc/mailcap qui contiennent le caractère $

4. Affichez les lignes du fichier /etc/profile qui terminent par le caractère \

5. Affichez tous les fichiers réguliers (les fichiers, et non pas les répertoires ou les liens) vous appartenant (i.e. se trouvant dans votre *homedir* ou récursivement dans un de ses sous-répertoires)

6. Affichez le nom de tous les répertoires (et seulement des répertoires) vous appartenant

7. Affichez tous les fichiers cachés (i.e. de nom . *) vous appartenant

8. Créez plusieurs fichiers d'extension .bak dans votre *homedir* et dans certains de ses sous-répertoires, oubliez leur emplacement, puis effacez-les tous à l'aide d'une seule commande

Comparaison de fichiers

cmp fichierl fichier2 Compare deux fichiers quelconques.

diff fichierl fichier2

Affiche les différences entre deux fichiers, ou rien si les fichiers sont identiques.

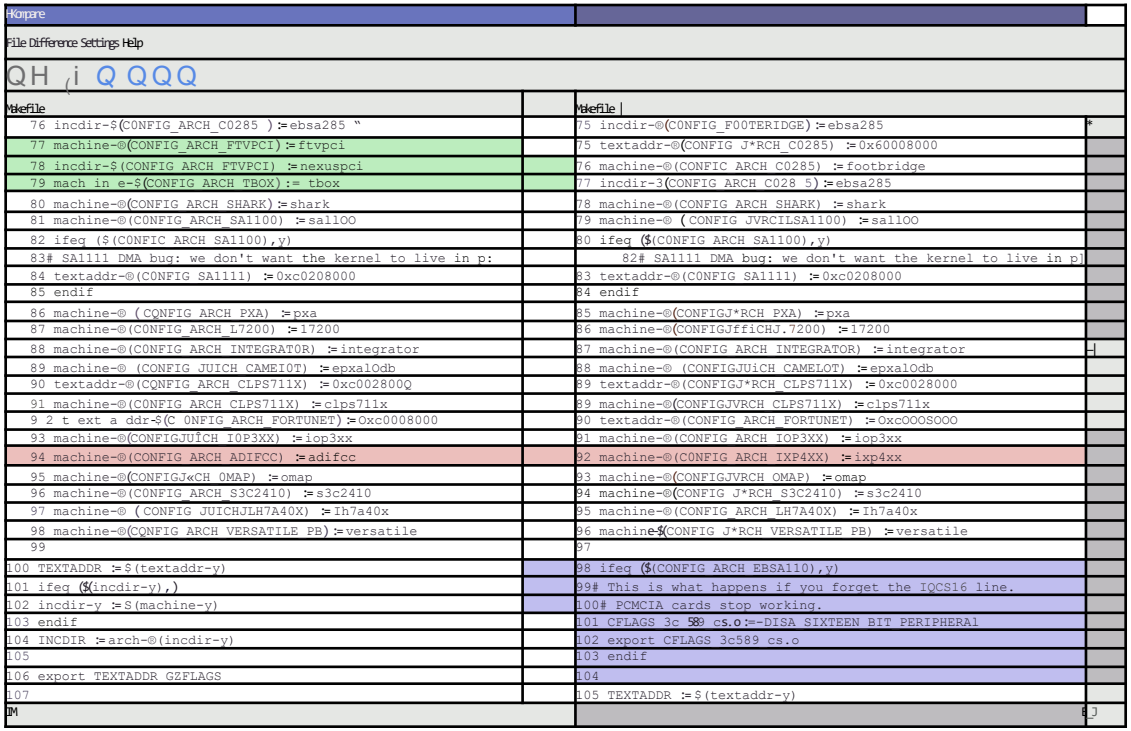
diff -r repl/ rep2/

Affiche les différences entre fichiers de même nom dans les deux répertoires.

Pour examiner en détail les différences, mieux vaut utiliser des outils graphiques

Le programme kompare

Un autre outil pratique pour comparer des fichiers et fusionner leurs différences



Mesure de la taille des fichiers

du -h fichier (disk usage)

-h: affiche la taille du fichier donné, sous forme lisible par un humain: K (kilo-octets), M (mega-octets) or G (giga-octets). Sinon du rend le nombre brut de blocs occupés par le fichier sur le disque (difficile à lire).

du -sh rep

- s : rend la somme des tailles de tous les fichiers dans le répertoire donné

Mesure de l'espace disque

df -h rep

Affiche des informations sur l'espace disque utilis é et disponible dans le système de fichiers qui contient le répertoire donné.

De même, l'option -h n'existe que dans GNU df.

Exemple :

```
> df -h .
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda5             9.2G  7.1G  1.8G  81% /
```

df -h

Affiche les informations d'espace disque pour tous les systèmes de fichiers disponibles sur le système. Quand des erreurs surviennent, utile pour vérifier si des systèmes de fichiers pleins.

Exercices

0. Copiez le fichier /etc/fstab dans votre répertoire de travail linux

1. Editez le fichier fstab , modifiez-en quelques lignes, ajoutez-lui une ligne et supprimez-lui une ligne

2. Testez les commandes cmp et diff sur les fichiers /etc/fstab et linux/fstab

3. Visualisez les différences entre les fichiers /etc/fstab et linux/fstab à l'aide de la commande kompare, que vous lancerez successivement :

a) par le menu 4 ^ Applications (sous-menu Développement)

b) par la commande kompare tapée dans un terminal

c) par le lanceur de commande (menu K puis action Exécuter une commande)

4. Affichez la taille en méga-octets occupée par votre *homedir*
5. Affichez la taille en kilo-octets du fichier /etc/fstab

6. Affichez les informations d'espace disque pour tous les systèmes de fichiers disponibles sur votre système

Compression

Très utile pour compacter de gros fichiers et économiser de la place

[un]compress fichier

Utilitaire de compression traditionnel d'Unix. Crée des fichiers . Z Seulement gardé pour raisons de compatibilité. Performance moyenne.

g[un]zip fichier

Utilitaire de compression GNU zip. Crée des fichiers . gz Assez bonne performance (un peu meilleur que Zip)

b[un]zip2 fichier

Le plus récent et le plus performant des utilitaires de compression. Crée des fichiers .bz2 . La plupart du temps 20-25% meilleur que gzip.
Utilisez celui-ci ! Maintenant disponible sur tous les systèmes Unix.

Archivage (1)

Utile pour sauvegarder ou publier un ensemble de fichiers

tar : à l'origine "tape archive" ("archive sur bande")

Création d'une archive:

tar cvf archive fichiers-ou-répertoires

c (créer) : pour fabriquer l'archive

v (verbeux) : utile pour suivre la progression de l'archivage f (fichier) : archive créée dans un fichier (et non sur une bande)

Exemple :

tar cvf /backup/home.tar /home

bzip2 /backup/home.tar

Archivage (2)

Afficher le contenu d'une archive ou vérifier son intégrité: tar tvf archive

Extraire tous les fichiers d'une archive: tar xvf archive

Extraire seulement quelques fichiers d'une archive: tar xvf archive fichiers-ou-répertoires Les fichiers ou répertoires sont donnés avec un chemin relatif au répertoire racine de l'archive.

Options supplémentaires GNU tar

GNU tar sous GNU / Linux

Permet de compresser et décompresser des archives au vol. Utile pour éviter de créer d'énormes fichiers intermédiaires. Plus facile à faire qu'en combinant tar et gip/bzip2

j : [dé]compresse au vol avec bzip2

z : [dé]compresse au vol avec gzip

Exemples

tar jcvf projet.tar.bz2 projet/

tar ztf projet.tar.gz

Exercices

0. Ouvrez votre navigateur et allez à l'URL https://tldp.org/guides.html
1. Téléchargez l'archive `abs-guide.html.tar.gz` (cliquer sur *Advanced Bash Scripting Guide)*
2. Créez le répertoire `unix/abs` et déplacez l'archive dans ce répertoire
3. Visualisez le contenu de l'archive sans la décompresser, ni l'extraire
4. Décompressez et extrayez l'archive en une seule commande
5. Créez un répertoire de nom votre *login*
6. Copiez sous ce répertoire tous les fichiers et sous-répertoires cachés de votre *homedir*
7. Fabriquez une archive compressée de ce répertoire (et non pas seulement du contenu de ce répertoire)

ATTENTION : ne décompressez jamais une archive, en particulier dans votre *homedir*, sans avoir auparavant listé son contenu, et vérifié mie son mntenn était bien dans un rénertoire

Liens symboliques

Un lien symbolique est un fichier spécial qui est juste une référence au nom d'un autre (fichier ou répertoire)

Utile pour simplifier et réduire l'utilisation du disque quand deux fichiers ont le même contenu.

Exemple:

/usr/local/bin/java -> /usr/jdk1.5.0_03/bin/java

Comment distinguer les liens symboliques:

ls -l affiche -> et le fichier référencé par le lien

GNU ls affiche les liens avec une couleur différente

Création de liens symboliques

Pour créer un lien symbolique (même ordre que dans cp) :

ln -s nom_fichier nom_lien

Pour créer un lien vers un fichier dans un autre répertoire, avec le même nom :

ln -s ../LISEZ_MOI.txt

Pour créer plusieurs liens d'un coup dans un dossier donné :

ln -s fich1 fich2 fich3 ... rep

Pour supprimer un lien : rm nom_lien

Bien-sûr, cela ne supprime pas le fichier référencé par le lien

Liens physiques

* **Par défaut, ln crée des *liens physiques***

* **Un *lien physique* vers un fichier est un fichier ordinaire, avec exactement le même contenu physique**

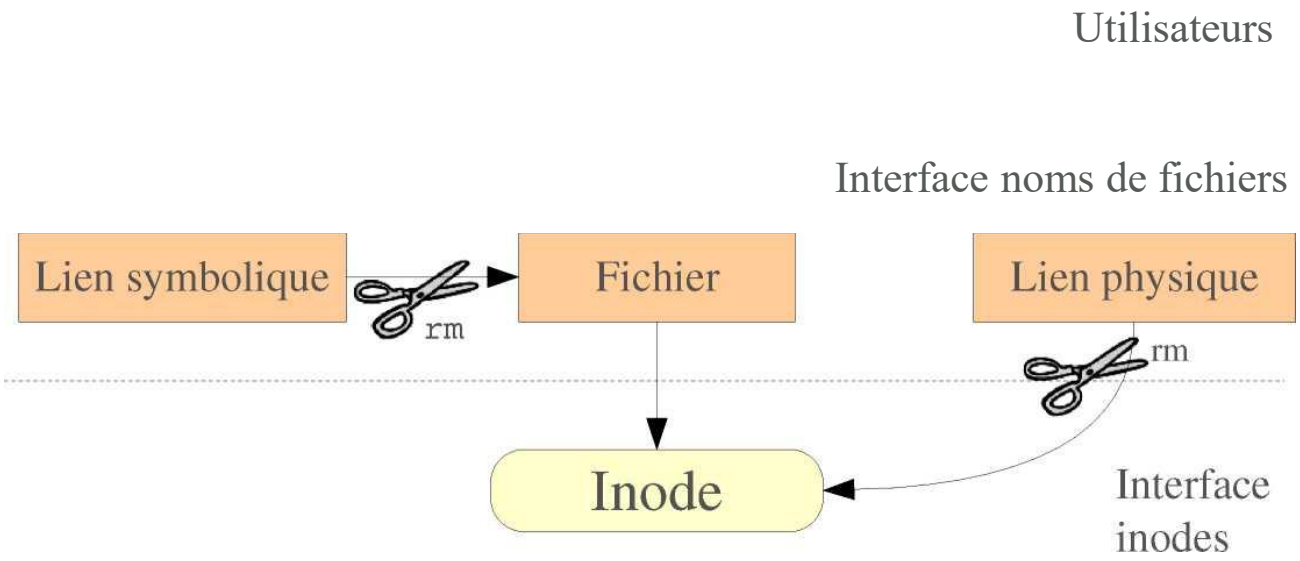
Bien qu'ils économisent toujours de la place, les liens physiques sont indiscernables des fichiers d'origine.

