# Formation Unix/Linux
# utilisation en ligne de commande

Guillaume Allègre
Guillaume.Allegre@silecs.info

Silecs

2019

# Licence Ceative Commons By - SA

- Vous êtes libre de
  - **partager** — reproduire, distribuer et communiquer l'oeuvre
  - **remixer** — adapter l'oeuvre
  - d'utiliser cette œuvre à des fins commerciales

- Selon les conditions suivantes
  - **Attribution** — Vous devez attribuer l'oeuvre de la manière indiquée par l'auteur de l'oeuvre ou le titulaire des droits (mais pas d'une manière qui suggérerait qu'ils vous soutiennent ou approuvent votre utilisation de l'oeuvre).
  - **Partage à l'identique** — Si vous modifiez, transformez ou adaptez cette oeuvre, vous n'avez le droit de distribuer votre création que sous une licence identique ou similaire à celle-ci.

`http://creativecommons.org/licenses/by-sa/3.0/deed.fr`

# Contribuer - Réutiliser

Ce document est rédigé en LaTeX+ Beamer.

Vous êtes encouragés à réutiliser, reproduire et modifier ce document, sous les conditions de la licence *Creative Commons, Attribution, Share alike 3.0* précédemment décrite.

J'accepte volontiers les remarques, corrections et contributions à ce document

Vous pouvez obtenir les sources LaTeXde ce document sur le dépôt Mercurial :
http://hg.silecs.info/hg/public/formations/linux/
où vous pouvez naviguer ou télécharger une archive.
Une version PDF est disponible sur
http://www.silecs.info/dld/lpi/

0

# Linux Professional Institute Certification

Ce document est un support de formation adapté à la préparation de la certification LPIC-1 (101 et 102).
Ce document **n'est pas** un support agréé officiellement par le LPI.

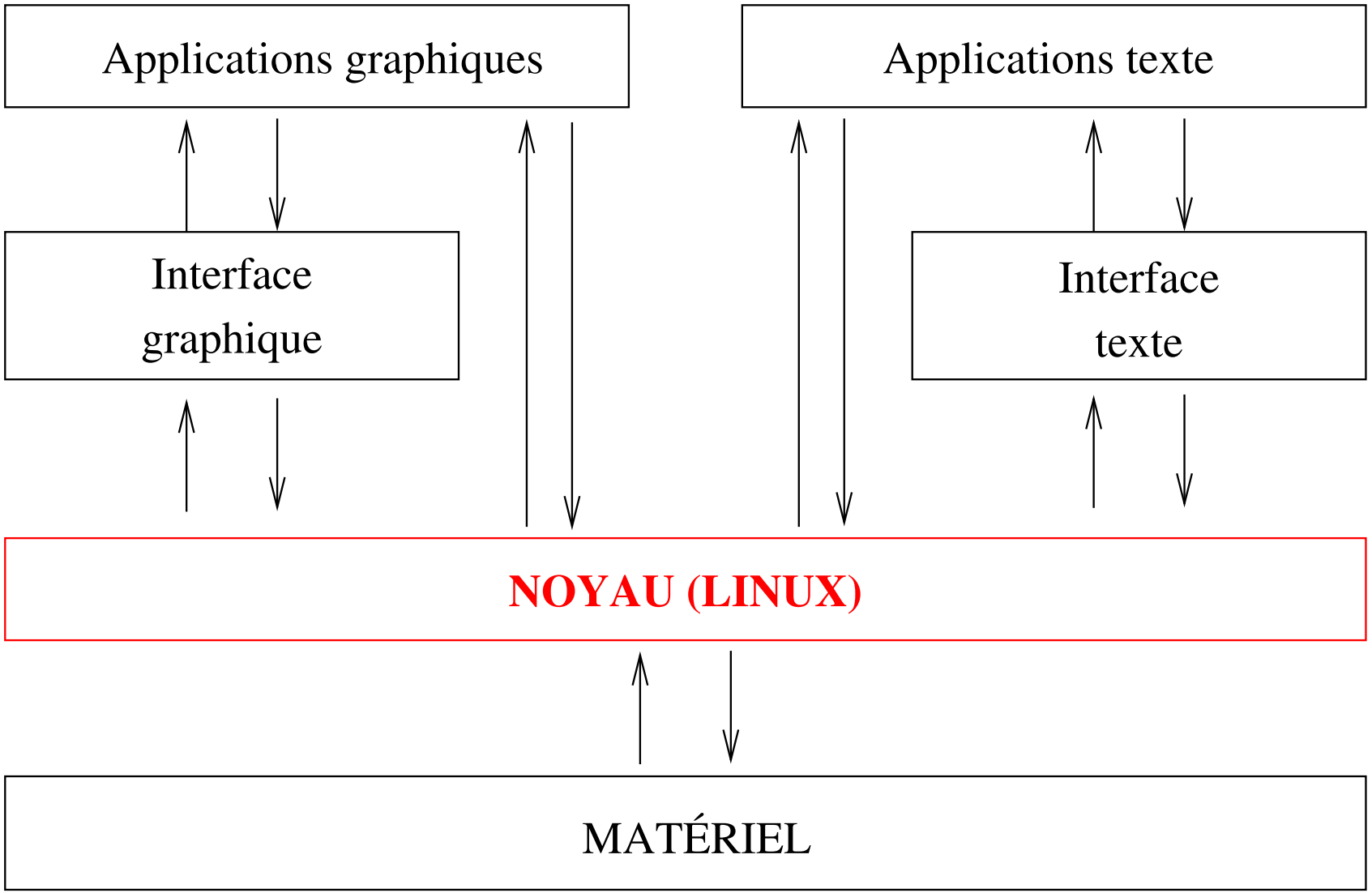
Chaque transparent directement lié au programme LPI porte la référence de l'item LPI correspondant (par exemple **105.3**) sur la ligne de titre. Les documents de référence sont *Objectifs détaillés des examens* LPIC 101 et LPIC 102, révision 4.0 (février 2015) : `http://wiki.lpi.org/wiki/LPIC-1_Objectives_V4(FR)`

L'ordre des notions abordées diffère de celui du programme LPI. Le parti-pris de ce document est de se concentrer d'abord sur la maîtrise des outils en ligne de commande (utilisateurs), puis seulement sur les outils d'administration.

L'auteur (Guillaume Allègre) est certifié LPIC-1.

# Qu'est-ce que Linux ?

# Architecture d'un système d'exploitation



# Une histoire de famille : Unix

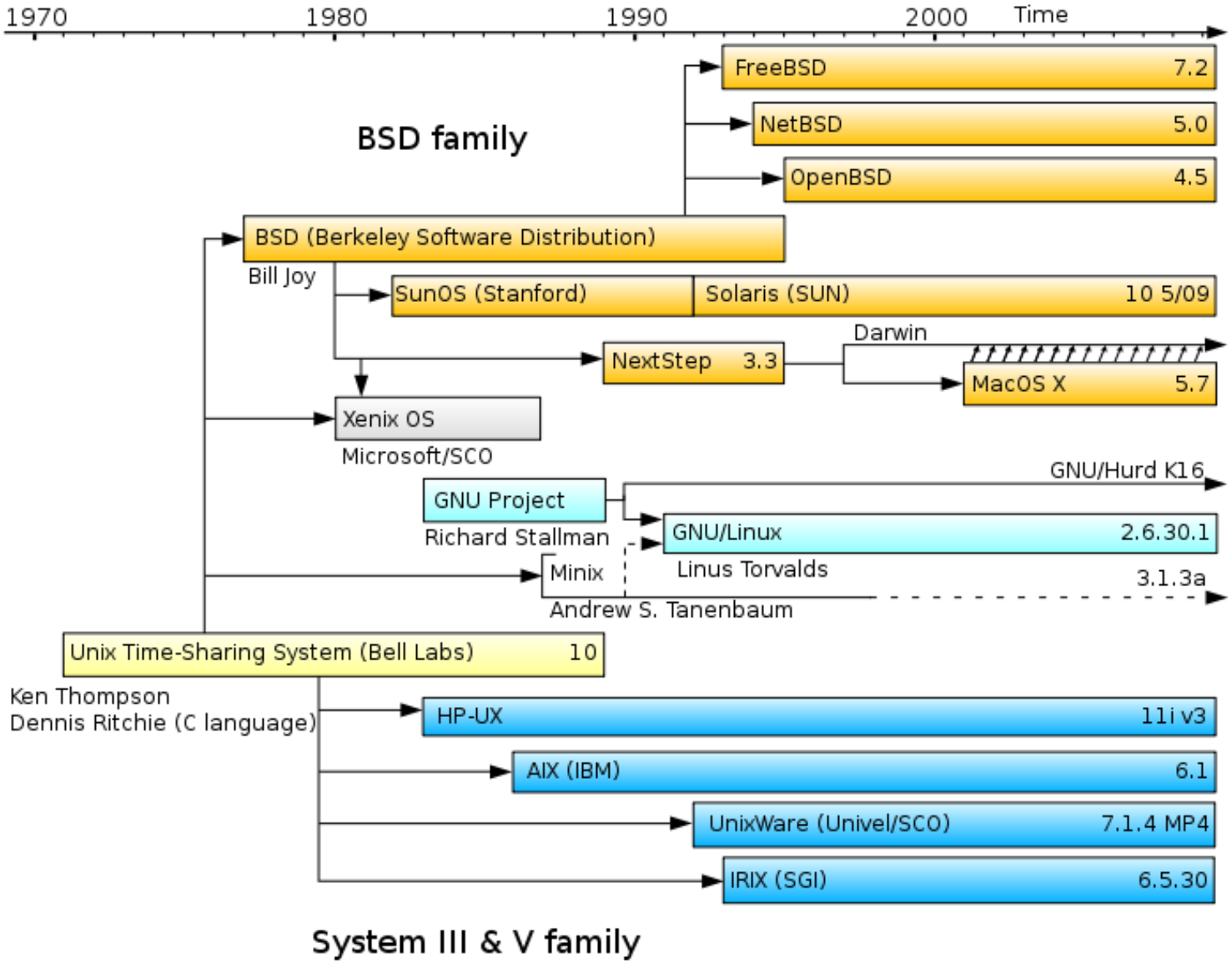
UNIX en quelques points :

1. apparu en 1969 à AT&T - Bell Labs., K. Thompson, D. Ritchie
2. beaucoup de dérivés : Solaris, AIX, BSD, OS X...
3. conçu comme un système professionnel :
   - orienté réseau,
   - multi-tâches,
   - multi-utilisateurs.
4. trois survivants propriétaires : Solaris (Sun), AIX (IBM), HP-UX

Une normalisation : POSIX (IEEE 1003) 1985-1998

1. 17 thèmes : Core, Real-time, Threads, Shell...
2. évolutions : POSIX :2001, POSIX :2004, POSIX :2008

