Initiation à l'Utilisation d'UNIX (et dérivés)

Fil conducteur

- B-A-BA Unix en mode Texte
  - (disponible aussi avec Cygwin)
  - Interpréteur(s) de commande
  - Arborescence Unix
  - Commandes de navigation dans l'arborescence
  - Aide en ligne
  - Contrôle des Processus

Connexion au Système

- A partir d'une console texte Unix
  - Sous Linux : Ctrl-alt-F*N*
    - *Bascule vers la console texte*

```
login: odalle
password: xxxxxx
```

  - Début de la session
    - Apparition du prompt de l'Interpréteur de Commandes
    - ...
  - Fin de session
    - 'exit' ou 'logout' ou <ctrl>-D
- Changer son mot de passe ...
  - Soit `passwd` (mot de passe conservé localement)
  - Soit `yppasswd` (base centralisée « NIS »)

Connexion au système (2)

- A partir de la bannière de connexion graphique Unix
  - (éventuellement) Choisir un type de session
    - Précise le *look and feel* de l'environnement de travail
      - Environnement intégré : CDE (Sun), Gnome ou KDE (Linux, Sun)
      - Session par défaut : configuration propre à l'utilisateur
      - Session *Failsafe* : minimum, en cas d'échec des précédentes
      - Session précédente : même environnement que la fois précédente
  - saisir login + password
    - Début session = chargement environnement graphique
    - Ouvrir fenêtre terminal
      - Sur Fedora / Gnome : menu appli/outils système/...
      - Lance un interpréteur de commandes
      - Possibilité d'ouvrir simultanément plusieurs fenêtres
        - Exécution indépendante (pseudo-parallélisme)

Interpreteur de Commande

- *shell* : coquille (autour du Système)
- Fonctionnalités
  - Message d'invitation (prompt)
  - Attend la saisie d'une commande
    - Commandes internes
    - Commanes externes (chargement et exécution de programmes)
    - Historique des commandes précédentes
  - Redirections et tubes
  - Structures de contrôles (boucles, tests, fonctions, ...)
  - Mécanisme d'alias
  - Variables d'environnement
  - Gestion de processus

Différents *shell* possibles

- Bourne shell (historiquement, le 1er)
  - Toujours très utilisé par les *scripts* internes du système
- C-shell (csh, tcsh)
  - syntaxe différente
- Evolutions du Bourne shell
  - ksh (Korn shell)
  - bash (Bourne Again shell)
    - le shell du projet GNU
  - zsh
    - Evolution prétendument "ultime" du bourne shell
  - Améliorations : historique, complétion, ...
- Possibilité de (demander à) changer le shell par défaut
  - lancé automatiquement au début de session