* **Si vous supprimez le fichier d'origine, cela n'affecte pas le contenu du lien physique.**

Le contenu est supprimé quand il n'y a plus aucun fichier (lien physique) qui y fait référence.

Noms de fichiers et inodes

Pour mieux comprendre les liens symboliques et physiques



Exercices

0. Créez un fichier de texte `linux/lien` contenant quelques lignes
1. Créez un lien symbolique de `linux/lien` vers `linux/lien. sym`
2. Créez un lien physique de `linux/lien` vers `linux/lien.phy`
3. Vérifiez que le contenu des deux liens ainsi créés est bien identique à celui de `linux/lien`
4. Renommez le fichier `linux/lien en linux/link`
5. Le contenu des deux liens est-il toujours le même ? Expliquez !
6. Créez un lien symbolique sur le fichier `/etc/fstab`
7. Vérifiez que le contenu du lien est correct
8. Recommencez en créant cette fois un lien physique
9. Recherchez tous les liens symboliques vous appartenant (i.e. se trouvant dans votre *homedir*, ou récursivement dans un de ses sous-répertoires)

Droits d'accès aux fichiers

Utiliser ls -l pour consulter les droits d'accès

3 types de droits d'accès :

* **Accès en lecture (r)**
* **Accès en écriture (w)**
* **Droits d'exécution (x)**

3 types de niveaux d'accès:

* **Utilisateur (u)**
  pour le propriétaire du fichier
* **Groupe (g)**
  tout fichier a un attribut "groupe", qui correspond à une liste d'utilisateurs
* **Autres (o)**
  pour tous les autres (propriétaire et groupe exclus)

Codage des droits d'accès

- Les droits d'accès sont codés alors sur 9 bits :

user			group			others		
r	w	x	r	w	x	r	w	x
400	200	100	40	20	10	4	2	1

- Expression des droits en code octal :

rw-r---> 640

rwxrwxr--> 774

Lorsqu'un nouveau fichier/répertoire est crée, quels seront ses droits d'accès?

Manipulation des droits d'accès par défaut

Les droits par défaut sur les ressources crées par un utilisateur peuvent être fixés par la commande umask

- `umask code_octal`

Manipulation des droits d'accès par défaut

Les droits par défaut sur les ressources crées par un utilisateur peuvent être fixés par la commande umask

- `umask code_octal`

Permission maximale pour un répertoire : 666
Permission maximale pour un fichier : 777

Manipulation des droits d'accès par défaut

Permissions Linux : umask

Permission par défaut = Permissions maximales - Umask

	Utilisateur non privilégié		Utilisateur root	
	dossier	fichier	dossier	fichier
Permissions maximales	777	666	777	666
Umask	**0002**	**0002**	**0022**	**0022**
Permissions appliquées	**775**	**664**	**755**	**644**
Notation symbolique	rwxrwxr-x	rw-rw-r--	rwxr-xr-x	rw-r--r--

Contraintes de droits d'accès

Pour les fichiers, x sans r est autorisé mais sans valeur. Vous devez pouvoir lire un fichier pour l'exécuter

Les répertoires requièrent à la fois les droits r et x : x pour entrer, r pour accéder au contenu

Vous ne pouvez pas renommer, supprimer ou copier des fichiers dans un répertoire si vous n'avez pas accès en écriture à ce répertoire

Si vous avez accès en écriture à un répertoire, vous POUVEZ supprimer un fichier même si vous ne disposez pas de droits d'écriture pour ce fichier (souvenez-vous qu'un répertoire est juste un fichier décrivant une liste de fichiers). Cela permet même de modifier un fichier (le supprimer et le recréer) même protégé en écriture

Exemples de droits d'accès

-rw-r—r—

Lisible et modifiable pour le propriétaire, seulement lisible pour les autres.

-rw-r-----

Lisible et modifiable pour le propriétaire, seulement lisible pour les utilisateurs appartenant au groupe du fichier.

drwx------

Répertoire seulement accessible par son propriétaire

-------r-x

Fichier exécutable seulement par les autres, mais ni par votre groupe ni par vous-même. Droits d'accès typique d'un piège !

chmod: modifier les permissions

chmod permissions fichiers

2 formats possibles pour permissions : format en base 8 ou format symbolique

Format en base 8 : chmod abc fichiers

avec a, b et c valant r*4+w*2+x (r, w, x égaux à 0 ou 1) Exemple : chmod 644 fichier (rw pour u, r pour g et o)

Format symbolique : plus facile à comprendre sur des exemples

chmod go+r ajouter le droit en lecture au groupe et aux autres

chmod u-w supprimer le droit d'écriture pour le propriétaire

chmod a-x *(all*) enlever les droits d'exécution à tous les utilisateurs

Autres options de chmod (1)

chmod -R a+rX linux/

Rend linux et tout ce qu'il contient accessible à tout le monde !

R : applique les changements récursivement

X : x, mais seulement pour répertoires et fichiers déjà exécutables. Très pratique pour ouvrir récursivement l'accès à des répertoires, sans donner le droit d'exécution à tous les fichiers

Autres options de chmod (2)

chmod a+t /tmp

t : (“sticky”: collant). Permission spéciale pour les répertoires, autorisant uniquement l’effacement d’un fichier par son propriétaire ou par celui du répertoire

Utile pour les répertoires accessibles en écriture par plusieurs utilisateurs, comme /tmp

Exercices

Placez-vous dans le répertoire `linux`, puis :

Créez le fichier vide `fichier`, et examinez ensuite ses permissions

Modifiez successivement les permissions du répertoire linux pour :

- supprimer tous les accès à tout le monde
- ajouter l'accès en lecture, écriture et traversée pour vous seul
- ajouter l'accès en lecture et traversée pour le groupe
- Créez le répertoire linux/tmp, puis le fichier linux/tmp/perm, et positionnez successivement les permissions adéquates pour que vous puissiez :
- lire, modifier et supprimer perm
- lire, modifier mais pas supprimer perm
- lire, supprimer mais pas modifier perm
- lire mais ni modifier et ni supprimer perm
- Créez le fichier linux/tmp/share et positionnez les permissions nécessaires pour qu'un utilisateur de votre groupe puisse lire, modifier mais pas supprimer share

III. Redirections et processus

Entrées/sorties et redirections

Les tuyaux

Contrôle de tâche et de processus

Descripteur de fichier

- **Les entrées/sorties des commandes sont repérées par des descripteurs de fichier *("file descriptor")***
- **Un descripteur de fichier est un *entier* associé à un fichier spécial**
- **On peut modifier (en les *redirigeant)* les descripteurs de fichier associés aux entrées/sorties de n'importe quelle commande**
- **Les entrées/sorties *standards* ne sont que des descripteurs de fichier prédéfinis associés par défaut aux entrées/sorties des commandes**
- **Pour chaque commande, on distingue :**
- **l'entrée standard : c'est le clavier**
- **la sortie standard : c'est l'écran**

Entrée standard (1)

De nombreuses commandes, invoquées sans arguments en entrée, lisent leurs données sur *l'entrée standard*

Le *file descriptor* décrivant l'entrée standard est 0

L'entrée standard est aussi le fichier /dev/stdin

Par défaut, l'entrée standard est le clavier

L'entrée standard peut être écrite (redirigée) dans un fichier en utilisant le symbole <

Entrée standard (2)

[hal@/home/bob] cat windows
windows linux linux <Ctrl D>

cat prend l'entrée standard comme entrée, c'est à dire tout ce qui est tapé sur le clavier ! (jusqu'au premier <Ctrl D>)

cat < participants.txt L'entrée standard de cat est prise dans le fichier indiqué. Note : ici, on écrirait plutôt cat participants.txt

ls < les_fichiers_a_lister

Ne marche pas car la commande ls ne lit pas sur sur entrée standard mais traite seulement ses arguments !

Entrée standard (3)

[hal@/home/bob] sort windows

linux <Ctrl D> linux windows

sort prend l'entrée standard comme entrée, c'est à dire tout ce qui est tapé sur le clavier ! (jusqu'au premier <Ctrl D>)

sort < participants.txt

L'entrée standard de sort est prise dans le fichier indiqué. Note : ici, on écrirait plutôt sort participants . txt

mail -s "Hello" dan < my_mail.txt

L'entrée standard de la commande mail est redirigée vers le fichier my mail. txt qui contient le corps du message

Sortie standard (1)

Toutes les commandes qui produisent du texte sur le terminal le font en écrivant sur leur *sortie standard*

Le *file descriptor* décrivant la sortie standard est 1

- **La sortie standard est aussi le fichier /dev/stdout**
- **Par défaut, la sortie standard est l'écran**
- **La sortie standard peut être écrite (redirigée) dans un fichier en utilisant le symbole 1 > ou simplement >**
- **La sortie standard peut être rajoutée à la fin d'un fichier existant par le symbole 1 >> ou simplement >>**

Sortie standard (2)

cat > un_fichiers.txt

Copie l'entrée standard (les caractères tapés au clavier) dans un_fichiers . txt ls ~ > tous mes fichiers.txt

Copie le résultat de la commande l s dans le fichier tous_mes_fichiers . txt. Le fichier est écrasé s'il existait avant ou bien créé s'il n'existait pas.

ls ~/bin >> tous mes fichiers.txt

Ajoute la sortie de la nouvelle commande ls à la suite du fichier tous_mes_fichiers.txt

cat .zlogin .zshrc .zlogout > zsh_all.txt

Concatène le contenu des trois fichiers . zlogin, . zshrc et . zlogout dans le fichier zsh_all. txt

find / -name '*.gif' -print > images.txt

Copie la sortie de la commande find dans le fichier images.txt

L'erreur standard (1)