# Une brève histoire d'Unix

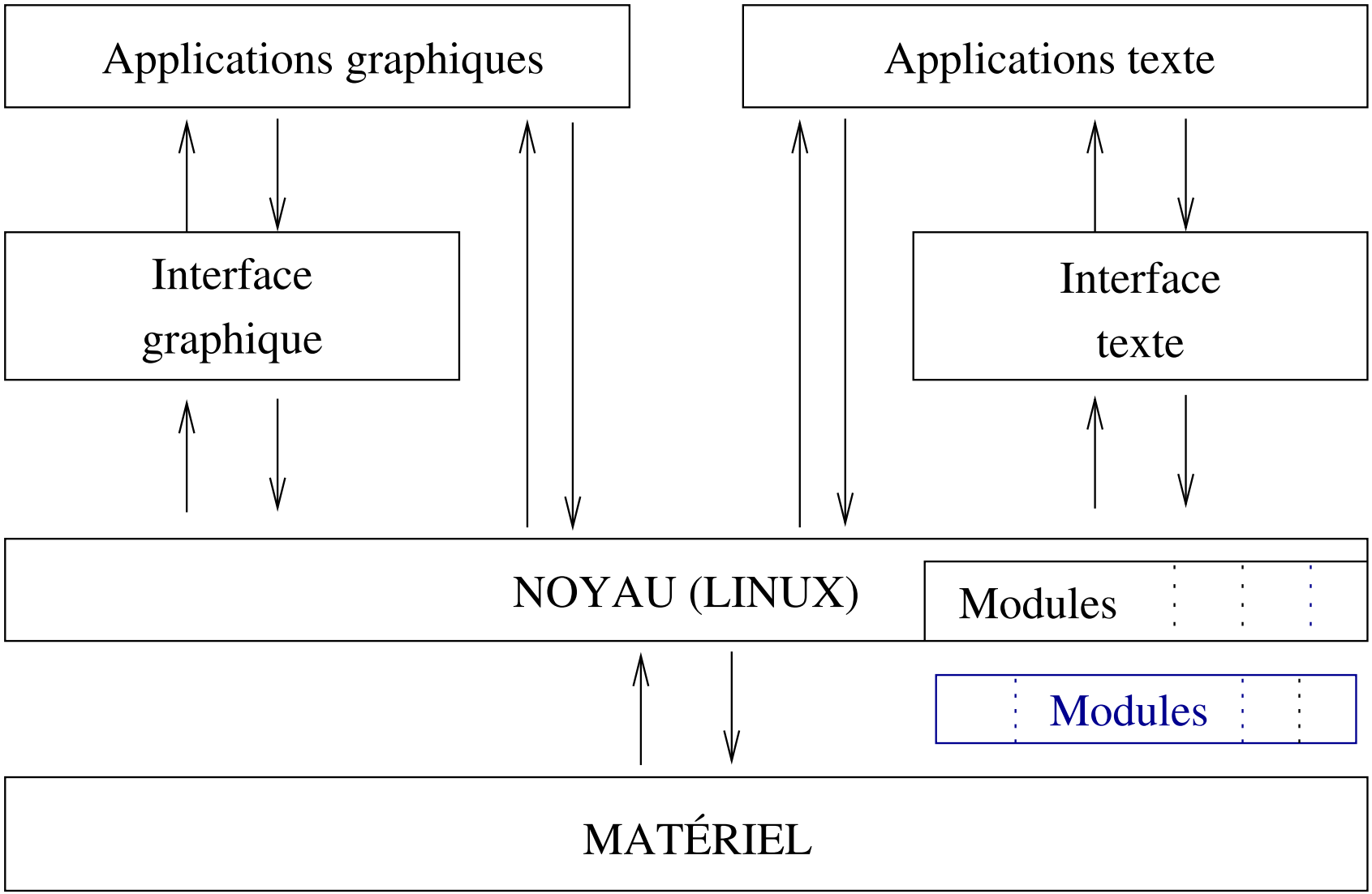



Domaine Public - Wikimedia Commons - Unix history.en.svg

# Les spécificités de Linux

- créé en 1991 par *Linus Torvalds*, étudiant finlandais.
- logiciel libre
  - inscrit dans la mouvance GNU
  - sous licence GPL depuis 1992
  - fer de lance du logiciel libre
- développement décentralisé et collaboratif
- modulaire : chargement d'extension du noyau à la demande (pilotes...)
- portable : compatible avec un très grand nombre d'architectures.

# Le système Linux



# Principales différences GNU/Linux / Windows

1. Un ensemble très modulaire vs. un bloc monolithique
2. Une seule arborescence (*tout est fichier*)
3. Fichiers de configuration et éditeurs de texte
   (pas de base de registres)
4. Importance de la ligne de commande (*une tâche, un outil*)
5. Profondément **réseau** et **multi-utilisateurs**

# Linux et le libre

- Linux est un système d'exploitation sous **licence libre**
  1. liberté d'usage, sans restriction
  2. liberté d'étude du logiciel et de modification
  3. liberté de copie et diffusion
  4. liberté de diffusion des modifications

- Pour 2. : importance du **code source**
- Sphère privée (1-2) / sphère publique (3-4)

- Licence **GPLv2** : General Public License
  Il existe d'autres licences libres (ex : BSD, MPL...)

- Projet GNU : Le complément du noyau...

# Le projet GNU : *GNU's Not Unix*

- Origine (1983) : réimplémentation libre des utilitaires Unix
  - `glibc` + `gcc` : GNU C library + GNU C Compiler
  - `binutils` (ld, as, gprof, nm, ar, strings...), make, gdb...
  - `coreutils` (ls, chmod, sort, du, nice...), grep, sed, awk
  - `bash` : shell compatible sh

- Récemment : focalisation sur les projets "stratégiques"
  - GNU Hurd : noyau libre (pas opérationnel, cf. Linux)
  - Gnu Privacy Guard : crypto personnelle (alternative à PGP)
  - Gnome : environnement de bureau (alternative à KDE)
  - Gnash : lecteur Flash libre (alternative à Adobe...)
  - ...

- Logiciels indépendants
  - Emacs (1976-) : éditeur texte original, alternatif à `vi`
  - GIMP : retouche d'images
  - Dia : conception de diagrammes
  - ...

# Linux et le libre

- Il existe des logiciels propriétaires pour Linux (ex. serveur Oracle)
- Il existe des logiciels libres pour Windows... (ex. Apache, Mozilla Firefox, OpenOffice.org)
- Il existe d'autres OS libres (ex. FreeBSD)
- Libre n'est pas gratuit
  - parfois si : Linux est libre **et** gratuit
  - *freeware* : gratuit, pas libre (code source)
  - développements à façon : libre, pas gratuit...

# En résumé : Unix, GNU, Linux...

1. Unix : expérimental, universitaire
   Bell Labs, Université Berkeley... : "libre" sans le savoir
2. Unices propriétaires
   (AIX, HP-UX, SunOS / Solaris...)
3. Unices propriétaires et outils libres
   projet GNU, gcc, libc...
4. GNU/Linux : tout est libre (ou presque)
   - le noyau Linux
   - les applications et services (GNU, Apache, Eclipse...)
   - les distributions Linux

# Les distributions Linux

Leur rôle :

- Simplifier la vie de l'administrateur.

Une distribution comprend :

- le noyau Linux
- un système d'installation
- des logiciels applicatifs
- des outils d'administration

- Éventuellement
  - un support physique (boîte, CDROM, documentation...)
  - des services (maintenance, hotline, formation...)

# Les distributions Linux - Diversité (2)

Près de 400 distributions actives.

Cf. `http://distrowatch.com/` et `http://futurist.se/gldt/`

Causes de diversité :

1. Modèle de développement
   - communautaire : Slackware, Debian et certaines dérivées...
   - commerciale : la plupart des autres
2. Modèle d'administration
   - Installation des logiciels (.deb / .rpm / .tar.gz)
   - Services (Redhat / Fedora)
3. Spécialisation
   - Autonome : Knoppix, Kaella
   - Grand public : Ubuntu
   - Sécurité réseau : IP Cop
   - Localisation : Mandriva
   - Dépannage : System Rescue
   - Recompilation (performances) : Gentoo

# Les distributions : la famille RedHat

- RedHat Linux (ancien modèle) : RH 1.0 (1994) à RH 9 (2003)
  - mise au point du format RPM (RedHat Package Manager)
- RedHat Enterprise Linux (RHEL) : depuis RHEL 3 (2003)
  - dernière : RHEL 6.2 (déc. 2011)
  - plusieurs variantes : Desktop, Workstation, ES, AS...
- Fedora (Core)
  - version communauté
  - dével. rapide (env. 2/an) depuis FC 1 (nov. 2003)
  - dernière : Fedora 16 (nov. 2011)
- CentOS
  - clone de RHEL, sans le service
  - utilise les sources fournies par RedHat
- autres utilisatrices de RPM : Mandriva, Novell SuSE...

# Les distributions : la famille Debian

- Debian GNU/Linux : 1.0 (1996) à 6.0 Squeeze (fév. 2011)
  - collaborative et non commerciale
  - essentiellement libre
  - format de paquets (avancé) .deb
  - dépôts et installation réseau
  - mises à jour régulières (6.0.4 jan. 2012)

- Ubuntu : commerciale (Canonical LTD, GBM)
  - installation simplifiée
  - deux sorties par an (ex. 11.04 et 11.10)
  - partiellement compatible Debian
  - basée sur Gnome, choix restreint de paquets

- Knoppix : distribution autonome (*live*)
  - s'exécute sans installation (depuis le CD et la RAM)
  - peut s'installer et se transformer en Debian

# Administration Linux : les paquets

Chaque distribution propose un système d'installation de logiciels via des paquets (.deb / .rpm / .tar.gz).

Avantages :

- Normalisation
- Simplification
- Gestion des dépendances
- Mise à jour centralisée

Remarque : possible d'installer un programme sans ce procédé.

# Un effort de normalisation pour Linux

- Linux Standard Base (LSB)
  - 2001 (1.0) - 2011 (4.1) ...
  - dérivée / inspirée de POSIX
  - indépendante des distributions (mais RPM-centrée)
  - normalisation des composants (bibliothèques...)
  - normalisation de la hiérarchie (FHS)
  - fourniture de tests de compatibilité
- Linux Foundation
  - créée en 2007 : fusion de l'OSDL et du FSG
  - sponsorise Linus Torvalds et d'autres développeurs
  - édite la LSB et d'autres documents de référence (OpenPrinting...)

# Les communautés du libre...

- Notion de communauté
  - modèle propriétaire : césure développeurs / utilisateurs
  - modèle libre : tous les intermédiaires
- Participation à la communauté
  - le « pot commun » : mutualisation et réciprocité
  - support informel (forums, listes de diffusion)
  - rapports de bugs (et plus)
- Émergence d'outils techniques
  - Internet et communication (mail, newsgroups)
  - Gestionnaires de versions (code source)
  - Suivi de bugs / de tickets (Bugzilla...)
  - SourceForge, GForge...

# Logiciel libre : économie de services

- Économie de l'immatériel
  - Une idée n'est pas un bien matériel
  - Le partage n'appauvrit pas
  - Le logiciel "en boîte" est un leurre
- Des modèles économiques multiples
  - Constructeur : vend du matériel, donne le logiciel
  - Services : expertise, formation, développements sur mesure
  - Éditeur
    - hébergement (Software as a Service), *cloud*
    - audit, expertise
    - double licence, licence "chronodégradable"
- Quelques points délicats
  - Relations éditeur / communauté
  - Conditions de contribution
  - L'*open source* comme argument marketing

# La "professionnalisation" de Linux

- Linux Foundation
- Linux Standard Base
- Linux Professional Institute : certification
  - Linux Essentials (2012)
  - LPIC-1 : administrateur junior
  - LPIC-2 : administrateur avancé
  - LPIC-3 : administrateur senior (3 spécialisations...)

# Avantages du libre

- Éthique : collaboration, partage
  concerne : enseignement, administrations...
- Économie : redéploiement coûts achat vers services (formation, support)
- Pérennité et indépendance : moins lié à un éditeur
- Souplesse : adaptabilité aux besoins
- Mutualisation (coûts de développement)
  concerne : administration, collectivités locales...

# Bases de la ligne de commande

# Linux au démarrage *101.2*

En général (poste de travail) :

1. BIOS / EFI...
2. Chargeur de démarrage (GRUB ou LILO)
3. Mode texte
4. Mode graphique
5. Authentification par login + mot de passe
6. Bureau utilisateur (KDE, Gnome, XFCE...)

# Linux au démarrage *101.2*

En général (poste de travail) :

1. BIOS / EFI...
2. Chargeur de démarrage (GRUB ou LILO)
3. Mode texte
4. Mode graphique
5. Authentification par login + mot de passe
6. Bureau utilisateur (KDE, Gnome, XFCE...)

On peut aussi avoir (serveur) :

1. BIOS / EFI ...
2. Chargeur de démarrage (GRUB / LILO)
3. Mode texte
4. Authentification par login + mot de passe
5. Shell (en mode console)

Changement de mode : **Ctrl** + **Alt** + **F1-F6/F7**

# Ligne de commande vs interface graphique

- Inconvénients de la ligne de commande

- Avantages de la ligne de commande

# Ligne de commande vs interface graphique

- Inconvénients de la ligne de commande
  - apprentissage plus long
  - efficacité moindre (utilisateur débutant)
  - mémorisation nécessaire (partiellement)
  - domaine d'application limité (mais pas tant que ça...)
- Avantages de la ligne de commande

# Ligne de commande vs interface graphique

- Inconvénients de la ligne de commande
  - apprentissage plus long
  - efficacité moindre (utilisateur débutant)
  - mémorisation nécessaire (partiellement)
  - domaine d'application limité (mais pas tant que ça...)
- Avantages de la ligne de commande
  - automatisation aisée
  - efficacité (rapidité) supérieure (utilisateur aguerri)
  - ressources négligeables (CPU, réseau...)
  - expressivité plus forte (options)
  - modularité et extensibilité (une tâche, un outil)
  - compréhension et contrôle des actions

# Session utilisateur

Comptes utilisateurs :

- session : login/mot de passe *(username/password)*
- homedir : répertoire personnel
- permissions d'accès aux ressources (fichiers, processus) :
  - utilisateur
  - groupe
  - autres

Un compte unique d'administrateur (super-utilisateur) : root

Des comptes "services"

- pour les tâches système : mail, impressions, ...
- des droits restreints (par rapport à root)
- sécurité accrue en cas de bug ou compromission

# Découverte du shell - 1

*103.1*

## Le prompt (invite de commandes)

- utilisateur courant
- nom de machine
- répertoire courant
- $ ou # : terminateur
- ... configurable à l'extrême
- un curseur !