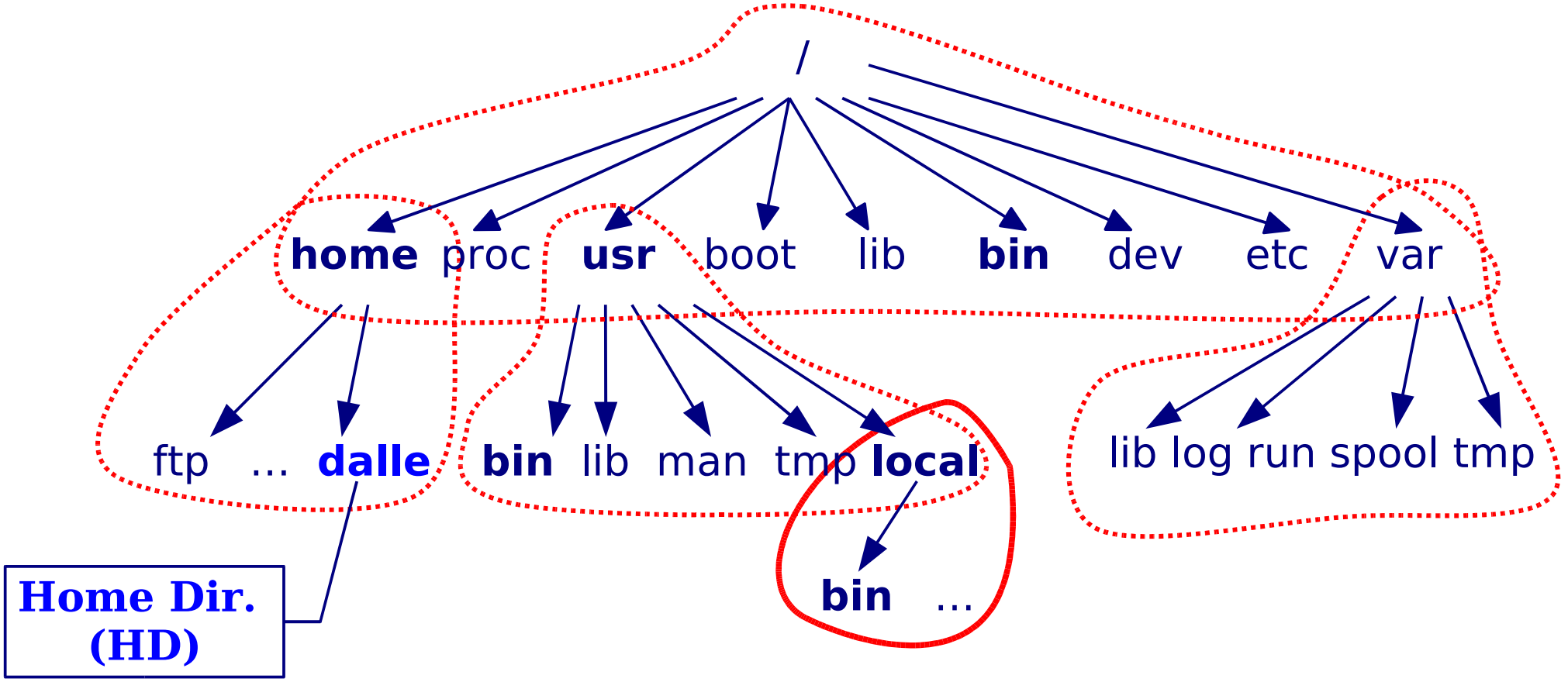
Le Système de Gestion de Fichiers

Structure arborescente : répertoires + fichiers

- Fichiers
  - fichiers de données normaux
  - fichiers spéciaux
    - Périphériques, tubes, liens, **répertoires,** ...
- Racine Unique (contrairement à DOS/Windows)
  - Les différents éléments (partitions, disquettes, CD) peuvent être attachés n'importe où dans l'arbre global
  - Racine nommée "/"
    - Caractère "/" indique le franchissement d'un répertoire
- Droits d'accès spécifiques associés à chaque fichier et répertoire
  - 3 types d'autorisations : lecture, écriture, exécution
  - 3 niveaux d'autorisation : propriétaire, groupe, autres

Organisation de l'Arborescence de Fichiers

- Hiérarchie en voie de normalisation
  - Norme FHS (Filesystem Hierarchy Standard)
  - www.pathname.com/fhs/



Visualiser l'occupation disque

- Commande df : espace libre sur chaque partition

```
[odalle@petiote]~% df -k
Sys. de fich.         1K-blocs       Occupé Disponible Capacité Monté sur
/dev/hda5              26667864  18494580   6818608  74% /
/dev/hda3                101105     14398     81486  16% /boot
none                     387768         0    387768   0% /dev/shm
/dev/hda2              19051792  17104544   1947248  90% /misc/dos_data
grignotte:/home        74730400  54154600  16779640  77% /u/grignotte/home
mascotte:/0           238395496  18088024 208197680   8% /u/mascotte/0
droitdvote:/home      238142568  61647616 164398008  28% /u/droitdvote/home
```