Les messages d'erreur d'une commande sont envoyés vers

l'erreur standard **(et non pas la sortie standard)**

Le *file descriptor* décrivant l'erreur standard est 2 L'erreur standard est aussi le fichier /dev/stderr Par défaut, l'erreur standard est l'écran

L'erreur standard peut être écrite (redirigée) dans un fichier en utilisant le symbole 2>

L'erreur standard peut être rajoutée à la fin d'un fichier existant par le symbole 2>>

L'erreur standard (2)

* **Redirection de l'erreur standard vers un fichier : gcc myfile.c 2> erreur.log**

Si Redirection de la sortie et de l'erreur standard vers des fichiers distincts : gcc myfile.c > comp.log 2> erreur.log

* **Redirection de la sortie et de l'erreur standard vers le même fichier : gcc myfile.c > comp.log 2>&1**

Syntaxe alternative :

gcc myfile.c &> comp.log

SÏ On peut ignorer les erreurs en redirigeant l'erreur standard vers le pseudo-fichier /dev/null :

find / -name '*.gif > images.txt 2> /dev/null

Exercices

1. Fabriquez à l'aide d'une seule commande le fichier de texte linux/date.txt qui contient la date et l'heure courantes
2. Fabriquez le fichier linux/debut formé des 15 premières lignes de /etc/fstab
3. Fabriquez le fichier linux/fin formé des 15 dernières lignes de /etc/fstab
4. Fabriquez le fichier linux/extremes qui contient les lignes de linux/debut suivies des lignes de linux/fin
5. Fabriquez (en plusieurs étapes) le fichier linux/extremes .2, identique à linux/extremes, mais sans utiliser de fichiers intermédiaires
6. Fabriquez le fichier linux/mes_reps . txt qui contient la liste récursive de tous vos répertoires et sous-répertoires
7. Recherchez dans le système le chemin *absolu* du fichier fstab sans générer aucun message d'erreur
8. Fabriquez le fichier linux/network. txt qui contient exactement la liste récursive de tous les fichiers et répertoires du répertoire /etc et dont le nom contient la chaîne net

Les tuyaux / pipe (1)

Pour combiner et enchaîner les commandes entre elles

Les tuyaux Unix servent à rediriger la sortie d'une commande vers l'entrée d'une autre commande :

```
find . -name '*.gif' -print | more

ls | wc -l

cat *.log | grep -i erreur | sort t

cat /home/*/devoirs.txt | grep note | more
```

On peut combiner tuyaux et redirections :

```
cat devoirs/*/note.txt | grep note > notes.txt

find / -name '*.cpp' 2> /dev/null | more
```

Les tuyaux (2)

Les tuyaux s'utilisent principalement avec des filtres destinés à modifier la sortie d'une commande par application successive d'autres commandes

Recherche de ligne contenant un motif parmi les lignes du flux d'entrée avec les commandes grep, egrep et fgrep

Sélection de colonnes ou de champs sur les lignes du flux d'entrée avec la commande cut

Tri des lignes du flux d'entrée avec la commande sort

Remplacement de caractères sur les lignes du flux d'entrée avec la commande tr

Duplication du canal de sortie standard avec la commande tee

Les commandes tr et cut

tr chainel chaine2

Remplace les caractères de chainel par les caractères de chaine2

echo Bonjour Tout le Monde | tr 'Bol' 'b0L' Affiche b0nj0ur t0ut Le m0nde

echo Bonjour Tout le Monde | tr 'A-Z' 'a-z'

Affiche bonjour tout le monde

echo 'a demain !' | tr -s ' '

Affiche a demain !

cut [option]... [fichier]...

Retourne des *parties* des lignes du fichier fichier (ou de l'entrée standard)

```
echo '/usr/bin/X11:/usr/bin:/bin' | cut -d':' -f2
```

Retourne /usr/bin

```
ls -l | tr -s ' ' | cut -d' ' -f3,6,8
```

Affiche les fichiers du répertoire courant précédés par le propriéraire et la date de dernière modification, à raison d'un fichier par ligne

La commande tee

tee [-a] fichier

La commande tee peut être utilisée pour envoyer en même temps la sortie standard vers l'écran et vers un fichier.

make | tee compil.log

Lance la commande make et stocke sa sortie dans le fichier compil.log

make install | tee -a compil.log

Lance la commande make install et rajoute sa sortie à la fin du fichier compil.log

Exercices

1. Reprenez les exemples d'utilisation des tuyaux et expérimentez !
2. Lisez avec attention les pages de manuel des commandes tr et cut, et analysez avec soin les options -s (pour tr), -d et -f (pour cut)
3. Affichez le nombre de fichiers et/ou répertoires contenus dans votre *homedir*
4. Affichez la liste des fichiers (et seulement des fichiers) contenu dans votre *homedir*
5. Affichez le nombre total de fichiers réguliers (ni répertoire, ni lien symbolique) contenu récursivement dans votre *homedir*
6. Afficher le nombre total de fichiers du système de nom *.cpp auxquels vous avez accès (en lecture)
7. Affichez à l'aide d'une seule commande, en les numérotant de 40 à 50, les lignes 40 à 50 (incluses) du fichier /etc/fstab
8. Affichez la taille suivie du nom des fichiers de votre répertoire . kde/share/wallpapers sous forme lisible (par exemple 4.0K au lieu de 4 096), et seulement la taille et le nom, à raison d'une ligne par fichier

La notion de processus (1)

Depuis sa création, Unix prend en charge le vrai multitâche préemptif

Faculté de lancer de nombreuses tâches en parallèle, et de les interrompre même si elles ont corrompu leur propre état ou leur propre donnée

Faculté de choisir quels programmes précis vous lancez

Faculté de choisir les entrées utilisées par vos programmes, et de choisir où vont leurs sorties

La notion de processus (2)

En Linux, tout ce qui s'exécute est un processus

Un processus :

- **une commande, un groupe de commande ou un script en cours d'exécution**
- **son contexte d'exécution : entrées/sorties, variables d'environnement**

Chaque processus :

- **s'exécute de façon indépendante et en temps partagé**
- **possède ses propres entrées/sorties**
- **peut être individuellement suspendu, réactivé ou tué**

Même mode d'utilisation dans tous les shells

Très utile :

- **Pour les tâches en ligne de commande dont les résultats peuvent être examinés plus tard, en particulier celles qui prennent beaucoup de temps.**
- **Pour lancer des applications graphiques depuis la ligne de commande et les utiliser ensuite à la souris.**

Démarrer une tâche et ajouter & à la fin

* **find / -name '*.gif' > -/images.txt &**

Contrôle des tâches de fond

jobs

Fournit la liste des tâches de fond issues du même shell

[1] - Running find / -name '*.gif' > ~/images.txt &

[2] + Running make install > /dev/null &

fg

fg %n

Faire de la dernière (ou nième) tâche de fond la tâche courante

Mettre la tâche courante en arrière plan :

<Ctrl Z>

bg

kill %n

Interrompt la nième tâche

Exemples de contrôle de tâches

[hal@home/bob] jobs
[1] - Running find / -name '*.gif' > ~/images.txt &
[2] + Running make install > /dev/null &

[hal@home/bob] fg make install

[hal@home/bob] <Ctrl Z>
[2]+ Stopped make install

[hal@home/bob] bg [2]+ make install &

[hal@home/bob] kill %1
[1]+ Terminated find / -name '*.gif' > ~/images.txt &

Liste de tous les processus

- **ps -ux**

Affiche tous les processus appartenant à l'utilisateur courant.

- **ps aux (remarque: ps -edf sur systèmes System V) Affiche tous les processus existant sur le système**

```
ps -aux | grep bart | grep bash
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
bob	3039	0.0	0.2	5916	1380	pts/2	S	14:35	0:00	/bin/bash
bob	3134	0.0	0.2	5388	1380	pts/3	S	14:36	0:00	/bin/bash
bob	3190	0.0	0.2	6368	1360	pts/4	S	14:37	0:00	/bin/bash
bob	3416	0.0	0.0	0	0	pts/2	RW	15:07	0:00	[bash]

PID: (Process ID) Identifiant du processus
VSZ: (Virtual SiZe)Taille virtuelle du processus (code + données + pile)
RSS: (ReSident Size) Nombre de Ko occupés en mémoire
TTY: (TeleTYpe) Terminal
STAT: Statut: R (Runnable: exécutable), S (Sleep: endormi), W (paging: en cours de pagination), Z (Zombie)...

Arrêt de processus

`kill pid ...`

Envoie un signal d'arrêt aux processus spécifiés. Cela permet aux processus de sauvegarder leurs données et s'arrêter eux-mêmes. A utiliser en premier recours.
Exemple:

`kill 3039 3134 3190 3416 kill -9 pid ...`

Envoie un signal d'arrêt immédiat. Le système lui-même se charge d'arrêter les processus. Utile quand une tâche est vraiment bloquée (ne répond pas à un `kill` simple)

`killall [-signal] commande`

Arrête toutes les tâches exécutant commande. Exemple: `killall bash`

`kill -9 -1`

Arrête tous les processus de l'utilisateur courant (`-1` : tous les processus)

Activité en temps réel des processus

top : Affiche les processus les plus actifs, triés par utilisation du proc.