# Découverte du shell - quelques commandes

*103.1*

| | |
|---|---|
| id | Qui suis-je ? |
| pwd | Où suis-je ? |
| uname -a | À qui ai-je l'honneur ? |
| lsb_release -a | Mais encore ? |

| | |
|---|---|
| ls | Liste les fichiers |
| cd | Changement du répertoire courant |
| man | Page de manuel d'une commande |
| cat | Affiche le contenu d'un fichier |

# Commandes : syntaxe générale *103.1*

Syntaxe :

```
commande [options] [- -] [paramètres]
```

Exemples :

- `ls --help`
- `ls -a`
- `ls --all`
- `ls -al`
- `ls -l .bashrc`
- `ls -w 60`
- `ls -w60`
- `ls --width=60`

Remarques : quelques exceptions

- `find . -name '*.tex' -print`
- `dd if=/dev/hda1 of=hda.img bs=512`

# Commandes internes et externes *103.1*

- Commandes d'identification
  - `which` : commandes externes (fichiers)
  - `type (-a)` : commandes connues du shell
  - `whereis` : binaire et page de man d'une commande
- Les principaux types de commandes
  - **commande externe (fichier exécutable) - hashed**
  - **commande interne ou primitive shell (builtin)**
  - alias
  - fonction shell
  - mot-clé du shell, ex. `if`, `for`
- Exo : déterminer le type des commandes suivantes

  `cd, cp, ls, which, type, echo`

# Documentation - formats et logiciels *103.1*

- aide en ligne de commande
  `ls --help`
- aide de bash : `help` (commandes internes)
- pages de manuel : `man`
  cf page suivante
- `info` : la documentation GNU (voir aussi `pinfo`, `tkinfo`)
- et encore : des pages .html, des fichiers `README`, `.chm`...
  voir `/usr/share/doc/`
- navigateurs d'aide (Gnome, KDE...) : interne, man, info...

# Documentation - manpages *103.1*

- `man ls`, `man man`

- Contenu organisé en neuf sections

| | | |
|---|---|---|
| **1. commandes util.** | 4. périphériques | 7. “conventions” |
| 2. appels noyau (C) | **5. fichiers conf.** | **8. commandes admin.** |
| 3. appels bibli. (C) | 6. jeux | 9. routines noyau |

`man (1) man`, `man 7 man`

- Parties génériques : Nom, **Synopsis**, Description, Auteurs, **Voir aussi**...

- Pager (`less` par défaut) intégré
  - défilement : flèches curseur, espace, lettres
  - recherche : /motif, `n`, `N`, `Esc-U`...
  - marqueurs : `m`a pose un marqueur ; `'`a y retourne

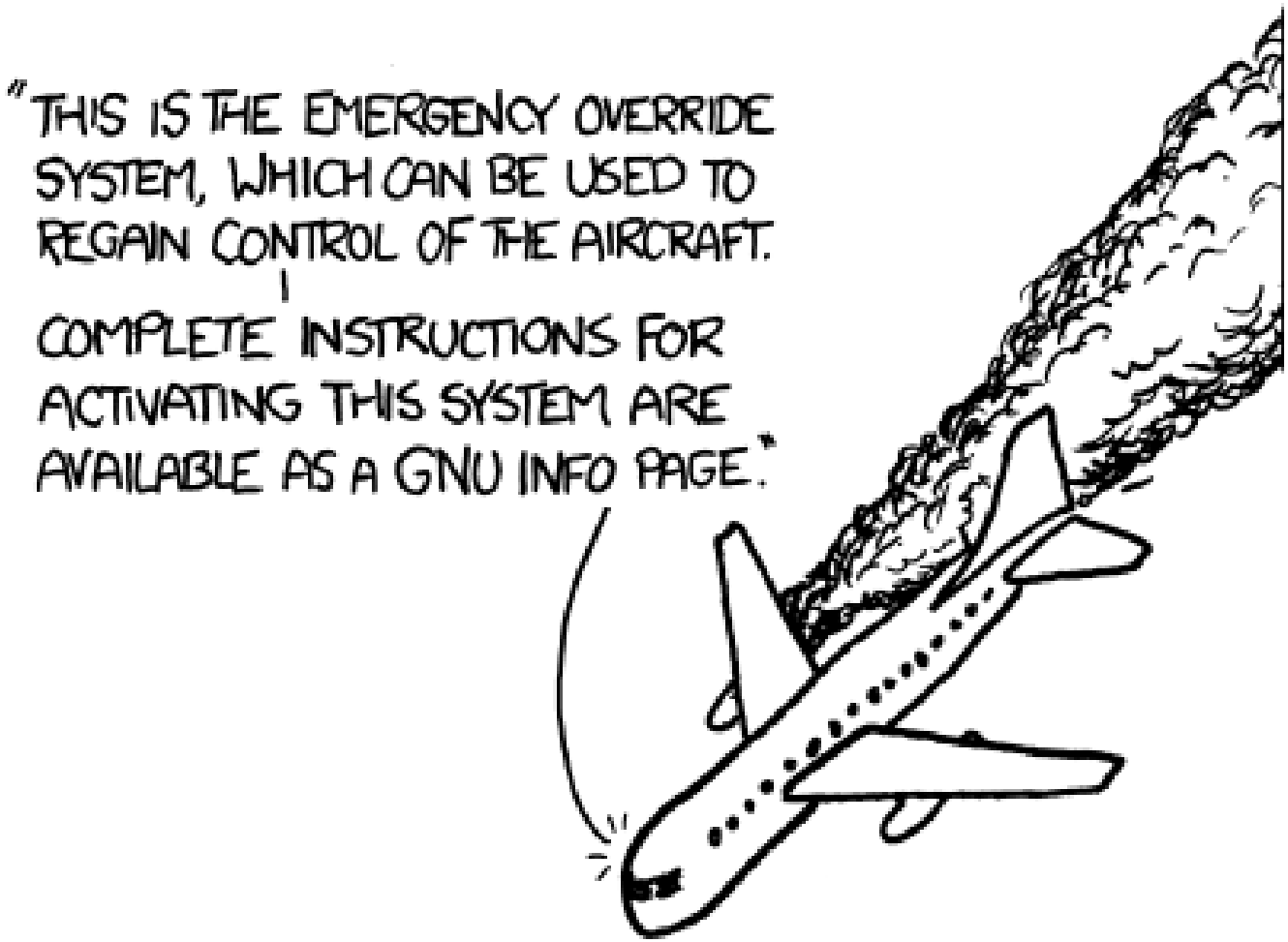
# Autour de man

*103.1*

- Commandes auxiliaires
  - `whatis` (ou `man -k`) affiche la description brève
    ex. `whatis [-s 1] man whatis apropos info help`
  - `apropos` (ou `man -f`) recherche dans l'index
    ex. `apropos --and interface réseau`
- Pour les experts...
  - `mandb` indexe les pages de manuel
  - `catman` stocke en cache les pages formatées
  - survivant du système `roff` (formatage à balises, adapté aux terminaux texte)
  - cf `man 7 man`

# XKCD 912 - Manual Override

*103.1*




(C) Randall Munroe, CC-BY-NC

http://xkcd.com/912/

# Gestion des fichiers

# Gestion des fichiers et répertoires *103.3*

- Commandes courantes
  - informatives : `ls, cat`
  - modificatrices : `touch, cp, mv, rm`
  - répertoires (informatives) : `pwd, cd, du, tree`
  - répertoires (modificatrices) : `mkdir, rmdir`
- Spécificités Unix
  - métadonnées Unix : `stat`
  - propriétaires : `chown`, `chgrp`
  - permissions : `chmod`
  - liens : `ln (-s)`, `readlink`

# Récapitulatif : chemins relatifs et absolus *103.3*

- Chemins absolus : exemples
  - `ls /home/stg1/Linux`
  - `ls ~stg1/Linux`
  - `ls ~/Linux`
- Chemins relatifs : exemples
  - `ls Linux`
  - `ls ./Linux`
  - `ls ../AutreRepertoire`
- Ne pas confondre : fichiers et répertoires cachés
  ex. `ls -l ./.bashrc`

# Globbing (expansion des noms de fichiers) *103.3*

## But

Ne pas avoir à taper le nom de tous les fichiers en argument.

## Exemple

```
ls *.rc
```

## Caractères spéciaux

- `*` Tout
- `?` Un caractère quelconque
- `[a-z]` Un caractère parmi ceux listés

## Protections contre l'interprétation par le shell

- `"..."` Protège partiellement ... de l'interprétation par le shell
- `'...'` Aucune interprétation de ...
- `\...` Aucune interprétation du caractère suivant

# Pour aller plus loin : globbing personnalisé *103.3*

- Personnalisation du globbing
  - Commande shell `shopt (-s | -u) option`
  - Variable d'environnement : `$GLOBIGNORE`

- Options concernant le globbing

| | |
|---|---|
| dotglob | inclut les fichiers "cachés" |
| failglob | erreur si rien ne correspond |
| globstar | récursif avec ** et **/ |
| nocaseglob | insensible à la casse |
| nullglob | chaîne vide si rien ne correspond |
| extglob | motifs étendus |

# Pour aller plus loin : globbing personnalisé *103.3*

- Personnalisation du globbing
  - Commande shell `shopt (-s | -u) option`
  - Variable d'environnement : `$GLOBIGNORE`

- Options concernant le globbing

| | |
|---|---|
| `dotglob` | inclut les fichiers "cachés" |
| `failglob` | erreur si rien ne correspond |
| `globstar` | récursif avec `**` et `**/` |
| `nocaseglob` | insensible à la casse |
| `nullglob` | chaîne vide si rien ne correspond |
| `extglob` | motifs étendus |

# Gestion des répertoires *103.3, 104.2*

- Rappels
  - `cd <cible>` changer de répertoire courant
  - `pwd` afficher le répertoire courant
- Création
  - `mkdir Toto`
  - `mkdir [-v] -p Toto/Titi/Tata/Tutu`
- Suppression
  - `rmdir [-p] <cible>`
  - `rm -rf <cible>` si les répertoires sont peuplés !
- Place disque occupée (104.2)
  - `df [-k|-m|-h] [-t ext3] ...` (*disk free*) tailles partitions
  - `du [-k|-m|-h] [-s] <cible>` (*disk usage*) tailles répertoires

# Gestion courante des fichiers *103.3*

- Suppression : `rm [-r] [-f|-i|-I] [-v] <cibles>`
- Copie : `cp [-r] [-f|-i|-n] [-v] <source> <destination>`
  ou `cp [...] -t <rép-destination> <sources>`
- Déplacement : `mv [-f|-i|-n] [-v] <source> <destination>`
  ou `mv [...] -t <rép-destination> <sources>`
- Renommage : `mv [-f|-i|-n] [-v] <source> <destination>`
  - renommage simple (emplacement fixe)
  - renommage avec déplacement
- Pour aller plus loin
  - `mmv` (et dérivées) renomme fichiers multiples d'après un motif
  - `rename` expressions régulières et réécriture (perl)

# Propriétaires d'un fichier *103.3*

## Commandes principales

- `chown [-R...] user:group file`
- `chgrp [-R...] group file`
- avec `user` défini par `uid` ou `username`
- avec `group` défini par `gid` ou `groupname`

## Commandes auxiliaires

- `groups` : à quels groupes appartient l'utilisateur
- `su` : changer d'utilisateur
- `adduser user group`

# Permissions sur les entrées de répertoires *103.3*

## Trois cibles de permissions

- u=user : utilisateur propriétaire
- g=group : groupe propriétaire
- o=other : tous les autres
- (a=all : tout le monde)

## Trois types de droits

| | sur fichier | sur répertoire |
|---|---|---|
| r=read | lecture | listage |
| w=write | écriture | ajout/suppression fichier |
| x=exec | exécution | traversée |
| *X=exec* | *conditionnelle* | *traversée* |

# Modifier les permissions *103.3*

## Commande `chmod`

1. `chmod u=rwx,g=rx,o= <fichier>`
2. `chmod u+w,a+x <fichier>`
3. `chmod -R g=u <Répertoire>` : affecter au groupe les permissions du propriétaire
4. `chmod --reference=<fichier-ref> <cible>`

## Notation octale

- r=4, w=2, x=1 ex. `rwx r-x r- -` = 754
- `chmod 750 <fichier>`

# Permissions : Travaux pratiques *103.3*

## Exercice : Remise des devoirs

Un enseignant cherche à récolter les programmes rédigés par ses étudiants dans un répertoire commun.
Tous doivent pouvoir déposer un fichier, mais aucun ne doit pouvoir lister ni lire les autres fichiers déposés.