- Commande du : espace occupé par une arborescence
  - Option -s : somme
    - Attention aux permissions ...

```
[odalle@petiote]~% du -ks .
9403696 .
[odalle@petiote]~% du -ksh .
9,0G    .
```

- Commande mount : liste des points de montage

Quelques Eléments Remarquables de l'Arborescence

- Current Working Directory (CWD)
  - position courante du shell dans l'arborescence
  - répertoire de travail par défaut
    - quand une commande à besoin de lire ou écrire un fichier, elle utilise par défaut le CWD
  - nom = '.'
- Home Directory (HD)
  - Répertoire personnel de l'utilisateur (maison)
  - Début de session : CWD <-- HD
  - nom (raccourci) : '~' (ou '$HOME')
- Répertoire parent
  - Un niveau plus près vers la racine
  - nom = '..'

Navigation dans l'Arborescence

- Définitions
  - Chemin **relatif** : par rapport au CWD
    - Ex: toto/titi (eqv à ./toto/titi), ../../tutu/tata
  - Chemin **absolu** : par rapport à la racine
    - Ex: /usr/bin/date, /home/odalle/toto/titi
- Commandes utiles
  - **cd** (change directory) : change valeur de CWD
    - Ex: cd toto/titi
  - **pwd** (print working directory) : affiche CWD
  - **ls** (list) : liste le contenu (fichiers et rep) de CWD
    - Ex: ls, ls -a, ls -al, ls -hl, ...

option(s): modifient le comportement de la commande

Quelques Commandes sur les Fichiers et Répertoires

- **mkdir** : création d'un répertoire
  - mkdir toto titi
- **rmdir** : destruction d'un répertoire
  - rmdir toto
- **touch** : modification date d'un fichier
  - Crée un fichier vide
  - Ex: touch tutu titi/toto
- **rm** : destruction d'un fichier
  - rm -i tutu
- **cp** <src ...> <dst> : copie d'un (ou plrs) fichier
  - cp titi/toto tata
- **mv** <src...> <dst> : déplacement (ou renommage)