```
top - 15:44:33 up 1:11,    5 users, load average: 0.98, 0.61, 0.59
Tasks:    81 total, 5 running, 76 sleeping, 0    stopped,    0 zombie
Cpu(s): 92.7% us, 5.3%    sy, 0.0% ni,    0.0% id,    1.7% wa,    0.3% hi, 0.0%
si
Mem:    515344k total,    512384k used,    2960k    free,    20464k buffers
Swap:    1044184k total,    0k used,    1044184k    free, 277660k cached
```

PID> USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3809 jdoe	25	0	6256	3932	1312	R	93.8	0.8	0:21.49	bunzip2
2769 root	16	0	157m	80m	90m	R	2.7	16.0	5:21.01	X
3006 jdoe	15	0	30928	15m	27m	S	0.3	3.0	0:22.40	kdeinit
3008 jdoe	16	0	5624	892	4468	S	0.3	0.2	0:06.59	autorun
3034 jdoe	15	0	26764	12m	24m	S	0.3	2.5	0:12.68	kscd
3810 jdoe	16	0	2892	916	1620	R	0.3	0.2	0:00.06	top

L'ordre de tri peut être changé en tapant

M: utilisation mémoire, P: %CPU, T: temps d'exécution.

On peut arrêter une tâche en tapant k (kill) et son numéro

Commandes time, times et uptime

time commande

Exécute la commande commande et affiche le temps consomm é par cette exécution. Exemple :
$ time ls -R / &> /dev/null real 0m7.917s
user 0m1.698s
sys 0m2.294s

times

Affiche les temps cumulés pour tous les processus lancés depuis le début de la session

uptime

Affiche le temps depuis lequel le système est actif (donc le temps depuis le dernier *boot)* ainsi que les valeurs moyennes de charge du système

w Il existe également de nombreuses applications graphiques pour contrôler les paramètres systèmes (charge, processus) comme par exemple ksysguard sous KDE

Groupage de commandes

- **Séquence de commandes avec le symbole ;**

  echo "mes fichiers"; ls; echo "--------"

- **Groupage logique avec les opérateurs | | (ou) et && (et)**

  cat ~/infos 2>/dev/null || echo "erreur" N'exécute echo que si la première commande échoue

  ls ~/infos 2>/dev/null && cat ~/infos

  N'affiche le contenu d'infos que si la commande ls réussit

- **Groupage en sous-shell avec les parenthèses " (" et ") "**

  pwd; (cd ..; pwd); pwd

Affiche successivement les chemins du répertoire courant, du répertoire père, puis à nouveau du répertoire courant

Exécutions différées

sleep 60

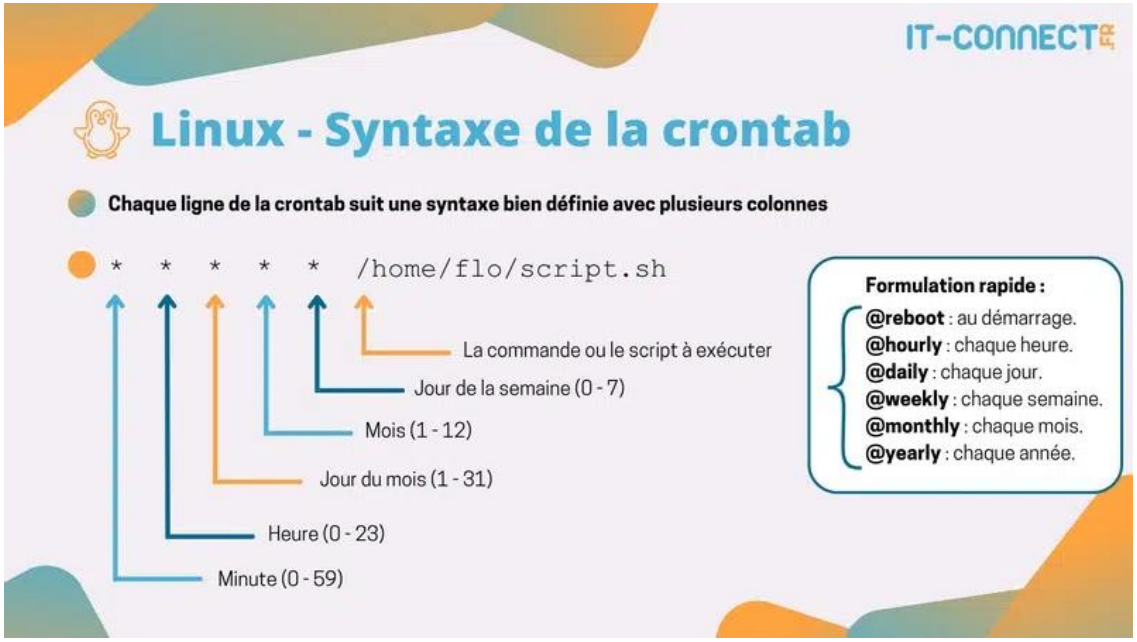
Attend 60 secondes (ne consomme pas de ressources système)

at 6:00pm today chmod o-rx /net/devoirs

Enlève aujourd'hui à 18h les droits en lecture et en accès au répertoire /net/devoirs pour *"others"*

crontab

Gère la crontab de l'utilisateur pour lancer des commandes en temps différé



 # Exécution toutes les 5 minutes
*/5 * * * * <commande>
 # Exécution chaque dimanche à 22h15
15 22 * * 0 <commande>
 # Exécution tous les jours ouvrés (lundi à vendredi) à 08h00
0 8 * * 1-5 <commande>
 # Exécution le premier jour de chaque mois à minuit
0 0 1 * * <commande>
 # Exécution à 03h00 du matin un jour sur deux
0 3 */2 * * <commande>
 # Exécution du 1er au 10ème jour de chaque mois, à 03h00 du matin
0 3 1-10 * * <commande>

*** : chaque valeur possible.**

, : plusieurs valeurs possibles (par exemple : 1,15 signifie le 1er et le 15e).

- : intervalle (par exemple : 1-5 signifie du 1er au 5e).

/ : incrément (par exemple : */10 toutes les 10 unités, soit toutes les 10 minutes, si c'est inscrit dans la colonne des minutes).

Il y a également des entrées spéciales qui sont comme des alias pour faciliter l'écriture de certaines fréquences :
@reboot : au démarrage.

@hourly : chaque heure.

@daily : chaque jour.

@weekly : chaque semaine.

@monthly : chaque mois.

@yearly : chaque année.

Exercices (1)

1. Exécutez la commande xclock -update 1. Pouvez-vous exécuter une nouvelle commande dans le terminal dans lequel vous venez de lancer ce processus ? pourquoi ?
2. Suspendez l'exécution de ce processus sans le tuer. Exécutez la commande jobs. Qu'indique cette commande ? Reprenez l'exécution du processus suspendu en arrière plan
3. Exécutez la commande xclock -digital -update 1 dans le même terminal. Suspendez l'exécution de ce processus sans le tuer. Exécutez la commande jobs. Qu'indique cette commande ? Reprenez l'exécution de ce dernier processus suspendu en arrière plan
4. Affichez le PID des deux processus que vous venez de lancer
5. Faites passer le premier processus (issu de xclock -update 1) en avant plan et arrêter ce processus. Exécutez la commande jobs. Qu'indique cette commande ?
6. Exécutez la commande kill -9 %2. Que se passe-t-il ? Exécutez la commande jobs. Qu'indique cette commande ?
7. Afficher le PID d'une commande par son nom, de manière que seul le PID et le nom de la commande exécutée s'affiche sur la ligne (ou sur les lignes si la même commande est lancée plusieurs fois)

Exercices (2)

1. Trouvez deux façons de lancer la commande xlogo dans soixante secondes à partir de maintenant (utilisez la commande xlogo avec l'option -d et tapez : xlogo -d 0.0)

2. A l'aide de la commande `crontab`, afficher un nouveau logo *Xwindow* toute les minutes

3. Exécutez la commande top et regardez en direct les processus qui s'exécutent sur votre machine, ainsi que la charge de votre machine

4. Lancez `gnome-system-monitor` et regardez en direct les processus qui s'exécutent sur votre machine, ainsi que la charge de votre machine

5. Affichez le temps que mets un ls -R pour s'exécuter depuis votre *homedir*, et n'affichez rien d'autre

6. Affichez le temps que mets un ls -R pour s'exécuter depuis le répertoire racine / , et n'affichez rien d'autre

7. Affichez exactement le temps depuis lequel le système linux dans lequel vous êtes a été lancé

IV. Compléments sur le shell

Variables

Alias

Substitution de commandes

Fichiers de configuration bash

La commande for

Les commandes basename et dirname

Variables d'environnement

Les shells permettent à l'utilisateur de définir des *variables.* Celles-ci peuvent être réutilisées dans les commandes shell. Convention : les noms sont en minuscules

Vous pouvez aussi définir des *variables d'environnement*: des variables qui sont sont aussi visibles depuis les scripts ou les exécutables appelés depuis le shell.

Convention : les noms sont en majuscules

La commande env

Affiche toutes les variables d'environnement existantes ainsi que leur valeur

Variables de shell

- **Un seul type possible : la chaîne de caractères**
- **Pas besoin d'être déclarée pour recevoir une valeur par affectation**

```
une_variable=100
```

Attention : il ne faut PAS mettre d'espace entre la variable, le signe = et la valeur

- **Utilisation d'une syntaxe particulière pour accéder à la valeur**

```
echo $une_variable
```

- **Pour qu'une variable garde sa valeur initiale dans un sous-shell**

```
export une variable
```

Exemples de variables de shell

Variables de shell

projdir=/home/bob/unix/projet ls -la $projdir; cd $projdir

Variables d'environnement

cd $HOME echo $PWD echo $PRINTER

*** export JAVA SOURCE=$HOME/java/src**

Variables standards

Utilisées par de nombreuses applications

HOME

Répertoire de l'utilisateur courant.

PATH

Chemin de recherche des commandes

USER

Nom de l'utilisateur courant

HOST

Nom de la machine locale

PRINTER

Nom de l'imprimante par défaut

SHELL

Nom du shell courant

MANPATH

Chemin de recherche des pages de manuel.

EDITOR

Editeur par défaut (vi, emacs...)

TERM

Nom du terminal / mode courant

DISPLAY

Ecran sur lequel afficher les applications X (graphiques)

LD_LIBRARY_PATH

Chemin de recherche de bibliothèques partagées

Les variables de type PATH

PATH

Spécifie l'ordre de recherche des commandes pour le shell

~/bin:/usr/local/bin:/usr/bin:/bin :/sbin

LD_L I BRARY_PATH

Spécifie l'ordre de recherche pour les bibliothèques partagées (codes binaires partagés par les applications, comme la bibliothèque C) pour ld

/usr/local/lib:/usr/lib:/lib:/usr/X11R6/lib w MANPATH

Spécifie l'ordre de recherche pour les pages de manuel

/usr/local/man:/usr/share/man

Variables et substitutions

Substitution de la valeur d'une variable

echo $HOST

Substitution protégée de la valeur d'une variable

TMP=${USER}_$$ (et non $USER_$$)

Substitution du résultat d'une commande

nb_files=$(ls | wc -l) (ou 'ls | wc -l')

Autres substitutions

Dans les substitutions suivantes, si la variable var a une valeur, alors $ {var-VAL} substitue var, sinon substitue VAL $ {var=VAL} substitue var, sinon donne à var la valeur VAL et
substitue var
$ {var?VAL} substitue var, sinon affiche VAL et quitte le shell
$ {var+VAL} substitue VAL, sinon ne substitue rien

Alias

Les shells vous permettent de définir des *alias* (des raccourcis pour des commandes que vous utilisez très souvent)

Exemples

alias ls='ls -F'

Utile pour toujours lancer des commandes avec certains param êtres

alias rm='rm -i'

Utile pour faire que **rm** demande toujours une confirmation

alias cherche='find $HOME -name'

Utile pour remplacer des commandes utilis ées régulièrement

alias cia='. /home/sydney/env/cia.sh'

Utile pour initialiser rapidement un environnement (**.** est une commande shell pour exécuter le contenu d'un script shell)

En **bash** les alias sont définis dans le fichier **.bash aliases**

Les fichiers de bash (1)

Des scripts shell sont lus automatiquement à chaque fois qu'un shell bash est lancé ou terminé

Le script /etc/profile : non modifiable par l'utilisateur

Les scripts ~/ .bash_profile et ~/ .bash_logout : modifiables par l'utilisateur

Permet de personnaliser son environnement

Les scripts ~/.bash* ne sont pas obligatoires

Les fichiers de bash (2)

/etc/profile et ~/.bash_profile

Scripts shell lus en séquence à chaque fois qu'un shell bash est lancé lors d'un login

~/.bashrc

Scripts shell lus si le shell lancé est interactif et non pas un shell de login. En général, ce fichier est également intégré dans le fichier ~/ . bash_profile

~/.bash_logout

Scripts shell lus à la sortie d'un shell de login (après la commande exit)

Les fichiers de bash (3)

/etc/profile et ~/.bash_profile Contient les initialisations des variables d'environnement propre à un shell de login (umask, prompt) et ensuite inclut le fichier ~/ . bashrc

~/.bashrc

Contient les initialisations des autres variables d'environnement comme le PATH, mais aussi des fonctions ou des alias

~/.bash_logout

Contient des commandes spécifiques à la sortie d'un shell login (nettoyage)

La commande for (1)

Exécute les commandes commandes en donnant successivement à la variable var les valeurs
val_1,..., val_2, val_n