1. Mettre en place la configuration nécessaire, ouverte à tous les utilisateurs.
2. Comment restreindre le dépôt à un groupe de TP, nommé `tp01` ?
3. Comment éviter les conflits de nommage entre plusieurs étudiants ?

# Permissions Unix - Compléments *103.3*

## SUID et SGID

- suid : changement d'UID à l'exécution `chmod u+s fichier`
- sgid : changement de GID `chmod g+s fichier`

## Sticky bit

- fichier : obsolète
- répertoire : restriction à l'ajout/suppression d'entrées
  `chmod +t rép.`

- `man chmod`
- Extension ACL : Access Control List `acl(5)`, `getfacl(1)`, `setfacl(1)`

# Métadonnées Unix *103.3*

- Commande `stat` : sur fichier ou système de fichiers
- Permissions
  - utilisateur propriétaire : `uid` numérique
  - groupe propriétaire : `gid` numérique
  - `mode` r,w,x... (champ de bits) ex. `0644/-rw-r-r-`
- Horodatage
  - `atime` (access) : dernier accès (lecture) `ls -lu`
  - `ctime` (change) : modification des métadonnées (inode) `ls -lc`
  - `mtime` (modification) : modification du contenu `ls -l`
  - `touch` : mise à jour ⇒ atime, mtime falsifiables, ctime sûr !

  Exo : que devient l'horodatage en cas de : `cat`, `vim` (avec et sans modif), `mv` (renommage), `chmod` ?
- Auxiliaires
  - type de fichier (régulier, répertoire...)
  - taille en octets
  - compteur de liens

# Liens physiques et liens symboliques - en pratique *103.3*

```
$ touch fichier
$ cp fichier fichier-cp
$ ln fichier fichier-ln      #lien physique
$ ln -s fichier fichier-lns  #lien symbolique

$ ls --inode --long

2080774 -rw-r--r-- 2  [...] fichier
2080775 -rw-r--r-- 1  [...] fichier-cp
2080774 -rw-r--r-- 2  [...] fichier-ln
2080776 lrwxrwxrwx 1  [...] fichier-lns -> fichier

$ ln -s fichier-lns fichier-lns2
$ readlink fichier-lns2
$ readlink -f fichier-lns2
```

# Liens physiques et liens symboliques - inodes *103.3*

## Usages des liens symboliques

- Alternatives ex. `vim -> /usr/bin/vim.basic`
- Rétro-compatibilité ex. `/tmp -> /var/tmp`
- “Raccourcis” ex. `./doc -> /usr/share/doc/debian-reference-fr`

## Usage des liens physiques

- relativement obsolète
- "instantané" (snapshot), cf `rsnapshot`

# Liens physiques et liens symboliques - inodes *103.3*

## Structure du système de fichiers - inodes

- répertoires
- inodes (métadonnées)
- contenus

# Liens physiques et liens symboliques - comparaison *103.3*

| | **lien symbolique** | **lien dur** |
|---|---|---|
| pointe sur | entrée de répertoire | inode |
| rôle | asymétrique | symétrique |
| cible | tout type | fichier régulier |
| chemin cible | absolu ou relatif | N.A. (inode) |
| système de fichiers | interne ou externe | interne |
| cohérence | peut être cassé | jamais cassé |
| stockage | fichier (spécial) | entrée de répertoire |

# Archives et compression *103.3*

- L'archivage : rassembler plusieurs fichiers en un seul.
  `tar -c`, `tar -x`, `tar -t`
  `-f archive.tar` : spécifier le fichier archive (sinon flux)

- La compression
  - `gzip` + `gunzip` (ou `tar -z ...`)
  - ...

- Exercice
  1. Prendre connaissance du contenu de `tp-access.tgz`
  2. Décompresser l'archive
  3. Créer une archive compressée avec les 20 premiers fichiers
  4. Compresser individuellement les 20 derniers

- Autres implémentations : S-tar (`star`)...
- Unix historique : `cpio` + `compress` (.Z)

# Compression *103.3*

- Utilitaires et algorithmes de compression
  - `gzip` + `gunzip` (ou `tar -z`), `zcat`, `zless` ... — algo LZ77
  - `bzip2` + `bunzip2` (ou `tar -j`), `bzcat` ... — algo Bzip2
  - `xz` + `unxz` (ou `tar -J`), `xzcat` ... — algo LZMA
  - `compress` + `uncompress` (ou `tar -Z`) — obsolète, algo LZW

- Travaux pratiques
  1. *benchmarking* des temps et tailles entre les compressions `gzip`, `bzip2` et `xz`.
  2. autres variables : niveaux de compression des algorithmes, taille mémoire...

- Compatibilité Windows : `zip`, `7z` (paquets `zip` et `p7zip-full`)

# Rechercher un fichier... 1/2 Indexation *104.7*

- `locate` : recherche rapide dans une base de données
  - `locate` (GNU) : source `findutils`
    - `-r` expression régulière, ex. `-r fst.b`
    - `-S` statistiques ...
  - `slocate` (obsolète) : + permissions
  - `mlocate` : + optimisation base

  - TP : Avec `updatedb` : lancer une indexation personnelle de son répertoire

- Fichiers et paquets (distribution)
  - (Debian) `dlocate` : recherche parmi les paquets installés alternative rapide à `dpkg -S`
  - (RH) `rpm -qf`

# Rechercher un fichier - 2/2 Find

*104.7*

- `find` : recherche multicritères
  `find /etc/ -size +10k -ctime -10 -printf '%s %p \n'`
  - répertoire de départ (/etc)
  - options de sélection (size, ctime)
  - options d'action (printf)

  - Toujours à jour
  - Potentiellement plus long que `locate`

- Exercices
  - Pour aller plus loin : options `find -H -L -P`
  - Trouver le nombre d'entrées de répertoire de chaque type sous `/`, sans changer de système de fichiers (`-xdev`).

  - Pour les quatre types minoritaires, afficher les entrées.

# Récapitulatif : différences avec le système de fichiers de Windows

- Pas de notion de lecteur C: D: etc.
- Tout est dans une même arborescence, de racine /
- Les répertoires sont séparés par des / et non des \
- Existence de liens symboliques
  `ln -s fichier lien`
  Sous windows, les liens sont de simples fichiers *.link*
- Des permissions explicites

# Principaux types de fichiers

# Principaux types de fichiers

- Trois principales distinctions :
  - texte ou binaire
  - exécutable ou pas
  - installé par la distribution ou pas

# Principaux types de fichiers

- Trois principales distinctions :
  - texte ou binaire
  - exécutable ou pas
  - installé par la distribution ou pas
- Quelques exemples :
  - programmes binaires, ex. `/bin/cp`
  - scripts shell, ex. `/etc/init.d/rc.local`
  - fichiers de configuration, ex. `/etc/fstab`
  - fichiers de log, ex. `/var/log/messages`
  - bibliothèques dynamiques **.so**

# Principaux types de fichiers

- Trois principales distinctions :
  - texte ou binaire
  - exécutable ou pas
  - installé par la distribution ou pas
- Quelques exemples :
  - programmes binaires, ex. `/bin/cp`
  - scripts shell, ex. `/etc/init.d/rc.local`
  - fichiers de configuration, ex. `/etc/fstab`
  - fichiers de log, ex. `/var/log/messages`
  - bibliothèques dynamiques **.so**
- Commandes utiles
  - `file` : le type du fichier
  - `which` ou `type` : pour une commande
  - `cat, head, tail` : le contenu du fichier (texte)
  - `hd, ldd, strings...` : le contenu du fichier (binaire)

# Exécutables interprétés et compilés *104.7*

- Langages interprétés
  - interpréteur standard : shell (`bash` ou ...)
  - autres : perl, python, ruby, php
  - *shebang* (ou *hashbang*) : `#! /usr/bin/perl -w`
  - interpréteur nécessaire pour l'exécution
  - code source = exécutable

- Langages compilés
  - entrée : code source texte ex. C, C++, Fortran...
  - chaîne de compilation : `gcc`, `as`, `ld`
  - sortie : binaire exécutable ELF (...)
  - source (C...) $\longrightarrow$ compilateur $\longrightarrow$ exécutable ELF
  - code source $\neq$ exécutable

# ELF : Executable and Linkable Format *104.7*

## Le format standard des exécutables Linux

- Buts
  - Assembler les unités de compilation (*.o)
  - Créer une image mémoire d'un programme

- Trois sous-types de fichiers ELF

  EXEC binaire exécutable
  DYN fichier objet partagé `*.so`
  REL fichier relocalisable `*.o, *.a`

- Commandes disponibles
  - `file /bin/ls` ⟶ ELF 32-bit LSB executable [...]
  - Pour aller plus loin : `readelf -h`, `nm`, `objdump`

# Pour aller plus loin : file et MIME

## Comment déterminer un type de fichiers ?

- Plusieurs concepts à distinguer
  - l'extension du fichier (si elle existe) : métadonnée
  - sa signature (si elle existe)
  - son type MIME (Multipurpose Internet Mail Extensions)
  - les applications le prenant en charge
- Techniquement
  - `libmagic` à la base de `file` : `man magic`
  - `file -i` renvoie le type MIME
  - `/etc/mime.types`
  - `/etc/mime-magic` et `/etc/magic`

# Filesystem Hierarchy Standard 1/2

*104.7*

Norme FHS maintenue par la Linux Foundation

| | |
|---|---|
| / | racine |
| /home/ | répertoires utilisateurs |
| /root/ | homedir de root |

# Filesystem Hierarchy Standard 1/2 *104.7*

Norme FHS maintenue par la Linux Foundation

| | |
|---|---|
| / | racine |
| /home/ | répertoires utilisateurs |
| /root/ | homedir de root |
| /bin/ | exécutables principaux (système) |
| /sbin/ | exécutables d'administration (superuser) |
| /etc/ | configuration du système |

# Filesystem Hierarchy Standard 1/2

*104.7*

Norme FHS maintenue par la Linux Foundation

| | |
|---|---|
| / | racine |
| /home/ | répertoires utilisateurs |
| /root/ | homedir de root |
| /bin/ | exécutables principaux (système) |
| /sbin/ | exécutables d'administration (superuser) |
| /etc/ | configuration du système |
| /usr/ | programmes (gérés par la distribution) |
| /usr/bin/ | exécutables des programmes |
| ... | |
| /usr/local/ | programmes (hors distribution) |

# Filesystem Hierarchy Standard 1/2 *104.7*

Norme FHS maintenue par la Linux Foundation

| | |
|---|---|
| / | racine |
| /home/ | répertoires utilisateurs |
| /root/ | homedir de root |
| /bin/ | exécutables principaux (système) |
| /sbin/ | exécutables d'administration (superuser) |
| /etc/ | configuration du système |
| /usr/ | programmes (gérés par la distribution) |
| /usr/bin/ | exécutables des programmes |
| ... | |
| /usr/local/ | programmes (hors distribution) |
| /var/ | données variables |
| /var/log | fichiers de log |
| /var/spool | fichiers tampons (mail, impressions...) |

# Filesystem Hierarchy Standard 2/2 *104.7*

## Extensions...

| | |
|---|---|
| /opt | applications installées hors conventions Unix |
| /mnt | montages externes (réseau...) |
| /media | montages amovibles (CD, clé USB...) |
| /srv | données utilisées par les services (FTP, WWW...) |
| /selinux | réservé pour Security Enhanced Linux |
| /run | données runtime (remplace /var/lock et /var/run) (non-LSB) |

## Systèmes “virtuels” (tout est fichier...)

| | |
|---|---|
| /dev | fichiers-périphériques |
| /proc | informations sur les processus : `man 5 proc` |
| /sys | informations système |

# Points de montage (introduction) *104.7*

Comment accéder à un CD-ROM sans D: ?

# Points de montage (introduction) *104.7*

Comment accéder à un CD-ROM sans D: ?

`mount /media/cdrom`

## Les points de montage

Initialement, seule existe la racine /.