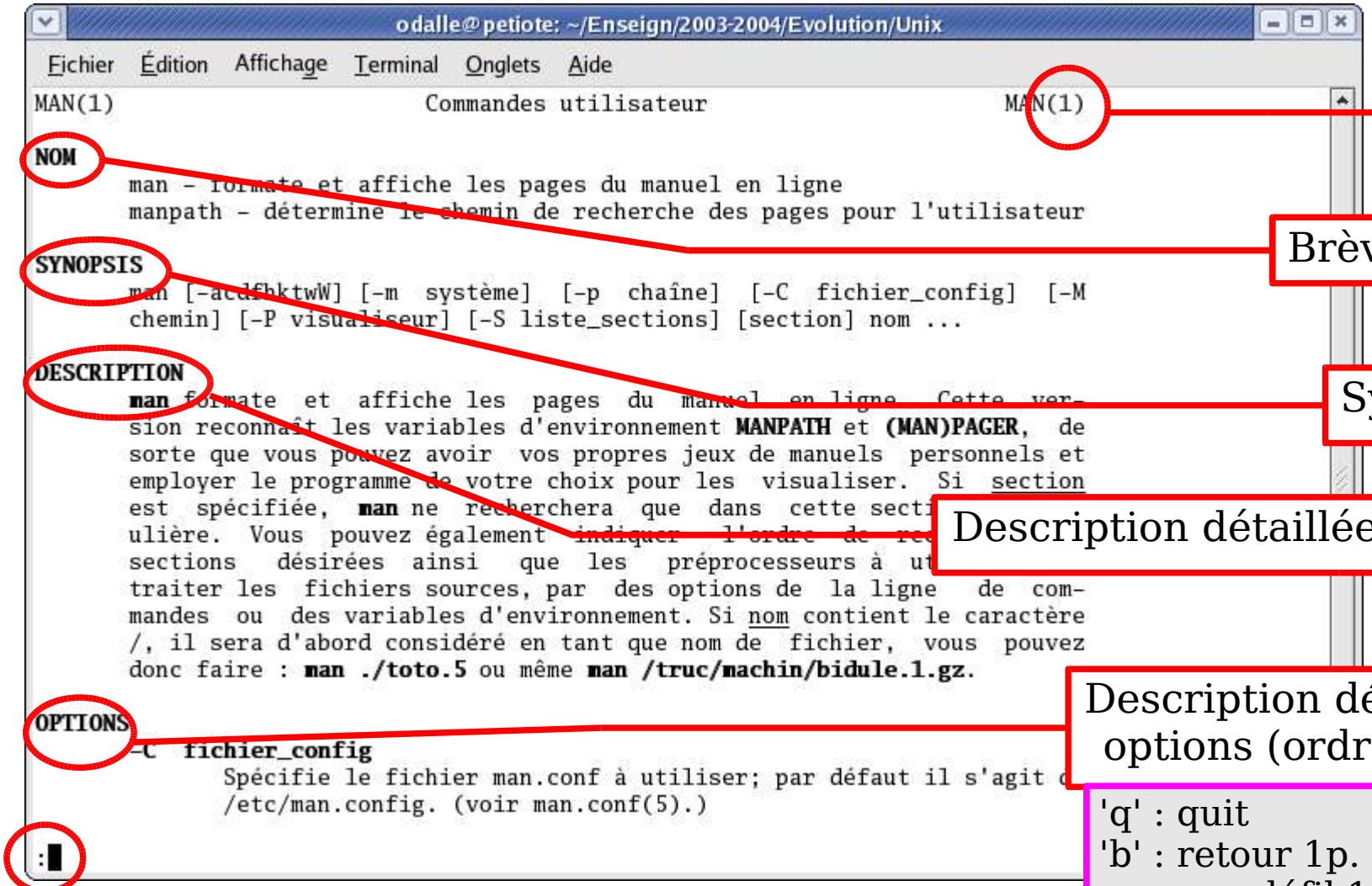
Options et Paramètres des Commandes

- Syntaxe usuelle
  - `commande [options] [parametres]`
- Options: permettent de changer ce comportement
  - Conventions :
    - Option formée d'une lettre précédée d'un simple tiret
      - Ex : ls -a, ls -l, ls -h
      - Possibilité de grouper les options : ls -alh
    - Option formée d'un (ou plrs) mots précédée d'un double tiret
      - Ex: ls --full-time, ls --help
- Paramètres (parfois facultatifs)
  - Désignent les données sur lesquelles appliquer la commande
  - Si plusieurs : séparation par un (ou plrs) espaces
    - Ex : ls -al -h –full-time /usr/bin /home/odalle