```
for var in val_1 val_2 ... val_n do
commandes
Done
```

val_i peut être une chaîne de caractères quelconque incluant des caractères génériques (wildcards)

```
for file in *; do ... done
```

En l'absence de val_i , la variable var prend comme valeurs successives les valeurs des **paramètres** du script :

```
for arg
do echo $arg; done
```

Pour boucler sur une séquence:

```
for in in {1..10};do
echo $i
done
```

```
for (( COUNTER=0; COUNTER<=10; COUNTER+=2 )); do
   echo $COUNTER
done
```

La commande for (2)

- Pour afficher les fichiers exécutables du répertoire courant :

```
for file in *
do [ -f $file -a -x $file ] && echo $file; done
```

- Pour compiler des fichiers sources C du répertoire courant :

```
for file in progl.c prog2.c prog3.c do gcc -c $file; done
```

- Pour imprimer tous les fichiers sources C du répertoire courant :

```
for file in *.c
do lpr $file; done
```

- Pour tuer tous les processus de l'utilisateur de nom "emacs" :

```
for proc in $(ps x | grep emacs | cut -d'    ' -f2)
do kill $proc; done
```

La commande seq

- Affiche une séquence de nombres

seq i j
Si i est plus petit ou égal à j, affiche les entiers i, i+1,...,j, sinon ne fait rien

seq -s char i j

Remplace le séparateur (un '\n' par défaut) par le caractère char seq i k j
Si i est plus petit ou égal à j, affiche les entiers i, i+k,..., j, pour tout les k tels que i+k ne dépasse pas j, sinon ne fait rien

- Utilisé la plupart du temps avec l'itération for

```
for i in $(seq 1 $max); do mkdir "tmp$i"
done
```

Manipulation des chemins

basename fichier [suffix] Retourne le nom de base d'un fichier :

- basename xxx/yyy/zzz Retourne zzz
- basename xxx/yyy/zzz.c Retourne zzz.c
- basename xxx/yyy/zzz.c '.c' Retourne zzz

dirname fichier

Retourne le *préfixe* répertoire d'un fichier :

dirname xxx/yyy/zzz

Retourne xxx/yyy

La chaîne de caractères fichier n'a pas besoin d'être le chemin d'un vrai fichier !

Exercices IV.1

1. Créez le répertoire linux/java, et sous ce répertoire, créez automatiquement plusieurs fichiers (vides) dont les noms sont de la formes tpX.exoY. java avec X prenant les valeurs de 1 à 10 et Y prenant les valeurs de 1 à 5 (soit 50 fichiers au total)

2. Fabriquez pour chaque fichier de nom tp9.exo*.java ou tp10.exo*.java une copie de nom tp9 . exo* . j ava. bak ou tp10. exo* . j ava. bak

3. Renommez chaque fichier de nom tp10. exo* . j ava. bak en tp10. exo* . txt

4. Créez automatiquement sous le répertoire linux/java 10 répertoires tp1, tp2 , tp10

5. Déplacez tout fichier de nom tpX* dans le répertoire tpX

6. Créez le répertoire linux/passwd, et sous ce répertoire, créez automatiquement autant de sous-répertoires qu'il y a de lignes dans le fichiers /etc/passwd, chaque sous-répertoire ayant pour le nom le premier champ de chaque ligne (par exemple root pour la première ligne)

7. Créez automatiquement dans chaque répertoire de nom rep du répertoire linux/passwd un fichier de nom info.txt vide

8. Ajouter à chaque fichier info.txt du répertoire linux/passwd/rep le nom en clair du l'utilisateur ou du processus correspondant

9. Renommez tous les fichiers linux/passwd/rep/info.txt en linux/passwd/info.rep et effacez tous les sous-répertoires linux/passwd/rep

V. Programmation shell

Script shell

Variables, paramètres et substitutions

Structures de contrôle Fonctions

Commandes utiles

Scripts shell

Langages du shell

- **Un langage de programmation interprété**
- **Notion de variables, de paramètres, de structures de contrôles, de fonctions,...**
- **Autant de langages que de shells (sh, bash, ksh, zsh,...)**

Script shell

Un fichier exécutable contenant des instructions d'un shell donné

Scripts shell et commandes Linux

Equivalents et traités de la même façon par le système

Contenu d'un script shell

L'en-tête

Indique le shell à utiliser pour interpréter les commandes placées dans le fichier : # ! /bin/bash (pour le Bourne Again SHell)

- Des variables
- déclaration/affectation : une_variable=123
- valeur : une_variable
- paramètres : $1, $2,..., $9
- Des appels à des commandes ou à des scripts
- Toutes les commandes Linux et tous les scripts accessibles
- Des instructions

Affectation/référence de variables, instructions conditionnelles et itératives, instructions d'entrée/sortie,...

Variables et paramètres (1)

- **Variables spéciales**
  - $0 : le nom du script (du fichier)
  - $# : le nombre de paramètres du script
  - $* : la liste des paramètres du script
  - $? : le code de retour de la dernière commande exécutée $$:

  •le numéro de processus du shell courant (PID)
- **Paramètres**

  Seulement 9 noms de variable $1, $2,..., $9

  L'instruction shift supprime le premier paramètre et décale les autres vers la gauche

  L'instruction shift n supprime les n premiers paramètres et décale les autres vers la gauche

Variables et paramètres (2)

Variables prédéfinies en bash et mise à jour automatiquement

PPID : le numéro du processus père

PWD : le nom du répertoire courant

RANDOM : un nombre entier aléatoire

SECONDS : le temps (en secondes) écoulé depuis le lancement du shell courant

! : le numéro du dernier processus lancé en arrière-plan

& _ :le dernier mot de la dernière commande exécutée

Variable d'environnement pour l'exécution de script

& BASH_ENV : le nom du fichier bash exécuté à chaque début de script

Exemple de script (1)

```
#!/bin/bash
#    début du script
echo "Hello $USER”
echo "You are currently on $HOST”
echo "The content of your homedir:”
/bin/ls
#    affectation de la variable NBFILES
NBFILES=$(ls | wc -l)
echo "Total: $NBFILES elements”
```

Exemple de script (2)

```
[hal@/home/bob] chmod a+x welcome.sh

[hal@/home/bob] ls -l welcome.sh
   -rwxr-xr-x 1 bob bob 201 2005-08-01 17:05 welcome.sh
[hal@/home/bob] ls -F
   Desktop/    Trash/    emacs/    tmp/ welcome.sh*
   Mail/    bin/    lib/    unix/

[hal@/home/bob] ./welcome.sh Hello bob
You are currently on hal The content of your homedir:
Desktop    Trash emacs tmp welcome.sh
Mail    bin    lib    unix
Total: 9 elements
```

Exécution d'un script shell

. my_script.sh ou bash my_script.sh
Exécute les commandes contenues dans le fichiers my_script. sh dans le **shell courant**

. /my_script (le fichier my_script a les droits rx)
Exécute les commandes contenues dans le fichiers my_script. sh dans un **sous-shell** du shell courant

Autres commandes qui s'exécutent dans un **sous-shell** du shell courant

Les commandes reliées par des tuyaux
cat *.log | grep -i erreur | sort

Les commandes groupées entre "(" et ")"
pwd; (cd pwd); pwd

Les structures de contrôles (voir plus loin)

if, case, for, while et until

Sortie d'un script shell

exit n

Termine et sort du script shell courant avec le code de retour n exit

Termine et sort du script shell courant. Le code de retour est celui de la dernière commande exécutée dans ce script

Exemple

```
if [ $# -ne 2 ]; then
echo "wrong number of arguments: $#" exit 1
fi
```

Valeur du code de retour
0 signifie que le script s'est exécuté correctement
Différent de 0 signifie qu'une erreur est survenue

Exercices V.1

1. Ecrivez la commande **args** qui prend un nombre quelconque d'arguments eux-même quelconques et qui, lorsqu'on l'exécute, affiche son nom, le nombre de ses arguments et la liste de ses arguments :

   args fichier $USER 777 tutu

   args est appelé avec 4 argument(s) : fichier zorro 777 tutu

2. Modifiez *éventuellement* la commande **args** pour qu'elle fonctionne sans aucune modification, même après qu'on l'a renommée (par exemple, par **mv args script**). Autrement dit, la commande doit toujours afficher son nom !

3. Ecrivez la commande **words** qui affiche chaque mot du fichier (de texte) qu'on lui passe en argument, à raison d'un mot par ligne, chaque mot étant précédé de son nombre d'occurences. Les mots doivent apparaître triés par ordre alphabétique (considérez les commandes **tr**, **sort** et **uniq**).

4. Modifiez la commande **words** pour qu'elle prenne plusieurs fichiers (de texte) en paramètres.

Variables de shell

- **Un seul type possible : la chaîne de caractères**
- **Pas besoin d'être déclarée pour recevoir une valeur par affectation**

  une_variable=100

  Attention : il ne faut PAS mettre d'espace entre la variable, le signe = et la valeur
- **Utilisation d'une syntaxe particulière pour accéder à la valeur**

  echo $une_variable
- **Pour qu'une variable garde sa valeur initiale dans un sous-shell**

  export une variable

Exemples de variables de shell

Variables de shell

projdir=/home/bob/unix/projet

ls -la $projdir; cd $projdir

Variables d'environnement

cd $HOME

echo $PWD

echo $PRINTER

*** export JAVA SOURCE=$HOME/java/src**

Variables et substitutions

- **Substitution de la valeur d'une variable**

  echo $HOST

- **Substitution protégée de la valeur d'une variable**

  TMP=${USER}_$$ (et non $USER_$$)

- **Substitution du résultat d'une commande**

  nb_files=$(ls | wc -l) (ou 'ls | wc -l')

Autres substitutions

Dans les substitutions suivantes, si la variable var a une valeur, alors ${var-VAL} substitue var, sinon substitue VAL $ {var=VAL} substitue var, sinon donne à var la valeur VAL et substitue var ${var?VAL} substitue var, sinon affiche VAL et quitte le shell ${var+VAL} substitue VAL, sinon ne substitue rien

Variables et lecture de valeurs

La commande read permet de lire des chaînes de caractères sur l'entrée standard et de les affecter à des variables

read line

Lit une ligne (une suite de caractères terminée par un *newline)* sur l'entrée standard et l'affecte à la variable line

read var_1 var_2 ... var_n

Lit une ligne sur l'entrée standard, la découpe en *mots*, et affecte chaque mot aux variables var_1, var_2, var_n

S'il y a moins de mots que de variables, les variables de traîne sont initialisées avec la chaîne vide ""

S'il y a plus de mots que de variables, la dernière variable reçoit comme valeur la chaîne formée des mots de traîne

Variables et affectations

La commande set permet d'affecter des chaînes de caractères aux variables spéciales d'un script shell (paramètres) $1, $2, ...

set — val_1 val_2 ... val_n

Affecte les valeurs val_1, val_2, val_n aux variables $1, $2, $n

Mais aussi

set (sans paramètre)

Affiche toutes les variables positionnées avec leur valeur **-v**

set -x

Affiche les commandes avant qu'elles soient exécutées

set -a [+a]

Exporte toutes les variables affectées entre les instructions

set -a et **set +a** (seulement dans un script)

Exercices V.2

1. Écrivez une première version de la commande welcomel qui ne prend aucun argument, et qui,, lorsqu'elle est exécutée par l'utilisateur zorro sur la machine de nom hal, affiche les informations suivantes :