Puis `mount` sert à associer

- un périphérique physique (disque, partition) ex. `/dev/sda2`
- un répertoire ex. `/mnt/windowsC`

Exemple : `mount -t vfat /dev/sda2 /mnt/windowsC`

Les montages par défaut sont décrits dans `/etc/fstab`.

`mount` (sans argument) liste les montages en cours.

Pour aller plus loin

- automontage : clés USB, périphériques *hotplug*
- montage par l'interface graphique

# Outils en ligne de commande

# Redirections - canaux *103.4*

## Le shell définit 3 canaux

STDIN (0) entrée standard - clavier par défaut

STDOUT (1) sortie standard - écran (terminal) par défaut

STDERR (2) sortie d'erreur - écran (terminal) par défaut

## Redirection

```
ls -1 > liste.txt
```

La sortie du programme `ls` est **redirigée** vers un fichier.

Pour **ajouter** au fichier (sans écraser l'ancien contenu) :

```
ls -1 >> liste.txt
```

`2>` redirection de la sortie d'erreur

`&>` redirection des deux sorties

`<` redirection d'entrée, ex. `cat < liste.txt`

# Pipes et filtres

*103.4*

`ls -1 | wc` sortie de `ls` canalisée vers l'entrée du filtre `wc`.

`find /etc |& wc` StdOut et StdErr fusionnées puis canalisées

## Exemples

1. `cat` taper `Ctrl+D` = fin de flux
2. `cat liste.txt | wc -l`
3. `wc -l liste.txt`
4. `wc -l < liste.txt`
5. `cat < liste.txt | wc -l`
6. `wc -l liste.txt l2.txt l3.txt`
7. `cat liste.txt l2.txt l3.txt | wc -l`

Exo. Dessiner le schéma correspondant à chacune des commandes. Identifier filtres et semi-filtres.

# Filtres textes courants *103.2*

Principe Unix : une tâche, un outil.

Beaucoup de filtres fonctionnent ligne par ligne :

- `head` Premières lignes
- `tail` Dernières lignes
- `sort` Trie les lignes
- `uniq` Enlève les doublons
- `grep` Garde les lignes correspondant à une expression donnée.
  Ex. `ls / | grep v`
- `cut` Conserve les colonnes (resp. champs) donnés

- moins courants : `tr`, `tac`, `paste`, `fmt`...
- paquet `coreutils`

# TP : synthèse de logs *103.2*

Le fichier `access.log` contient un extrait de logs du serveur Apache, duquel on va essayer de tirer des statistiques.

1. Combien de requêtes sont enregistrées dans le fichier access.log ?
2. Extraire du fichier access.log la liste des adresses IP clientes.
3. Compter le nombre d'adresses IP différentes.
4. Afficher le nombre d'occurrences de chaque IP, puis présenter la liste par nombre décroissant d'occurrences.
5. Afficher uniquement les IP ayant effectué au moins 10 accès.
6. Question subsidiaire : pour chacune des IP de la liste précédente, effectuer une résolution de nom (commande host).
   a. en passant par un fichier temporaire
   b. sans intermédiaire, en une seule ligne de commande

# TP : manipulation de texte *103.2*

Le fichier `auteurs.txt` contient une liste d'auteurs avec leur fréquence d'apparition. Ceux qui sont placés entre « ... » sont identifés clairement, à la différence des autres.

1. Séparer énoncé et données dans deux fichiers différents.
2. Combien y a-t-il d'auteurs au total ? Combien de bien identifiés ? De mal identifiés ?
3. Classer les auteurs selon leur fréquence.
4. Lister les 20 auteurs les plus courants, le plus fréquent en premier.
5. Créer un fichier `auteurs2.txt` dans lequels ne figurent pas les auteurs n'ayant qu'une occurrence. Combien sont-ils ?
6. Quels sont les 10 auteurs mal identifiés qui apparaissent le plus souvent ?

# Pour aller plus loin : Sed et Awk *103.2*

- Sed : Stream Editor
  - adapté aux opérations sur les chaînes et les regexp
  - `sed -e "s/Old/New/g" f-in > f-out`
- AWK : un langage-filtre
  - pour les fichiers structurés en colonnes ou en champs
  - `gawk -F: '$3 > 999' /etc/passwd`
- Encore plus loin : Perl

# Filtres - pour aller plus loin *103.2*

- La commande `tee` : brancher une dérivation
  ```
  egrep ":[0-9]:" /etc/passwd | tee listing | wc -l
  ```
- La commande `xargs` : transformer STDIN en arguments
  ```
  find /etc/ -size +100k | xargs wc -l
  ```
- La commande `mkfifo` : créer un pipe nommé
  ```
  mkfifo listing
  cut -d: -f1-3 listing
  egrep ":[0-9]:" /etc/passwd | tee listing | wc -l
  ```
  $\longrightarrow$ synchronisation forcée des processus

# Pour aller plus loin : fonctions avancées du shell

- Mode interactif
  - autocompletion
  - raccourcis clavier
  - historique
  - alias
- Scripts shell
  - gestion des arguments
  - boucles
  - tests et structures de contrôle (if ...)
  - fonctions
- Configuration du shell
- Choix du shell : `bash`, `tcsh`, `zsh`...

# Gestion des tâches *103.5*

## Commandes et mécanismes du shell

- `commande &` : lancer en arrière-plan
- `jobs (-l)`
- **Ctrl-C** : arrête (et termine)
- **Ctrl-Z** : met en pause
- `bg` : redémarre en arrière-plan le processus en pause
- `fg` : remet en avant-plan

## Notions de *session*

- exécutions liées au *shell interactif* courant
- liées également au *terminal (virtuel)* courant
- numéros de jobs, propres à la session courante (1, 2, 3...)

# Gestion des processus *103.5*

## Processus

- globaux à l'ensemble du système
- identifiés par un PID (Process Id), numéro entre 1 et 32768

## Affichage des processus

- `top` : affiche les ressources consommées par les processus
- `ps` : Process Show
- `pstree` : **arbre** des processus $\longrightarrow$ init
- `prtstat` : (paquet `psmisc`) tous les détails d'un processus
- `htop` : interface semi-graphique interactive
- `qps` : interface graphique conviviale

# Commande ps - les options *103.5*

1. syntaxe BSD : `ps U root`, `ps aux`
2. syntaxe SysV : `ps -U root`, `ps -ef`
3. syntaxe longue GNU : `ps --user root`

## Principales options

1. Options de sélection
   - -e, -A : tous les processus
   - -C <liste commandes>
   - -G, -U ... <liste utilisateurs, groupes>
   - -t, --tty <liste de terminaux>
2. Options de niveau de détail
   - -f, -F : full, extra-full
   - -o, -O, --format : personnalisé Ex. `ps -O ppid,pgrp,sess`
3. Options d'affichage
   - --sort : tri Ex. `--sort tt,-pid`
   - -H, --forest : hiérarchie
   - --headers --lines=20 : répéter l'en-tête toutes les 20 lignes

# Regroupement de processus 1/2 - sessions *103.5*

- **session (SESS)** : processus d'une session
- *session leader* (bash...) : fournit son PID à la session
- **TPGID** : groupe au premier plan du terminal (TTY) du processus
- `ps -t pts/1 -O ppid,sess`

## Terminaux et pseudo-terminaux

- Consoles virtuelles (TTY)
  - consoles texte standard (Alt + F1-F6...)
  - `/dev/tty1-63`
- Pseudo-terminaux (PTYs)
  - terminaux X, session shell...
  - `/dev/pts/0...` + `/dev/ptmx` (System V)

# Regroupement de processus 2/2 - groupes *103.5*

## Regroupements de processus

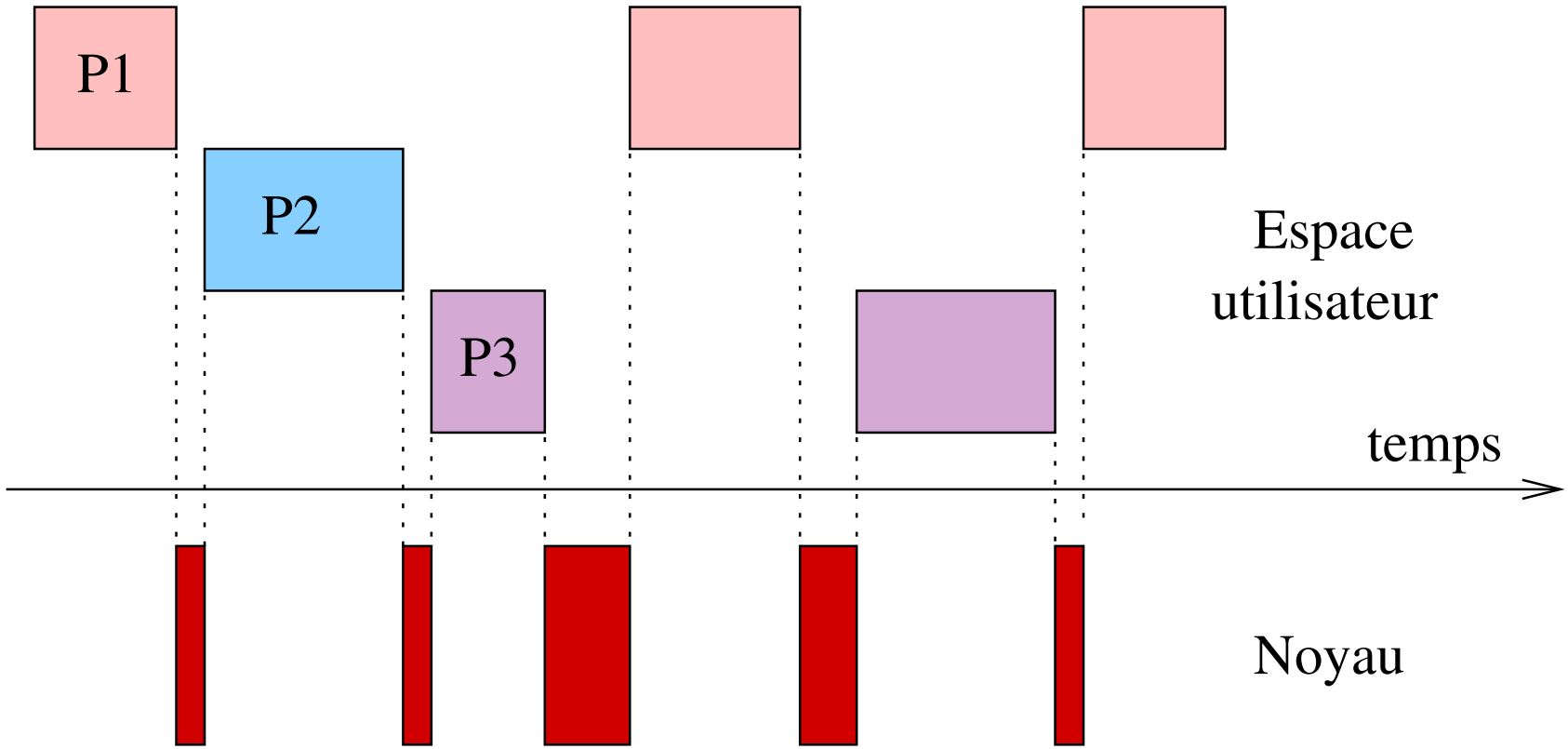
- **groupe (PGRP)** : processus formant une même commande (=job)
  ex. `find / | grep pass | less`
  *group leader* = `find`
- une *session* regroupe plusieurs *groupes*

## Exercice

- Combien de sessions différentes et de groupes différents tournent sur la machine ?

# Notions d'ordonnancement

*103.6*




## Paramètres

- fréquence : réactivité du système
- proportion : priorités des processus (cf *niceness*)

# Processus - états et ordonnancement *103.6*

## Etats des processus

| | |
|---|---|
| **R** | demande d'exécution (Running) |
| **S** | attente interruptible (Sleep) |
| D | attente non interruptible (I/O) |
| T | stoppé (par SIGSTOP) |
| Z | "zombie" (ou defunct) |

| | |
|---|---|
| < | priorité haute |
| N | priorité basse (Nice) |
| s | session leader |
| l | multi-thread |
| + | groupe d'avant-plan |

## Trois classes d'ordonnanceur (CLS)

- TS : Time Shared (standard)
- FF : Real Time Fifo
- RR : Real Time Round Robin

# Charge machine

*103.6*

## Commandes

- `w`, `uptime` (statique)
- `top` (dynamique)
- `xload` (graphique basique)

## Définition de la charge

- Nombre moyen de processus dans l'état exécutable (R) ou en attente non interruptible (D).
- Moyenne temporelle sur 1, 5 et 15 minutes.