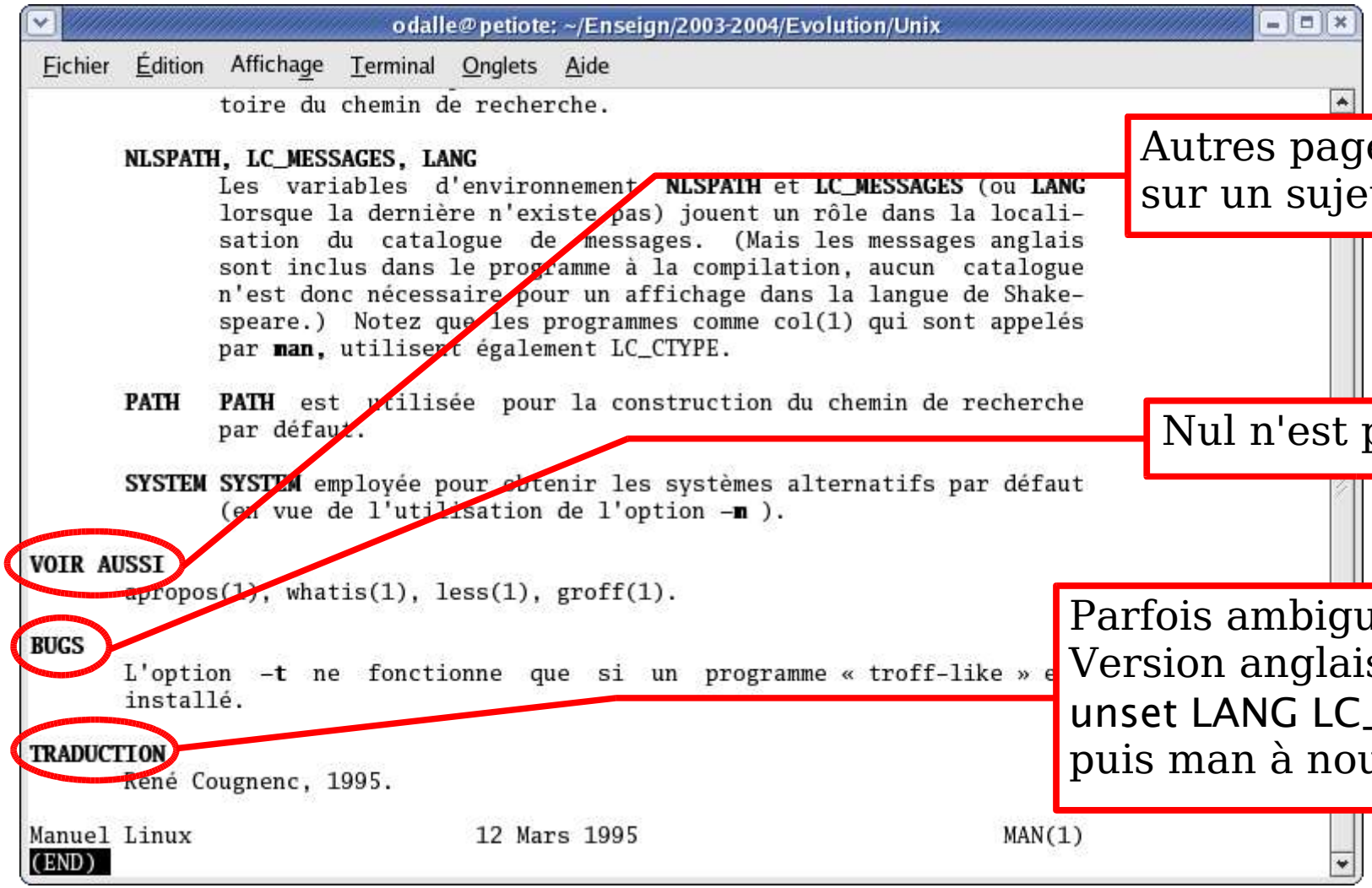
Documentation sur les Commandes

- manuel en ligne : commande 'man'
  - Ex : man ls
  - Manuel décomposé en une petite 10aine de sections
    1. Manuel des commandes utilisateur
    2. Manuel des appels systèmes en C
    3. Manuel des fonctions de bibliothèque du langage C
    - Ex :
      - man man , man 1 intro, man 2 intro, ...
      - man 1 kill / man 2 kill
- Auto-documentation des commandes
  - Ex : ls --help
- Outil 'info'
  - Documentation hypertexte
  - Page parfois plus à jour que la page de manuel

Anatomie d'une Page de Manuel (le haut)



Anatomie d'une Page de Manuel (le bas)