```
welcomel
Hello zorro, you're logged on hal and current date and time are: Fri Oct 6 17:56:03 CEST 2006
```

2. Écrivez une première version de la commande nospace1 qui prend en argument un fichier ou un répertoire dont le nom contient des *espaces,* et qui renomme son argument en remplaçant les *espaces* par des _ (soulignés). Vous devez utiliser la commande tr pour réaliser la commande nospace1. Exemple :

```
ls Bon*
Bonjour tout le monde
nospace1 Bonjour\ tout\ le\ monde
$ ls Bon*
Bonjour_tout_le_monde
```

Structures de contrôle

Identiques à un langage de programmation (comme C ou Java)

- Instructions conditionnelles
- Les formes && et | |

Déjà vues !

- L'instruction if-then-else-fi Sélection à une, deux ou plusieurs alternatives
- L'instruction case-esac Aiguillages à valeurs multiples
- Instructions itératives (boucles)
- la boucle for-done Itération bornée
- les boucles while-done et until-done Itérations non bornées

Groupage de commandes

- **Séquence de commandes avec le symbole ;**

echo "mes fichiers"; ls; echo "--------"

- **Groupage logique avec les opérateurs | | (ou) et && (et)**
- **cat ~/infos 2>/dev/null || echo "erreur" N'exécute echo que si la première commande échoue**
- **ls ~/infos 2>/dev/null && cat ~/infos N'affiche le contenu d'infos que si la commande ls réussit**
- **Groupage en sous-shell avec les parenthèses " (" et ") " pwd; (cd ..; pwd); pwd**

Affiche successivement les chemins du répertoire courant, du répertoire père, puis à nouveau du répertoire courant

La commande if (1)

Sélection à une alternative

```
if test_0
   then commandes_1
fi
```

Sélection à deux alternatives

```
if test_0
   then commandes_1 else
   commandes_2
fi
```

Sélection à plusieurs alternatives

```
if test_0
then commandes_0 elif test_1 then
commandes_1 • • • •
then commandes_n-1 else commandes_n
fi
```

Le code de retour de `test_0` est interprété comme un booléen :

- **code de retour 0 : le booléen est VRAI**
- **code de retour différent de 0 : le booléen est FAUX**

La commande if (2)

- **Comparaison de fichiers**

```
if cmp $1 $2; then
echo "les fichiers $1 et $2 sont identiques" else
echo "les fichiers $1 et $2 sont differents"
```

- **Une version protégée de cat**

```
if ls $1 &> /dev/null; then
cat $1
else
echo "le fichier $1 est introuvable"
Fi
```

- **Version équivalente avec les constructeurs && et || ls $1 &> /dev/null && cat $1 | | echo "..."**

- **La forme if s'utilise surtout avec la forme test**

La commande test (1)

Une commande spéciale pour effectuer des tests, très utilisée avec les commandes if **,** while **et** until

- Syntaxe standard

  test expression

- Syntaxe alternative

  [expression]

  Attention aux espaces entre ” [”, ”] ” et expression

- Produit une valeur de retour (un entier) égale à 0 si le test est VRAI, à 1 sinon
- expression est un **prédicat** spécifique qui ne fonctionne qu’avec la construction test

La commande test (2)

Prédicats sur les chaînes de caractères

c1 = c2 (resp. ! =)	VRAI si les chaînes sont égales (resp. différentes)
-z chaine (resp. -n)	VRAI si la chaîne est vide (resp. non vide)

Prédicats sur les entiers

Ici n1 et n2 sont des chaînes de caractères représentant des entiers

n1 -eq n2 (resp. -ne)	VRAI si les entiers sont égaux (resp. différents)
n1 -gt n2 (resp. -lt)	VRAI si n1 est strictement supérieur (resp. inférieur) à n2

n1 -ge n2 (resp. -le) VRAI si n1 est supérieur (resp. inférieur) ou égal à n2

La commande test (3)

Prédicats sur les fichiers et répertoires

Pour que les tests suivants soient VRAIS, il faut avant tout que fichier existe sur le disque

-r	fichier	-w	VRAI si fichier est lisible (droit r) VRAI si
fichier		-x	fichier est modifiable (droit w) VRAI si
fichier		-f	fichier possède le droit x VRAI si fichier
fichier		-d	n'est pas un répertoire VRAI si fichier est
fichier		-s	un répertoire VRAI si fichier a une taille
fichier			non nulle

La commande test (4)

On peut combiner les prédicats à l'aide de trois opérateurs et de parenthèses

! expr	VRAI si **expr** est FAUX
exprl -a expr2	VRAI si **exprl et expr2** sont VRAIs
exprl -o expr2	VRAI si **exprl ou expr2** est VRAI
(expr)	Attention : les parenthèses ont une signification pour le shell, il faut donc les faire précéder du caractères "\"

Exemple

```
if [ $x -gt 0 -a \( $y -ge 1 -o $y -lt $x \)    ]
   then
```

Exercices V.3

1. Écrivez le script welcome2, deuxième version du script welcomel (exercice II.1.1), qui affiche le message de bienvenue adéquat en fonction de l'heure de la journée. Le message doit être Good morning entre 0 heure et 12 heures, Good afternoon entre 13 heures et 18 heures, et Good evening entre 19 heures et 23 heures :

   welcome2

   Good morning zorro, you're logged on hal

2. Écrivez le script info1 qui affiche les informations de l'unique fichier ou répertoire qui est son argument :

info tutu	info lien.sym
tutu n'existe pas	lien.sym est un lien symbolique
info /etc	info .bashrc
/etc est un répertoire	.bashrc est un fichier (rw)

Si le script info1 est appelé avec zéro ou plus d'un argument, il doit produire un message d'erreur

La commande case (1)

Sélectionne val_i, le premier choix parmi val_1,..., val_2, val_n, qui correspond à la valeur de expr, et exécute ensuite commandes_i

```
case expr in
val_1 ) commandes_1 ; ; val 2) commandes 2 ; ;


    val_n) commandes_n; ; esac
```

expr peut être une chaîne de caractères quelconque

val_i peut être une chaîne de caractères quelconque incluant des caractères génériques (wildcards)

commandes_i est une suite de zéro, une ou plusieurs commandes séparées par des "; "

La commande case (2)

Sélections simples sur une chaîne de caractères

```
case $lang in
french) echo "Bonjour";;
english) echo "Hello";;
Spanish) echo "Buenos Dias";;
esac
```

Sélections avec motifs et expressions régulières

```
case $arg in
-i | -r) echo "$arg is an option";;
-*) echo "$arg is a bad option"; exit 1;;
[0-9]*) echo "$arg is an integer";;
*.c) echo "$arg is a C file"; gcc -c $arg;;
*) echo "default value";;
esac
```

Exercices

1. Écrivez la commande **vizl** qui prend un fichier en argument et le visualise. Le programme de visualisation sera choisi en fonction de l'extension du fichier. La commande doit traîter les extensions **.ps** (fichier PostScript) et **.pdf** (fichier PDF). Pour visualiser les fichiers PostScript, vous utiliserez la commande **kghostview** et pour les fichiers PDF la commande **kpdf**.

2. Écrivez la commande **viz2**, deuxième version de la commande **viz** précédente, qui maintenant peut visualiser également des fichiers d'extension **. txt** (fichier de texte). Un fichier de texte sera affiché par la commande **less** dans un **xterm** (considérez l'option **-e**). Prévoyez un message et une confirmation de fin de visualisation.

3. Écrivez la commande **viz3**, troisième version de la commande **viz** , qui maintenant peut visualiser des fichiers d'extension ext, mais aussi des fichiers d'extension ext**.gz** (fichiers compressés avec la commande **gzip**) où *ext* est l'une des trois extensions précédentes. Pour les fichiers compressés, vous devez décompresser les fichiers sans toutefois modifier les fichiers arguments !

Exercices

1. Écrivez la commande info2, deuxième version de la commande infol (exercice III. 1.2), qui maintenant prend un nombre quelconque d'arguments (au moins un) et affiche les informations pour chacun d'entre eux.
2. Écrivez la commande nospace2, deuxième version de la commande nospacel (exercice II.1.2), qui prend maintenant en paramètres des fichiers ou des répertoires (au moins un) et qui, seulement quand le fichier ou le répertoire existe et quand son nom contient des *espaces,* le renomme en remplaçant ces *espaces* par des *soulignés*. Dans tous les cas, la commande ne provoque pas d'erreur.
3. Écrivez la commande sizel qui prend en arguments des fichiers et qui, pour chacun , affiche sa taille en (kilo-)octets suivie de son nom. La commande affiche ??? à la place de la taille si le fichier n'est pas un fichier régulier.
4. Écrivez la commande afind qui prend en argument un répertoire et un nom, et qui cherche dans quelles bibliothèques C appartenants au répertoire le nom est défini. Une bibliothèque C est un fichier d'extension .a et on trouve en général les bibliothèques standards sous /usr/lib. Utilisez la commande nm pour connaître les noms externes définis dans une bibliothèque.

La commande while

Exécute les commandes commandes tant que le test un_test est VRAI

```
while un_test do
commandes
done
```

un_test peut être une commande **quelconque** mais le plus souvent c'est un appel à la commande test

(ou [ )

```
while [ $# -gt 0 ]; do echo $1; shift
done
```

La commande un_test peut être remplacée par : ou true qui retournent toujours le code 0 (VRAI)

La commande until

Exécute les commandes commandes tant que le test un_test est FAUX