# Priorité et "courtoisie" *103.6*

## Courtoisie (*niceness* NI)

- un nombre entier, entre -20 et 19
- -20 à -1 : réservé à root, priorités hautes
- 0 : valeur par défaut
- 1 à 19 : accessibles à tous, priorités basses

## Priorité (PRI) : calculée à partir de la courtoisie

- $PRI = 19 - NI$ en temps partagé
- $PRI = 19 - NI + 100$ en temps réel (FF, RR)

## Commandes

- nice commande `nice -n10 md5sum cd.iso`
- renice courtoisie PID `renice -n +20 5124`

# Contrôle des processus *103.5*

## Recherche multicritères d'un processus

Ex. `pgrep -u stg1 -t pts/0 -l --newest bash`

## Arrêter un processus

- `kill` [options] PID — `kill -TERM 1955`
- `killall` commande — `killall gimp`
- `pkill` [-signal]

## Un signal (Posix)

- un numéro, entre 1 et 31 (ex. `15`)
- un nom symbolique (ex. `SIGTERM`)
- une action par défaut (ex. terminer le processus)

## Pour en savoir plus

- référence : `signal (7)`
- le mécanisme le plus simple d'IPC (*inter-process communication*)

# Signaux *103.5*

## Les actions par défaut

| | |
|---|---|
| Term | terminer le processus |
| Ign | ignorer le signal |
| Core | créer un fichier core et terminer le processus |
| Stop | arrêter le processus (pause) |
| Cont | relancer le processus s'il est en pause |

## Les principaux signaux

| | |
|---|---|
| SIGTERM | (15) terminer normalement ("proprement") |
| SIGKILL | (9) terminaison forcée (non ignoré) |
| SIGSTOP | (19) arrêt temporaire (pause) (non ignoré) |
| SIGCONT | (18) reprise d'un processus arrêté |
| SIGINT | (2) terminaison interactive (Ctrl-C) |
| SIGTSTP | (20) arrêt temporaire interactif (Ctrl-Z) |
| SIGUSR1,2 | (30, 31) signal utilisateur 1 (resp. 2) |

# Pour aller plus loin : threads noyau *103.5+*

## Les threads noyau

- le démon `kthreadd (PID=2)`
- et tous ses fils : `ps -f --ppid=2`
- parfois liés à un processeur : `[ksoftirqd/0]`

## En pratique

- Combien de threads noyau sont en cours d'exécution ?
- Quel est le premier "vrai" processus utilisateur ? (hors init)

# Processus légers (threads) *103.5+*

## Les threads : des “sous-processus”

Partage de : **code**, données, E/S fichiers, signaux, *pile*

## Les threads utilisateurs : affichage avec `ps`

- `ps -L -f` : LWP (pid du thread), NLWP (nombre de threads)
- `ps -Lf -m` : sous-processus affichés après les processus “principaux”

## En pratique

- Combien de processus multi-threadés tournent ?
- Combien de threads au maximum ? Pour quel processus ?

# TP – Processus

*103.5*

1. Combien, approximativement, de processus ont été créés depuis le dernier démarrage du système ?
2. Lister les processus **bash** en cours.
3. Utiliser `top` pour trouver le processus utilisant le plus de mémoire. Tenter de l'arrêter.
4. Faire le lien entre `/proc/` et les processus. Cf `man 5 proc`

5. Trouver le processus de PID maximal, puis le dernier processus lancé
6. Chercher le taux de création des processus (en p/s).
7. Créer une fonction pour rechercher le père d'un processus donné, puis une autre pour déterminer la profondeur d'un processus donné (en argument)

# Pour aller plus loin...

- Surveiller un processus avec `watch`
  ```
  watch ls -l /var/log/messages
  watch -d ps -F
  ```
- Utiliser `wait` (interne) pour synchroniser des tâches (script)
- Utiliser `procinfo`
- Utiliser `unhide` pour chercher les processus dissimulés (rootkits...)
- Utiliser `pidstat` pour obtenir les ressources utilisées (paquet `sysstat`)

# Compilation d'un exécutable

- Exemple : compilation de `ncdu`
- Procédure standardisée : utilisation d'autoconf/automake
  - `./configure (--help)`
  - `make`
  - `make install`
- Dépannage : recherche de dépendances (bibliothèques dynamiques)

# Diff et Patch

- Commande `diff`
  - direct : entre deux fichiers
  - -c, -u : contexte, unifié
  - -r : récursif (entre répertoires)
- Commande `patch`
  - syntaxe `patch -p0 <patchfile`

# Éditeurs de texte

*103.8 ?*

Éditeurs sans interface graphique

- parfois nécessaire (connexion réseau, problème graphique)
- plus rapide

# Éditeurs de texte

*103.8 ?*

Éditeurs sans interface graphique

- parfois nécessaire (connexion réseau, problème graphique)
- plus rapide

- nano
  - simple d'utilisation
  - installé par défaut
- emacs -nw
  - puissant et configurable
  - généralement utilisé en mode graphique
- vi / vim
  - éditeur modal : déroutant au premier abord
  - puissant et efficace pour l'administration système

# vi / vim

*103.8*

- Historique Vi
  - `qed` → `ed` (K. Thomson) → `ex` → `vi`
  - 1976 par Bill Joy, étudiant à Berkeley (puis `csh`, `NFS`, Sun)
  - mode *visuel* de `ex` : premier éditeur "pleine page"
  - POSIX (IEEE 1003.2, Part 2 : Shell and utilities)
  - Développement stoppé en 1985 (licence Sun)

- Nombreuses variantes
  - `elvis`, Steve Kirkendall (Minix, Slackware), 1990-2003 ?
  - `nvi`, Keith Bostic (4.4BSD et dérivés libres), 1992-1996 ?
  - `vile` : VI Like Emacs

- VIM (Vi IMproved)
  - auteur Bram Moolenar (NE)
  - 1991 (1.0) - 2008 (7.2)...
  - toutes plateformes : Unix, Linux, Windows...
  - interfaces graphiques : gtk et gnome

# vim - en pratique *103.8*

Fonctionne par modes : commande, édition, visualisation.

## Raccourcis principaux

| | |
|---|---|
| Esc | sortir du mode courant |
| i | insérer (insert) |
| yy | copier une ligne (yank) |
| dd | coupe une ligne (delete) |
| p | coller (put) |
| :w | écrire dans le fichier (write) |
| :q | quitter vim (quit) |

## Pour aller plus loin

- 5dw $\longrightarrow$ efface 5 mots
- yf, $\longrightarrow$ copie le texte jusqu'à la prochaine virgule
- vimtutor pour s'entraîner aux manipulations

# vim - Fichiers de configuration *103.8*

## Fichiers de configuration

- `/etc/vim/vimrc` : global système
- `~/.vimrc` : personnel, ex. :

```
syntax on
set nu
```

## Fichiers auxiliaires

- `~/.viminfo` : historique commandes, tampons . . .

# Le réseau vu par l'utilisateur

# Architecture TCP/IP

*109.2*

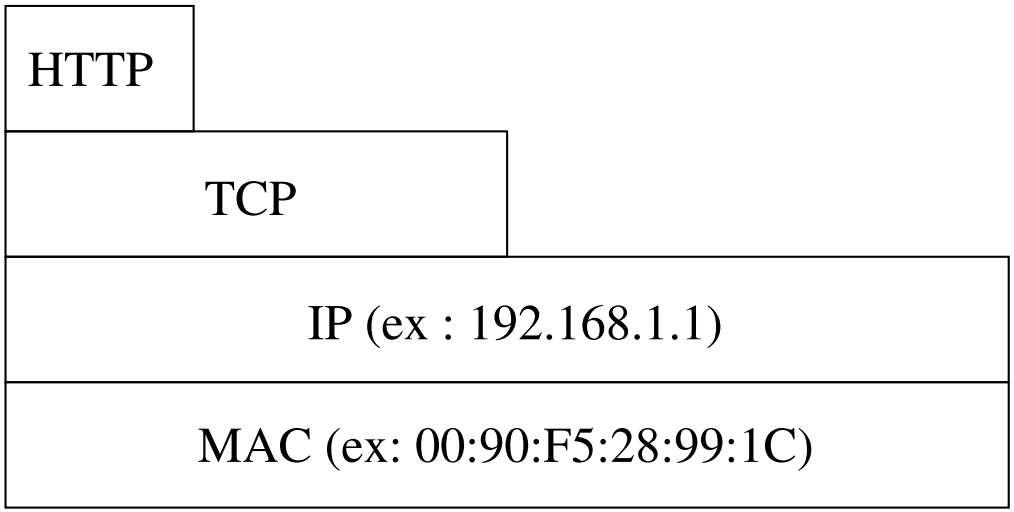
## Un modèle par couches

ernet réseau local Ethernet-MAC

IP l'adressage Internet, avec une double fonction

- identifiant unique de l'hôte sur le réseau (*identifier*)
- emplacement sur le réseau (topologie) (*locator*)

TCP le transport



# Commandes de diagnostic *109.2*

## ifconfig

- lo *(interface virtuelle boucle locale)*
- eth0 *(première interface ethernet)*

- l'adresse MAC : 6 octets
  ex. *HWaddr : 00 :90 :F5 :28 :99 :1C*
  Propre à la carte réseau
- l'adresse IPv4 : 4 octets, 32 bits
  ex. *inet addr : 192.168.1.1*
- l'adresse IPv6 : 16 octets, 128 bits
  ex. *inet6 : fe80 : :219 :66ff :fee9 :381/64*

# Commandes de diagnostic - 2 *109.2*

- `ping` Tester soi-même, un voisin, un absent, le réseau...
  - `ping -a -c5 192.168.1.1`
  - `ping -b 192.168.1.0`

- `traceroute` (champ TTL)
  affiche le chemin suivi par un paquet (tous les routeurs)

- `mtr [-t]` (my traceroute)
  combinaison des deux précédentes commandes

# Résolution de noms (DNS) *109.4*

## En local : /etc/hosts

Établit des correspondances *nom d'hôte* $\leftrightarrows$ *adresse IP*

## Domaine Name Server (DNS)

- Permet une équivalence entre nom et adresse IP.
  Ex : coriolan.silecs.info $\leftrightarrows$ 82.233.121.16
- Fonctionnement par hiérarchie de serveurs

## Clients DNS

- Client léger : `nslookup`
- Clients complets :
  - `dig` (dnsutils)
  - `host` (host)
- Sans oublier `ping` (/etc/hosts puis DNS)

# Exemple de service : SSH

# SSH : connexions sécurisées *110.3*

## La famille SSH

- `sshd` : le serveur
- Les clients essentiels
  - `ssh`, `slogin` : connexion interactive ou batch
  - `scp` : copie de fichiers via ssh
  - `sftp` : émulation ftp via ssh
- Les utilitaires
  - gérer les clés utilisateurs : `ssh-keygen`, `ssh-copy-id`
  - mémorisation des clés : `ssh-agent`, `ssh-add`

## Remarques

- conçu pour remplacer `rlogin`, `rcp`...
- X11 forwarding : ouverture à distance d'applis graphiques

# Clients SSH - 1 - shell distant *110.3*

- Shell interactif `slogin`
  - `slogin user@distant`
  - Variables d'environnement : `env | grep SSH` : `SSH_CLIENT`, `SSH_TTY`, `SSH_CONNECTIONS`
  - Qui est là ? commandes `who -l` et `w`
- X11 Forwarding
  - `slogin -X | -Y user@distant`
  - Variable d'environnement `DISPLAY=localhost:10.0`
- Shell non-interactif (commande à distance) `ssh`
  - `ssh user@distant /bin/ls`
  - `ssh user@distant "cat /etc/passwd | grep /home"`
  - `ssh user@distant "cat /etc/passwd" | grep /home`

# Clients SSH - 2 - transferts de fichiers *110.3*

- Copie distante `scp`
  - `scp user@distant:/home/user/.bashrc ./bashrc` *pull*
  - `scp ./fichier.txt user@distant:/home/user/Linux/` *push*
- Protocole SFTP (SSH File Transfer Protocol)
  - `sftp user@host:/path/to/dir` puis session interactive
  - `lftp` ou autres commandes multi-protocoles
  - graphique : gftp, filezilla, ou autres interfaces multi-protocoles
  - Note : SFTP $\neq$ FTPS (FTP over SSL) !
- TP pour aller plus loin
  - copie réseau en flux avec `tar` et `ssh`.
  - utilisation de `rsync` sur `ssh`.