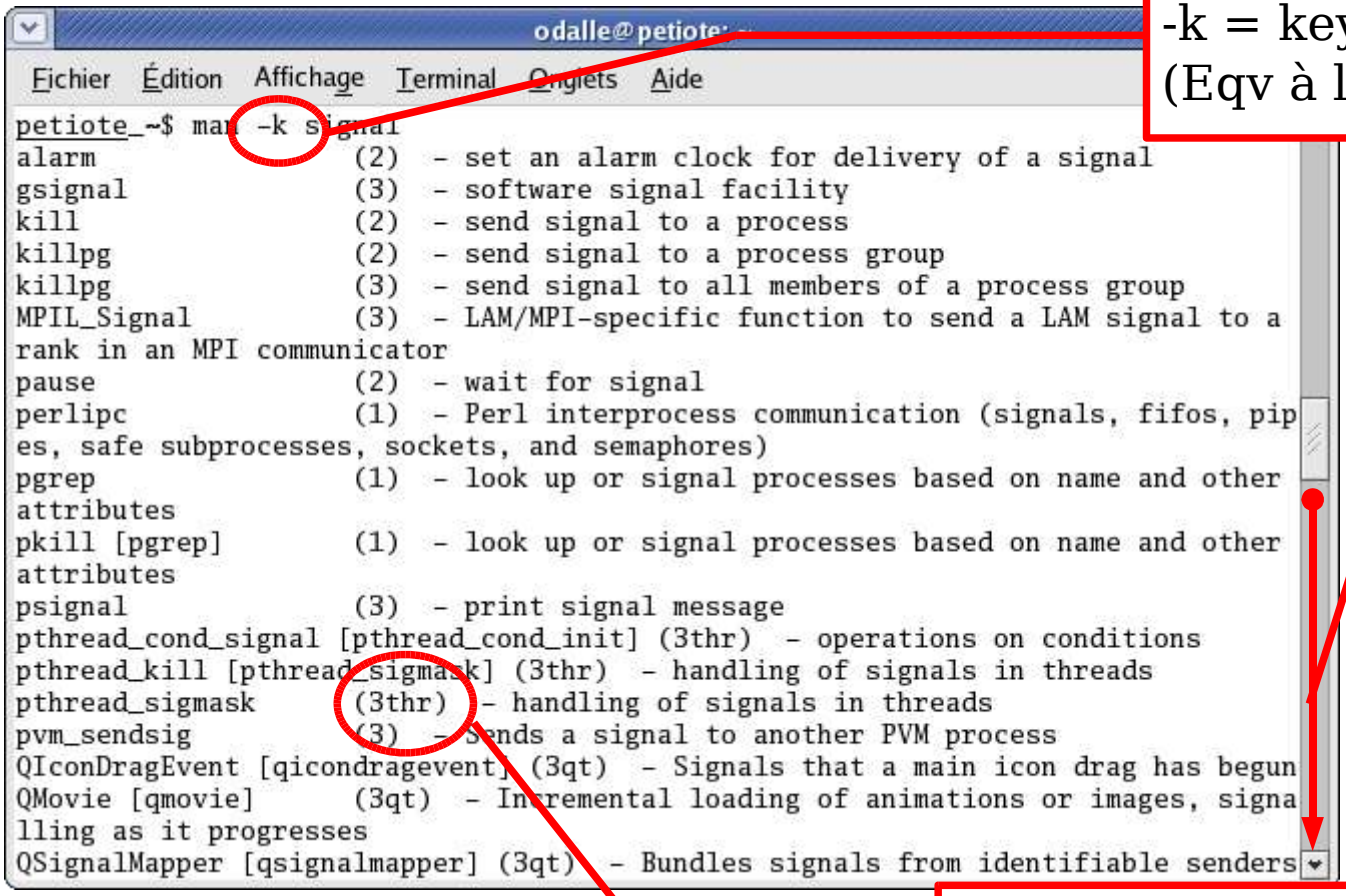
Autres pages intéressantes sur un sujet voisin

Nul n'est parfait ...

Parfois ambiguës ou confuses
Version anglaise (défaut) :
unset LANG LC_ALL
puis man à nouveau...