```
until un_test do
commandes
done
```

Équivalente à une commande while en prenant la négation du test

```
while [ $# -gt 0 ]; do until [ $# -eq 0 ]; do echo $1; shift    echo $1; shift
done    done
```

La commande un_test peut être remplacée par false qui retourne toujours le code 1 (FAUX)

La commande break

La commande break permet de sortir d'une boucle sans terminer l'itération en cours et sans passer par le test

```
while true do
echo -n "list the current dir? (y/n/q) ” read yn
    case $yn in
    [yY] ) ls -l . ; break;;
    [nN] ) echo "skipping" ; break;; q ) exit ; ;
    * ) echo "$yn: unknown response";;
    esac
done
```

La commande continue

La commande continue permet de passer directement à l'itération suivante sans nécessairement terminer l'itération en cours

```
for f in *; do
        [ -f [  $f ]    continue
        -r      $f ]    continue
        echo    "printing file $f:"
        case    $f in
                *.ps ) lpr $f;;
                *.txt) a2ps $f;;
                *) echo "don't know how to print $f"

        esac
        echo "done"
done
```

Structures de contrôle et redirections

L'évaluation d'une structure if, case, for, while ou until s'effectue dans un sous-shell

On peut rediriger l'entrée et/ou la sortie d'un tel sous-shell

```
for file in *.c; do gcc -c $file
done 2> compil.error
```

On peut lier un tel sous-shell avec un tuyau

```
ls ~dan/images/*.gif | while read file; do [ -f $(basename $file) ]    || cp $file .;
done
```

On peut exécuter un tel sous-shell en arrière plan

```
for file in *.c; do gcc -c $file; done &
```

Exercices V.6

1. Écrivez la commande mycal qui améliore la commande /usr/bin/cal en autorisant alternativement de qualifier les mois à l'aide de l'un des mots jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov et dec, et ce indépendamment de la casse (par exemple, jan, Jan, JAN, JaN, jAn, etc. sont équivalents). La commande mycal doit se comporter comme /usr/bin/cal et en particulier admettre les mêmes options. Quand l'unique argument est le mois en lettre, l'année est l'année en cours.
2. Écrivez la commande size2, deuxième version de la commande size1, qui affiche maintenant ses résultats en colonne (considérez la commande column).
3. Écrivez la commande chlnk qui prend en paramètre un ou plusieurs répertoires, et qui affiche tous les liens symboliques invalides appartenant récursivement à ces répertoires. Un lien symbolique est invalide s'il ne pointe pas vers un fichier ou un répertoire existant.
4. Écrivez la commande chext1 qui prend en paramètre deux extensions ext1 et ext2 ainsi qu'un répertoire, et qui renomme tous les fichiers du répertoire de nom xxx. ext 1 en xxx.ext2. Si le nom du répertoire n'est pas fourni, la commande s'applique au répertoire courant.

Les fonctions (1)

Syntaxe :

Pour définir la fonction de nom fonction :

```
fonction () {

commandes

}
```

Même mécanisme de passage de paramètre qu'un script shell

Peut contenir des appels à des commandes quelconques, à d'autre fonctions ou à elle-même (fonction récursive)

L'instruction return n permet de sortir d'une fonction avec le code de retour n

Attention : l'évaluation de la commande exit depuis l'intérieur d'une fonction termine le **script shell englobant**

Les fonctions (2)

Une fonction est interprétée dans le shell courant (pas de sous-shell)

Une sorte d'alias avec paramètres Peut être définie dans le . bashrc pour être conservée *

Une fonction peut être exécutée dans le shell interactif courant

Modification **possible** des variables d'environnement

Une fonction peut être exécutée dans le sous-shell interprétant un script

Attention : l'évaluation de la commande exit depuis l'intérieur d'une fonction termine le **script shell englobant**

Une fonction peut être récursive

Les **fonctions (3)**

Une fonction pour se déplacer facilement dans un répertoire

```
cdd() {

    for dir in $(find $HOME -name $1 -type d); do

    echo -n "$dir ? " read yes

    if [ -n "$yes" ]; then cd $dir; return 0 fi

    done
    }
```

Les fonctions (4)

Une fonction pour les erreurs

```
usage() {
   echo "usage: $(basename $0) FILE” 2>/dev/stderr exit 1


}
```

Une fonction pour confirmer une saisie avec la question en paramètre

```
confirm () {

echo -n $1

while read ANSWER; do
case $ANSWER in y
Y) return 0; ; n
N) return 1; ;
*) echo -n "please answer y or n: ";;
esac
done
}
```

Exercices

1. Écrivez la commande movel, clone de la commande mv, mais qui n'admet aucune option et exactement deux arguments. Bien sûr, il est interdit d'utiliser la commande mv !
2. Écrivez la commande move2, deuxième version de la commande movel précédente, qui est toujours un clone de mv sans option mais qui cette fois peut prendre plus de deux arguments.
3. Écrivez la commande move3, troisième version de la commande movel, commande identique à move2 mais qui admet éventuellement l'option -i.
4. Écrivez la commande copyl, clone de la commande cp, mais qui n'admet aucune option et exactement deux arguments (un fichier régulier suivi d'un fichier ou un répertoire).
5. Écrivez la commande copy2, deuxième version de la commande copyl précédente, qui est toujours un clone de cp sans option mais qui cette fois peut prendre plus de deux arguments.
6. Écrivez la commande copy3, troisième version de la commande copyl, commande identique à copy2 mais qui admet éventuellement les options -i et -r.

La commande expr (1)

Une commande pour effectuer des calculs arithmétiques sur les entiers

Utilisation

```
expr 2 + 3
incr=$(expr $incr + 1)
```

Erreurs fréquentes

expr 2 +3 => expr 2 + 3 : +3 est vu comme un unique argument

expr 2 * 3 => expr 2 * 3 : * est un méta-caractère du shell

expr 2 * (3 + 5) : les parenthèses ' (' et ') ' sont interprétées par le shell =>

```
expr 2 \* \( 3 + 5 \)
```

La commande expr (2)

Une commande pour effectuer des tests ou des calculs sur les chaînes de caractères

expr match chaine modèle(ou expr chaine : modèle) Teste si chaine correspond à modèle :

expr match linux ' l i * '

expr substr chaine i n

Affiche la sous-chaîne de chaine de longueur n et commençant au caractère à l'indice i (le premier caractère est à l'indice 1)

expr index chaine caractère

Affiche l'indice de la première occurence de caractère dans chaine, ou 0 si caractère n'est pas présent dans chaine

expr length chaine

Affiche la longueur de chaine

Arithmétique entière

La notation ((. . .)) permet d'effectuer simplement des tests ou des calculs arithmétiques sur les entiers

Test arithmétique :

```
if (( x > 1 )) ; then
    . . .
```

Le $ devant la variable x n'est pas nécessaire et les caractères < et > perdent leur signification spéciale Calcul (et substitution) arithmétique :

```
x=10
y=20
z = $ ( ( x  +  ( 2  *  y ) / 5 ) )
```

On peut utiliser les caractères +, *, /, (,) selon leur signification arithmétique usuelle

La commande eval (1)

Une commande pour évaluer la commande qui lui est passée en paramètre

eval commande

Evalue commande comme si elle avait été directement interprétée par le shell courant :

le shell applique le découpage en sous-commandes, les substitutions de variables et de commandes et la génération de noms de fichiers à commande

le shell applique le découpage en sous-commandes, les substitutions de variables et de commandes et la génération de noms de fichiers au résultat précédent et finalement interprète le tout

Affectation avec calcul de la variable affectée

```
x=y; eval $x=123; echo $y
```

(y a la valeur ”123”)

La commande eval (2)

Une fonction qui modifie une variable

```
function chvar() {
eval $1="$2"
}
```

Exemple : modification de la variable PATH

```
[hal@/home/bob] echo $PATH
/bin :/usr/bin:/usr/local/bin
[hal@/home/bob] chvar PATH '~/bin:$PATH'
[hal@/home/bob] echo $PATH
/home/bib/bin:/bin:/usr/bin:/usr/local/bin
```

La commande exec

Une commande à deux usages w exec commande

Le shell courant exécute la commande commande qui **devient** le shell courant

aucun intérêt dans un shell interactif (ne le testez pas)

à ne pas confondre avec l'option -exec de la commande find w exec et redirections

exec < file, exec > file, exec 2> file

Redirigent l'entrée, la sortie ou les erreurs du **shell courant** vers le fichier file

exec 3<&0

Affecte l'entrée standard au descripteur de ficher 3

La commande getopts

Une commande pour traiter plus simplement les arguments d'un script

```
while getopts f:ri opt; do case $opt in
r)    echo    "r    present";;
i)    echo    "i    present";;
f)    echo    "f    present with    argument $OPTARG";;
esac

done

shift $(expr $OPTIND - 1)

echo "command arguments:"
for arg in $@; do echo " $arg"

done
```

La commande select

Une commande pour fabriquer simplement un menu de saisie

```
select action in ajouter supprimer modifier do
echo "Vous avez choisi $action" break done
case $action in