# Cryptographie symétrique et asymétrique *110.3*

## Chiffrement symétrique

Une seule clé pour le chiffrage et le déchiffrage

## Chiffrement asymétrique

- Principe
  - une clé privée + une clé publique
  - une clé chiffre, l'autre déchiffre
  - secret : chiffrement avec la clé publique du destinataire
  - authentification : chiffrement avec la clé privée de l'expéditeur
  - une infrastructure de distribution des clés publiques (PKI)
- Diversité des clés SSH
  - clés d'hôtes (systématiques) vs clés d'utilisateur (optionnelles)
  - clés RSA, DSA, ECDSA : trois algorithmes différents
  - clé publique vs privée

# Authentification utilisateur SSH par biclé *110.3*

1. Création de la clé
   `ssh-keygen -t rsa -C "commentaire" [-f ma-clef]`
   → fichiers `ma-clef` et `ma-clef.pub` dans `/home/moi/.ssh/`

2. Installation de la clé publique
   `ssh-copy-id [-i ma-clef] [user@]distant`
   ou bien `scp` + `slogin` + `cat ... >> authorized_keys`

3. Connexion sans mot de passe
   `slogin [-i ~/.ssh/ma-clef] user@distant`

4. Pour aller plus loin : TP utilisation d'un agent SSH
   - 4.1 Protéger la clé existante par un mot de passe
   - 4.2 Comment ne pas retaper le mot de passe ?
   - 4.3 `ssh-agent` cf gnome-keyring...
   - 4.4 `ssh-add /.ssh/ma-clef` puis `ssh-add -l`

# Complément : configuration SSH *110.3*

Exemple de fichier /home/USER/.ssh/config

```
Host eniac
Hostname eniac.moore.upenn.edu.
IdentityFile /home/gallegre/.ssh/eniac_rsa
User gallegre
Port 22

Host hal
Hostname hal9000.nasa.gov.
ServerAliveInterval 30
ServerAliveCountMax 120
```

man 5 ssh_config

# Le chiffrement personnel avec GPG

# De PGP à GPG

*110.3*

- Historique
  - 1991 Phil Zimmermann crée PGP (Pretty Good Privacy)
  - 1997 OpenPGP (standard) devient la RFC 2440 (IETF)
  - 1999 Gnu Privacy Guard (GnuPG / GPG)
  - 2003- *G10code* fondée par Werner Koch, principal développeur

- Technologies utilisées par GPG
  - compression de fichiers
  - hachage (cryptographique)
  - cryptographie symétrique
  - cryptographie à clés publiques

# GPG : interfaces utilisateurs *110.3*

- Ligne de commande
  - `gpg` couteau-suisse complet, chiffrement et signature
  - `gpgv` uniquement vérification de signature
  - paquets nécessaires : `gnupg`, (`gnupg-agent`, `pinentry`)
- Interfaces graphiques générales
  - *Seahorse*, frontal Gnome
  - *KGPG*, frontal KDE
  - *GnuPG Shell*, interface multi-plateformes
- Intégrations multiples
  - mail : Kmail (KDE), Evolution (Gnome), plugin Enigmail (Thunderbird...)
  - messagerie instantanée (Gajim...)
  - éditeurs de texte : vim, emacs...

# En pratique : chiffrer un fichier *110.3*

- Créer une clé personnelle
  - `gpg --gen-key`, puis
  - `gpg (--armor) --output maclef.pub.asc --export`
- Chiffrer un fichier
  - clé publique : `gpg --recipient qui@mail.domain --encrypt msg.txt --output msg.gpg`
  - symétrique : `gpg --symmetric msg.txt --output msg-sym.gpg`
- Déchiffrer un fichier
  - clé privée : `gpg --decrypt msg.gpg --output message.txt`
  - symétrique : `gpg --decrypt msg-sym.gpg --output message.txt`

# En pratique : signer et authentifier un fichier *110.3*

- Signer un fichier : 3 modalités
  - `gpg --sign fichier` ⟶ `fichier.gpg` (binaire)
  - `gpg --clearsign fichier` ⟶ `fichier.asc` (texte)
  - `gpg --detach-sign fichier` ⟶ + `fichier.sig` (annexe)
- Extraire et vérifier un fichier signé
  - Vérifier : `gpg --verify fichier.[gpg|asc|sig]`
  - Extraire : `gpg --decrypt fichier.[gpg|asc]`

# GPG et *web of trust* *110.3*

- Importer une clé
  - fichier : `gpg --import pubkey.asc`
  - `gpg --listkeys | --list-secret-keys`
  - depuis un serveur : `gpg --search-keys <qui@mail.domain>` ou `gpg --recv-keys`
- Signer une clé
  - `gpg --sign-key <clef-id>`
  - cf `gpg --list-sigs` et `gpg --check-sigs`
- Envoyer la clé sur un serveur
  - `gpg --send-key <clef-id>`

# Autour de PGP / GPG

*110.3*

- Solutions d'authentification
  - OpenPGP Card (carte à puce)
  - poldi (pam + openpgp smartcard)
  - ...
- Bibliothèques
  - libgcrypt : bibliothèque principale de GPG
  - autres : bibliothèqu Gnu TLS

# Shells et scripts

# Panorama des shells - 1/2

*103.1*

- Référence
  cf Wikipedia, *Comparison of command shells*

- Shells historiques
  - `sh` original (1971), K. Thompson, Unix AT&T
    mode interactif seulement
  - Bourne shell (`sh`, 1977), Bell Labs, Unix v.7
    ajout des scripts
  - C shell (`csh`, 1978), Bill Joy, Unix BSD
    descendant du Thompson, syntaxe plus proche du C

# Panorama des shells - 2/2 *103.1*

- Shells courants
  - `tcsh` (1981, Tenex C shell), Ken Greer (Carnegie-Melon U.)
    par défaut sur FreeBSD
  - `ksh` (1982), Korn shell, Bell Labs : longtemps propriétaire
    évolutions ksh88 (POSIX), ksh93
  - `bash` (1987) Bourne Again Shell (projet GNU)
    par défaut sur GNU/Linux (GPL) ; v4.0 en février 2009
  - `zsh` (1990), Paul Falstad (Princeton U.)
    probablement le plus riche en fonctionnalités
- Shells restreints
  - `(d)ash`, Kenneth Almquist (sh compact)
  - `sash`, stand-alone shell (commandes internalisées)
- Changer de shell par défaut : `chsh`

# Les fonctionnalités du shell *103.1*

- Mode interactif
  - complétion automatique
  - historique des commandes, recherche... (readline)
  - alias
  - ...
- Fonctionnalités mixtes
  - boucles (for, while...)
  - enchaînements de commandes et valeurs de retour
  - fonctions
  - développement (globbing, variables...)
  - fichiers de configuration (`bashrc`...)
  - ...
- Mode script
  - gestion des paramètres ($1, $2...)
  - tests et conditions (if ... then ... else)
  - ...

# Documentation *103.1*

- Documentation électronique
  - `man bash`
  - `help help`
- Documentation libre
  - *Advanced Bash Scripting Guide*, M. Cooper (6.0, mars 2009)
    VF : *Guide avancé d'écriture des scripts Bash* (5.3)
  - *Bash Guide for Beginners*, M. Garrels (1.11, déc. 2008)
    VF : *Guide Bash du débutant* (avril 2007)
  - nombreux *tutoriels bash*, plus courts ou plus ciblés
- Livres
  - *Programmation shell sous Unix/Linux*, Ch. Deffaix Rémy, ENI
  - *Introduction aux scripts shell*, A. Robins, N. Beebe, O'Reilly

# Complétion

*105.1*

## Complétion standard

- noms de commandes
- entrées de répertoires (fichiers...)

# Complétion

*105.1*

## Complétion standard

- noms de commandes
- entrées de répertoires (fichiers...)

## Complétion étendue

- `shopt -s progcomp`
- `source /etc/bash_completion`
- sous-commandes
- options longues
- fichiers distants (ssh...)
- ...

# Readline - historique

*105.1*

- Commande `history`
  stockage dans `/home/user/.bash_history`
- édition accélérée
  - `C-a`, `C-e`, `C-←`, `C-→` : déplacements
- recherche et parcours de l'historique
  - `man readline` + `/etc/inputrc` : fichier de configuration
- développement de l'historique
  - indicateur d'événement : ex. !!, !123, !#
  - indicateur de mots : ex. 0, 1, ^, $
  - modificateurs : ex. ^chaine1^chaine2^

# Rappel : les alias

*105.1*

## Quelques exemples

- `alias ls="ls -color=auto"`
- `alias ll='ls -l'`
- `alias today='date +"%Y%m%d"'`
- `alias rm='rm -I'`

- `alias` seul : liste les alias définis
- `unalias (-a)` détruit un alias défini

## Pour aller plus loin : les fonctions

utilisation interactive : “alias à arguments”

# Fichiers de configuration *105.1*

- Fichiers personnels : dans `/home/user`
  - `.profile` ou `.bash_profile` : shells de login
  - `.bashrc` : autres shells
  - `.bash_logout` : nettoyage de session
- Fichiers globaux (système)
  - `/etc/profile`
  - `/etc/bash.bashrc`

# Fichiers de configuration *105.1*

- Fichiers personnels : dans `/home/user`
  - `.profile` ou `.bash_profile` : shells de login
  - `.bashrc` : autres shells
  - `.bash_logout` : nettoyage de session
- Fichiers globaux (système)
  - `/etc/profile`
  - `/etc/bash.bashrc`
- Contenu : les principaux réglages de session
  - Variables d'environnement : `export`
    p.ex. prompt : `$PS1, $PS2...`
  - alias (ex. `ll`, `la`, `dir`...)
  - fonctions
  - réglages du shell : `shopt`, `set`
  - inclusions : `source` ou `.`

# Configuration du shell *105.1*

- Variables d'environnement
  - aspect du prompt : `$PS1`, `$PS2` ...
  - `$GLOBIGNORE` ...

- `set -f/+f` ou `set -o/+o [option]`
  - ex. `set -f` ou `set -o noglob` désactive le globbing
  - ex. `set -x` ou `set -o xtrace` affiche chaque commande "développée"
  - + variable d'environnement `$SHELLOPTS`

- `shopt -s / -u` (set / unset)
  - env. 40 options booléennes : `shopt -p`
  - + 27 options "à la set" : `shopt -o -p`

# Bash - les "développements" *105.1*

## Sept types de développements successifs (*expansions*)

1. développement des accolades { } : factorisation
2. développement du tilde ~ ou ~user
3. développement des paramètres et variables
4. substitution de commande : `` `cmd` `` ou `$(cmd)`
5. développement arithmétique
6. découpage en mots
7. développement des chemins (globbing)

# Bash - les "développements" *105.1*

## Sept types de développements successifs (*expansions*)

1. développement des accolades { } : factorisation
2. développement du tilde ~ ou ~user
3. développement des paramètres et variables
4. substitution de commande : `` `cmd` `` ou `$(cmd)`
5. développement arithmétique
6. découpage en mots
7. développement des chemins (globbing)

## Rappel : les protections

- guillemets simples : protège de toute interprétation
- guillemets doubles : protège de tout sauf développement de variable
- antislah : protège totalement un caractère

# Diagnostic et enchaînements *105.2*

## Valeurs de retour et booléens du shell

- $? : valeur de retour du dernier processus terminé
- 0 = OK $\Longrightarrow$ vrai !
- >0 = erreur $\Longrightarrow$ faux !

# Diagnostic et enchaînements *105.2*

## Valeurs de retour et booléens du shell

- `$?` : valeur de retour du dernier processus terminé
- 0 = OK $\Longrightarrow$ vrai !
- >0 = erreur $\Longrightarrow$ faux !

## Enchaînement des commandes

- ET : `mkdir Toto && cd Toto`
- OU : `mkdir Titi || echo "erreur d'écriture"`
  $\rightarrow$ `mkdir Tutu && echo "OK" || echo "impossible"`
- enchaînement sans condition : `cmd1 ; cmd2`
- en parallèle + arrière-plan : `cmd1 & cmd2` && $\neq$ &

# Métaprogrammation *103.4*

- La commande `xargs`
  ex. `find /etc/ -size +100k | xargs wc -l`

- La substitution de commande
  ex. `wc -l $( find /etc -size +100k )`
  ou `wc -l ‘find /etc -size +100k‘` (backquotes)
  `echo "Vous êtes connecté sur $(uname -n)."`

- Remarque : la substitution de commande est plus générique (mais plus gourmande).

# Développement des paramètres et variables *105.1*

# Un exemple de script : bonjour.sh

*105.2*

```
#! /bin/sh

# Ce script salue l'utilisateur ($NAME)

echo "je suis $$"
echo "bonjour $NAME"
NAME="Alice"
echo "bonjour $NAME"
exit 0
```

**Rappel**

$$ : numéro du processus courant

# Scripts shells : modèles d'exécution *105.2*

- Deux modèles d'exécution
  - exécution `bash monscript.sh` shell fils, processus
  - inclusion `source script` inclus dans le shell interactif

# Scripts shells : modèles d'exécution *105.2*

- Deux modèles d'exécution
  - exécution `bash monscript.sh` shell fils, processus
  - inclusion `source script` inclus dans le shell interactif
- Rendre le script autonome
  - en en-tête de fichier : `#! /bin/sh`
  - le rendre exécutable : `chmod +x monscript.sh`
  - optionnellement, `PATH=$PATH:.`

# Scripts shells

*105.2*

- Paramètres positionnels
  - $0, $1, $2...
  - $# nombre d'arguments
  - "$*" la liste des arguments, sans tenir compte des blancs
  - "$@" la liste des arguments, en tenant compte des blancs
  - shift

# Panorama des structures de contrôle *105.2*

- Boucles
  - `for VAR in VALEURS ...; do ... done` — énumération
  - `for (( E1; E2; E3 )) ; do ... done` — numérique
  - `while ...; do ...; done` — tant-que
  - `until ...; do ...; done` — until
  - `select MOT in VALS ; do ... done` — menu (boucle interactive)
- Tests
  - `test` ou `[ ... ]` — test standard
  - `[[ ... ]]` — test avancé (Bash)
- Conditions
  - `if ... then ... fi`
  - `if ... then ... elif ... else ... fi`
  - `case MOT in MOTIF) ... esac`

# Boucle `for` *105.2*

- Usage interactif (ligne de commande)
  - `for VAR in un deux trois ; do echo $VAR ; done`
  - `for F in *.txt ; do wc -l $F ; done`
- Usage en script

```
for ARG in $@
    do
        echo $ARG
        ...
    done
```

- Variante `select` (en script)

# while et until

*105.2*

# TP scripts 1 - boucles

*105.2*

- Avec `find`, trouver le nombre de répertoires dans la partition `/`, sans changer de système de fichiers (`-xdev`).
- Même question, mais pour chacun des 7 types : f, d, l, p, s, c, b.
- Calculer et afficher également le total (évaluation arithmétique, cf. ci-après)
- Affiner l'affichage pour obtenir une présentation du genre

```
123456  fichiers
 12987  répertoires
 26789  liens
```

# Évaluation arithmétique - 1/2

*105.1*

- Variable "standard" (chaîne)

```
Var=0
Var=$Var+1      # "0+1"
Var=Var+1       # "Var+1"
```

- Typage numérique (entier)

```
declare -i Ent=5
Ent=Ent+1       # "6"
a=Ent+1         # "Ent+1"
```

# Évaluation arithmétique - 2/2

*105.1*

- Évaluation arithmétique forcée

```
i=0
i=$(( i+1 ))     # standard
(( i=$i+1 ))     # extensions bash...
(( i=i+1 ))
let i=i+1
let i=$i+1
```

- Bonus : formatage numérique
  - `printf 'James Bond %03d, No %02d' 7 3`

# Écrire une boucle numérique *105.2*

- La commande `seq`
  - `for i in $(seq 0 2 8) ; do echo $i ; done`
  - `seq 8` 1 à 8
  - `seq 0 8` 0 à 8
  - `seq 0 2 8` 0, 2, 4, 6, 8
  - Bonus `seq -f '%03.0f' 0 2 12` format virgule flottante ( !)
- Bash, mode standard

  `while [ $i -lt 9 ] ; do echo $i ; let i=i+1 ; done`
- Bash, mode arithmétique
  1. `while ((i<9)) ; do echo $i ; done`
  2. `for ((i=0; i<9; i+=2)); do echo $i; done`

## TP Table de multiplication 5x5

1. tout à la suite, de type I x J = IJ
2. dans un tableau (matrice) I en lignes, J en colonnes

# Tableaux en Bash 1/2 : index numériques *105.2*

- Déclaration et définition

```
declare -a Tab
Tab[0]=zero
Tab=(zero un deux trois quatre cinq)
declare -p Tab  # Vérification
```

- Utilisation des tableaux

```
echo ${Tab[0]}     #... de 0 à 5
echo ${Tab[-1]}    #... de -1 à -6
echo ${Tab[*]}     # une liste
echo ${Tab[*]:2:3} # une tranche de tableau
```

- Travaux pratiques
  1. Compter à rebours, de "cinq" à "zéro"
  2. Afficher une table d'addition en toutes lettres

# Tableaux en Bash 2/2 : tableaux associatifs *105.2*

- Déclaration et définition

```
declare -A Asso
Asso[couleur]=rouge
Asso=([couleur]=rouge [outil]=marteau [animal]=lion)
declare -p Asso
```

- Utilisation

```
echo ${Asso[couleur]}
echo ${Asso[*]}
echo ${!Asso[*]}

for KEY in ${!Asso[*]}; do
  echo "$KEY => ${Asso[$KEY]}"; done
```

# Exemples de tests

*105.2*

- Tests standard `[ ... ]`- exemples
- Tests avancés `[[ ... ]]`- exemples

# La condition `if`

*105.2*

- En ligne de commande
  - `if mkdir Rep ; then echo Fait ; else echo Erreur ; fi`
  - cf `mkdir Rep && echo "Fait" || echo "Erreur"`

# La comparaison case

*105.2*

*105.2*

# TP scripts 2 : disable/enable

*105.2*

1. Ecrire un script `disable.sh` qui
   - prend en argument un nom de fichier
   - le renomme en lui ajoutant le suffixe `.OFF`

2. Ecrire le script inverse, `enable.sh`, qui supprime le suffixe `.OFF`. Il doit accepter en argument les deux variantes `fichier` et `fichier.OFF`.

3. Transformer les deux scripts en un seul `xable.sh`, qui prend une option (-d ou -e) pour indiquer le sens de l'opération.

# TP scripts 3 : gestion des liens *105.2*

1. Ecrire un script `rmlink.sh` qui
   - prend en argument une entrée de répertoire
   - la supprime si c'est un lien symbolique
   - retourne un message d'erreur sinon

2. Variante `rmbrlink.sh` : supprime seulement les liens cassés

3. Variante : transforme `rmbrlink.sh` en option (-b) de `rmlink.sh`

4. Ecrire un script `rmhlink.sh` qui supprime l'entrée de répertoire si c'est un fichier régulier avec (ref>1), autrement dit si c'est un lien dur.

5. Ecrire une fonction `ireadlink` qui affiche une résolution de lien symbolique avec intermédiaires : ex. `/usr/bin/rsh -> /etc/alternatives/rsh -> /usr/bin/ssh`.

# Les fonctions shell

*105.1*

- Déclarer une fonction : la commande `function`

```
function lprman
    {
    man -t $1 > $1.ps
    lpr $1.ps
    }
```

- Lister les fonctions déclarées
  - `declare -F` liste les déclarations
  - `declare -f [nomfonction]` liste les contenus

- Appeler une fonction

```
nomfonction <param1> <param2> ...
```

# TP : tout ensemble !

*105.2*

- TP : Trouver la place occupée par les fichiers de chaque type MIME dans le répertoire utilisateur.
  Astuce : utiliser la commande `file -i` pour les types MIME.
  Variante : remplace le type MIME par l'extension.

# Redirections étendues : HERE... *105.1*

- HERE-Documents <<

```
$ wall <<FIN
> ETEIGNEZ VOS MACHINES
> coupure electrique imminente
> --- l'equipe systeme
> FIN
```

- HERE-Strings <<<

ex. `cut -b cut -b 1,3-5,16- <<< "internationalisation"`

# sed, expressions rationnelles

# Expressions rationnelles (ou régulières) *103.7*

- un “outil” commun à de nombreux utilitaires
  `grep`, `sed`, `awk`, `vim`...
- Deux formes (malheureusement !)
  - forme “basique” interne à chaque commande
  - forme “étendue” standardisée (POSIX.2)
  - `man 7 regex`

# sed - Stream EDitor *103.7*

## Contexte

- écrit par Lee McMahon en 1973/1974 (Bell Labs),
- dérivé de l'éditeur monoligne `ed`
- applique une série de règles d'édition de texte...
- à chaque ligne d'un fichier, successivement
- reconnaît deux types d'expressions régulières

# sed - Stream EDitor *103.7*

## Contexte

- écrit par Lee McMahon en 1973/1974 (Bell Labs),
- dérivé de l'éditeur monoligne `ed`
- applique une série de règles d'édition de texte...
- à chaque ligne d'un fichier, successivement
- reconnaît deux types d'expressions régulières

## Quelques exemples

- `sed -e "s/Old/New/g" f-in > f-out`
- `sed -e '/^ *$/d' f-in`

# awk

# AWK - un filtre-langage

- Origines...
  - langage défini par Aho, Weinberger, Kernighan en 1977
  - standard POSIX, NAWK (New AWK), courant 1980s
  - *The AWK Programming Language*, 1988
  - plusieurs interpréteurs libres (orig-awk, gawk, mawk...) ou pas
  - une syntaxe intermédiaire entre C et le shell
  - à l'origine de Perl

# AWK - un filtre-langage

- Origines...
  - langage défini par Aho, Weinberger, Kernighan en 1977
  - standard POSIX, NAWK (New AWK), courant 1980s
  - *The AWK Programming Language*, 1988
  - plusieurs interpréteurs libres (orig-awk, gawk, mawk...) ou pas
  - une syntaxe intermédiaire entre C et le shell
  - à l'origine de Perl
- Caractéristiques principales
  - conçu pour analyser un fichier (ou flux) texte divisé en champs
  - tableaux associatifs
  - expressions régulières
  - bien adapté à des scripts unilignes (comme `sed`)

# AWK - un filtre-langage

- Origines...
  - langage défini par Aho, Weinberger, Kernighan en 1977
  - standard POSIX, NAWK (New AWK), courant 1980s
  - *The AWK Programming Language*, 1988
  - plusieurs interpréteurs libres (orig-awk, gawk, mawk...) ou pas
  - une syntaxe intermédiaire entre C et le shell
  - à l'origine de Perl
- Caractéristiques principales
  - conçu pour analyser un fichier (ou flux) texte divisé en champs
  - tableaux associatifs
  - expressions régulières
  - bien adapté à des scripts unilignes (comme `sed`)
- Particularités des implémentations

| | |
|---|---|
| mawk | performances et efficacité (précompilé) |
| gawk | richesse et documentation (i18n) |
| xgawk | extensions XML, PostgreSQL |
| awka | comilateur AWK -> C |

# AWK - invocation et structure

- Invocation de awk
  - `awk -f script.awk fichier`
  - `awk 'code AWK' fichier`
  - exécutable commençant par `#! /bin/awk -f`

# AWK - invocation et structure

- Invocation de awk
  - `awk -f script.awk fichier`
  - `awk 'code AWK' fichier`
  - exécutable commençant par `#! /bin/awk -f`

- Structure d'un script

```
motif  { action }
...
```

  - motif : sélecteur de lignes ou BEGIN ou END
  - action : instruction de type procédural

# AWK - invocation et structure

- Invocation de awk
  - `awk -f script.awk fichier`
  - `awk 'code AWK' fichier`
  - exécutable commençant par `#! /bin/awk -f`

- Structure d'un script

```
motif  { action }
...
```

  - motif : sélecteur de lignes ou BEGIN ou END
  - action : instruction de type procédural

- Quelques exemples

```
awk 'BEGIN { print "Bonjour !" }'
awk 'length($0) > 60' /etc/passwd
awk 'NR % 2 ==0' /etc/passwd
awk 'BEGIN {FS=":"}  NR % 2==0 {print $1}' /etc/passwd
```

# AWK - TP avec find

- Utilisation basique
  Trouver la place occupée par l'ensemble des fichiers de plus de 1 Mo dans le répertoire utilisateur (on peut varier les critères...).

- Utilisation avancée : tableaux associatifs
  Trouver la place occupée par les fichiers de chaque type d'extension (txt, sh, ...) dans le répertoire utilisateur
  Astuce : utiliser la directive `split` pour les extensions.