ajouter)    ...;;
supprimer)    ...;;
modifier)    ...;;
esac
```

Exercices V.8

1. Écrivez la commande myhead, clone de la commande head, mais admettant seulement les deux options -n et -q et ne lisant que dans un ou plusieurs fichiers (et pas sur l'entrée standard). Pour simplifier l'écriture de myhead, on interdira la forme myhead -15 au profit de myhead -n 15. Enfin, lorsque l'option -n n'est pas présente, le nombre de lignes à afficher sera la valeur de la variable d'environnement MYHEAD DEFAULT si elle existe, ou bien 10 sinon.

2. Écrivez la commande myuniq, clone de la commande uniq, mais admettant uniquement les trois options -c, -d et -u et ne prennant aucun argument (l'entrée et la sortie de la commande sont l'entrée et la sortie standards).

3. Écrivez la commande chext2, deuxième version de la commande chext1 (exercice III.4.4), commande identique à chext1 mais qui admet éventuellement les options -i et -r. Si l'option -i est présente, la commande demande confirmation à l'utilisateur avant d'effectuer le changement d'extension. Si l'option -r est présente, la commande cherche les fichiers d'extension . ext 1 récursivement dans le répertoire.

4. Écrivez la commande kproc qui prend en argument des mots et qui tue interactivement tous les processus dont le nom contient un ou plusieurs de ces mots. Pour chaque processus, la commande affiche toutes les informations qui lui sont associées.

VI. Exercices de synthèse

Gestion de sauvegardes Gestion d’une poubelle Un mécanisme d’annulation

Gestion de sauvegardes

1. Le but de cet exercice est d'écrire les deux commandes backup et restore, commandes permettant de gérer simplement des sauvegardes de fichiers et de répertoires.

2. La commande backup prend comme arguments un ou plusieurs fichiers ou répertoires, et en crée des copies. Par exemple, backup xxxx fabrique la copie xxxx.bak, que xxxx soit un fichier ou un répertoire. Si xxxx est un répertoire, son contenu est copié récursivement dans xxxx.bak mais les éléments qu'il contient ont le même nom relatif, à moins qu'on utilise l'option -r. Afin d'éviter les sauvegardes de sauvegardes de sauvegardes... il est interdit de sauvegarder un fichier de sauvegarde avec la commande backup.

3. La commande restore prend comme arguments un ou plusieurs fichiers ou répertoires qui ont été sauvegardés, et les remplace par leur copie. Après restauration, les copies sont effacées. Pour faciliter l'utilisation de la commande restore, on peut l'appeler indifféremment avec le nom du fichier à restaurer ou avec le nom de la sauvegarde. Ainsi, restore xxxx et restore xxxx.bak sont équivalents. La commande restore admet également l'option -r.

4. L'extension utilisée pour les fichiers de sauvegarde est la valeur de la variable BACKUP EXTENSION ou bien back si cette variable n'est pas définie.

Gestion d'une poubelle

Le but de cet exercice est de concevoir et d'écrire un ensemble de commandes pour gérer une *poubelle*. La poubelle est un répertoire dans lequel on déplace des éléments (fichiers ou répertoires) qu'on peut éventuellement récupérer plus tard. La poubelle peut être vidée définitivement, après quoi les éléments qui s'y trouvaient ne sont plus accessibles. Les commandes à réaliser ne se limitent pas à déplacer les fichiers ou les répertoires simplement dans le répertoire poubelle. Par exemple, on peut mettre à la poubelle consécutivement deux fichiers de même nom relatif, ou plus délicat, de même nom absolu (mais néanmoins différents, car existant à des moments différents) et vouloir récupérer chacun d'eux ! L'ensemble des commandes à développer comprend au minimum les commandes :

0. trash, qui met à la poubelle un ou plusieurs fichiers ou répertoires

. intrash, qui liste de manière lisible et structurée le contenu de la poubelle

2. untrash, qui permet de récupérer simplement un fichier ou un répertoire contenu dans la poubelle (à partir des informations fournies par intrash)

3. cleantrash, qui permet de vider tout ou partie de la poubelle en fonction de certains paramètres et options (par exemple, en fonction du temps depuis lequel le fichier ou le répertoire est dans la poubelle)

Un mécanisme d'annulation (1)

Certaines commandes Linux modifient leur environnement, comme les commandes rm, cp ou mv pour ne citer que les plus courantes. Il serait utile de disposer d'un mécanisme d'*annulation* de l'exécution d'une telle commande qui permette de revenir à l'état initial (d'avant l'exécution de la commande). Le but de cet exercice est de réaliser un ensemble de commandes pour gérer l'*annulation* de certaines commandes.

Le principe d'annulation proposé repose sur une commande très particulière, la commande undo. Par exemple, si on veut pouvoir annuler l'effet de la commande mv xxx yyy , il faut que la commande mv elle-même produise avant de s'exécuter la commande annulante mv yyy xxx . La commande annulante peut être placée toujours dans le même script, la commande undo ! Ainsi, les effets de la commande mv xxx yyy peuvent être annulés par la commande mv yyy xxx , à condition que le fichier yyy n'existe pas au moment de l'exécution de la commande mv. Dans le cas contraire, le commande mv doit effectuer en premier une sauvegarde du fichier yyy, par exemple avec mv yyy /tmp/yyy , et les commandes annulantes (placées dans le script undo) deviennent mv yyy xxx suivie de mv /tmp/yyy yyy. Pour résumer, on dira que c'est la commande à (éventuellement) annuler qui, connaissant ses paramètres, est à même de produire les commandes suceptibles d'annuler ses effets !

Un mécanisme d'annulation (2)

Notez que seule la dernière commande exécutée dans le shell interactif courant peut être (éventuellement) annulée. Aussitôt une nouvelle commande évaluée, si cette dernière n'est pas une commande annulable, la commande undo ne doit rien faire si ce n'est délivrer un message d'erreur.

Vous devez développer au minimum les trois nouvelles commandes rm, cp et mv , versions annulables des commandes initiales du même nom. Idéalement, ces commandes doivent admettre les mêmes options et les mêmes paramètres que les commandes intiales correspondantes, mais vous pouvez vous limiter aux options essentielles (-i et -r par exemple).

La commande undo doit admettre au minimum les options -v et -n :

- '■V -v (verbose) indique que la commande undo affiche les opérations intermédiaires qu'elle réalise pour annuler la dernière commande tout en les effectuant
- '■V -d (dry run) affiche les opérations intermédiaires qu'elle réaliserait pour annuler la dernière commande, mais sans les réaliser ! Remarquez qu'on doit pouvoir exécuter la commande undo après qu'on ait exécuté une commande annulable suivie d'un nombre quelconque de undo -d !

🧩 Contexte

Vous développez un outil en ligne de commande nommé sys_toolkit.sh proposant plusieurs **sous-commandes** :

- inspect : inspecter un répertoire et produire des rapports.
- backup : créer une archive compressée des fichiers sélectionnés.
- cleanup : nettoyer des fichiers temporaires et journaux.
- process : analyser un fichier texte (statistiques, top mots).
- monitor : rapport succinct sur l'état du système.
- Chaque sous-commande accepte des **options** et affiche une **aide**.

```
mkdir -p lab/dirA lab/dirB
printf "alpha beta gamma\nbeta gamma delta\n" > lab/dirA/a.txt
printf "gamma delta alpha\nalpha alpha beta\n" > lab/dirB/b.txt
touch lab/dirA/tmp.log lab/dirB/tmp.tmp
ln -s lab/dirA/a.txt lab/dirB/a.link

./sys_toolkit.sh help
./sys_toolkit.sh inspect lab
./sys_toolkit.sh inspect lab --pattern "alpha" --ext ".txt"
./sys_toolkit.sh process lab/dirA/a.txt --top 3
./sys_toolkit.sh backup lab --output backup.tar.gz --ext ".txt"
tar -tzf backup.tar.gz | sort
./sys_toolkit.sh cleanup lab --dry-run
./sys_toolkit
```

Les éditeurs de texte

Vi

vi est un éditeur de texte en mode texte plein écran écrit par Bill Joy en 1976 sur une des premières versions de la distribution Unix BSD.

Vim

Vim est un éditeur de texte, c'est-à-dire un logiciel permettant la manipulation de fichiers texte. Il est directement inspiré de vi, dont il est le clone le plus populaire. Son nom signifie d'ailleurs Vi IMproved, que l'on peut traduire par « VI aMélioré »

Nano

Ce petit éditeur est un clone libre de "pico" qui était inclus dans le logiciel de gestion de courrier "Pine". Nano utilise la bibliothèque "ncurse" pour offrir une interface dite "semi-graphique" à l'intérieur d'une console. Ici pas de surenchère de fonctions, pas de raccourcis "à quatre mains", juste ce qu'il faut pour éditer un fichier, y effectuer une recherche, c'est un outils simple et efficace pour le spécialiste, mais surtout c'est peut-être le seul éditeur de texte en console totalement accessible au grand débutant.

Les éditeurs de texte (cheat Sheet)

Vi-Cheat-Sheet-by-linuxsimply.com_.pdf

vim-commands-cheat-sheet-by-pnap.pdf

Nano_Cheat-_Sheet_by_Linuxsimply.pdf

VIII. Epilogue

Ressources internet sur Linux

Ressources Linux sur internet

Une petite sélection de ressources internet diverses sur Linux

& Le site officiel de Linux *
http: //www .linux .org/

& Le site officiel GNU *
http: //www .gnu .org/

& Le site de Linux-France *
http: //www .linux-france.org/

& La page web du Linux Journal *
http://www.linuxjournal.com/

* Toutes les nouveautés sur Linux *
http://www.linuxcentral.com/

* Un site sur les distributions Linux *
http://distrowatch.com/

Un site d'informations sur Linux
http://www.toolinux.com/

Un site de forums sur Linux http: //www .linuxquestions. org/

Un site dédié aux débutants Linux
http://www.delafond.org/survielinux/

Un site généraliste sur Linux http://linuxfr.org/pub/

Le site de Linux-Azur http://www.linux-azur.org/

Le site de la conférence Solutions Linux
http://www.solutionslinux.fr/

Commandes de bases - réseau

1. Afficher la Configuration Réseau

•**`ip a`** : Affiche les interfaces réseau et leurs adresses IP.

```
ip a
ip a show <interface> # Exemple : ip a show eth0
```

•**`ip route`** : Montre la table de routage.

```
ip route show
sudo ip route add default via <gateway> dev <interface>
```

2. Tester la Connectivité

•**`ping`** : Vérifie la connectivité avec un hôte.

```
ping -c 4 www.google.com
ping -4 www.google.com # Forcer IPv4
ping -6 www.google.com # Forcer IPv6
```

•**`traceroute`** : Affiche le chemin emprunté par les paquets.

```
traceroute www.google.com
sudo traceroute -I www.google.com # Utiliser ICMP au lieu d'UDP
```

3. Analyser le Trafic Réseau

•**`tcpdump`** : Capture et analyse les paquets réseau.

```
sudo tcpdump -i eth0
sudo tcpdump port 80 # Filtrer par port
```

•**`ss`** : Affiche les connexions réseau actives (remplace `netstat`).

```
ss -tulnp # Liste des ports ouverts (TCP/UDP)
ss state established # Connexions TCP établies
```

4. Résolution DNS

•**`nslookup`** : Interroge un serveur DNS.

```
nslookup www.google.com
nslookup www.google.com <serveur_DNS>
```

•**`dig`** : Fournit des informations détaillées sur les enregistrements DNS.

```
dig www.google.com +trace # Suivre la résolution DNS complète
dig @8.8.8.8 www.google.com # Utiliser un serveur DNS spécifique
```

Commandes de bases - réseau

5. Scanner et Tester les Ports

•**nmap** : Scanne les ports et détecte les services actifs.

```
nmap -p- <adresse_IP> # Scanner tous les ports (1-65535)
nmap --top-ports=100 <adresse_IP> # Scanner les ports les plus utilisés
```

•**nc (netcat)** : Teste l'ouverture des ports ou transfère des fichiers.

```
nc -zv <h&#244;te> <port> # Exemple : nc -zv localhost 22
```

6. Mesurer la Bande Passante

•**iperf** : Évalue les performances réseau entre deux machines.

```
iperf -s # Serveur
iperf -c <adresse_IP_serveur> # Client
```

•**speedtest-cli** : Teste la vitesse Internet.

```
speedtest-cli
```

Ces commandes sont essentielles pour tout administrateur système ou utilisateur souhaitant diagnostiquer ou configurer un réseau sous Linux.

QUIZZ

https://wayground.com/admin/quiz/69372139655cb86ca2797250

https://wayground.com/admin/dashboards/lesson/6937260ba244fcf50d60d834?waitingScreen=true