Mode Apropos de Man (apropos ou man –k)

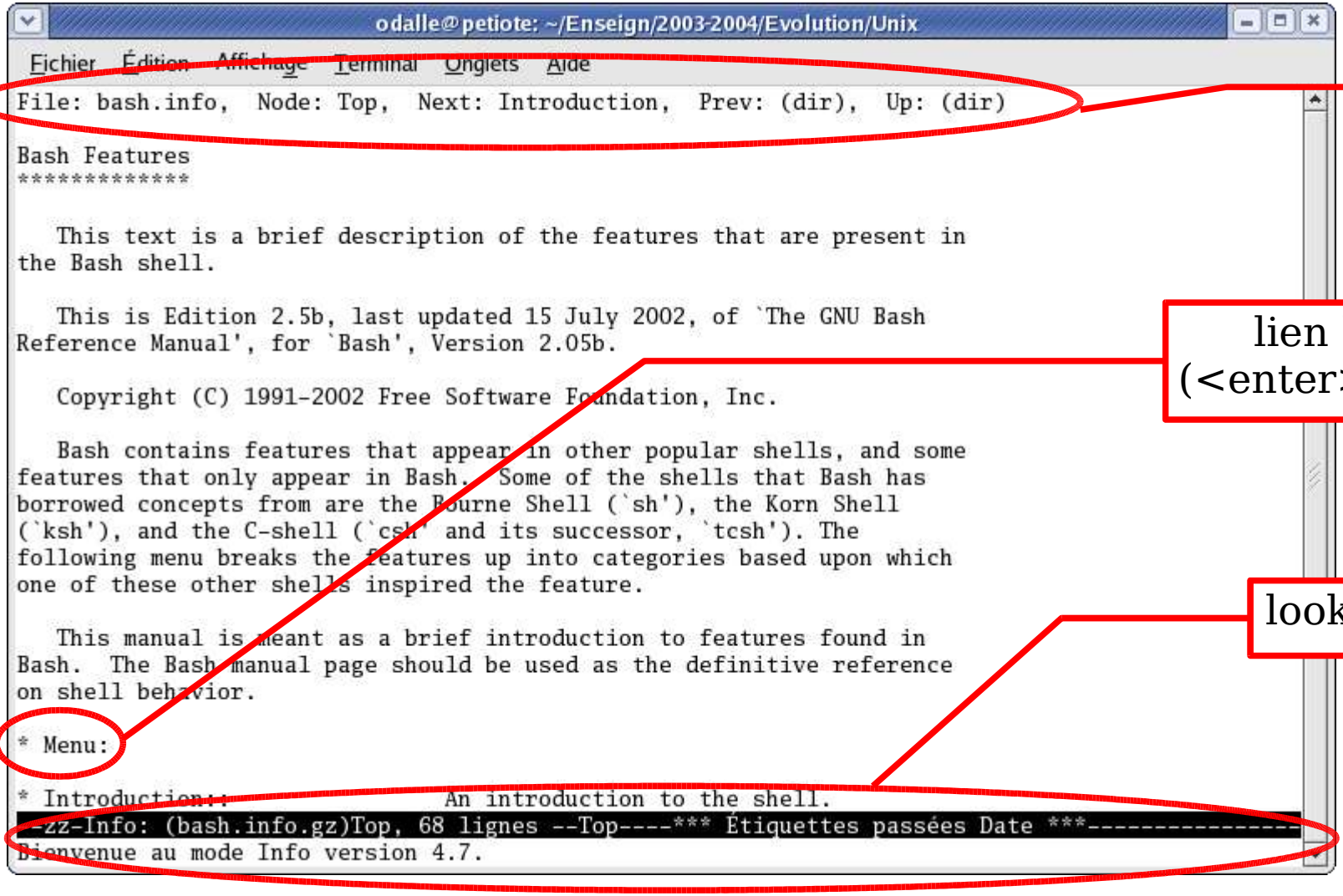
Mode de recherche par mot clef :
-k = keyword
(Eqv à la commande apropos)



Résultats souvent très nombreux !

Certaines bibliothèques ont des documentations en ligne très fournies : elles se distinguent en ajoutant un suffixe spécifique au numéro de section

Exemple d'Utilisation de GNU info (ici : info bash)



Navigation :
'n' : next
'p' : prev
'u' : up

lien hypertexte
(<enter> pour suivre)

look&feel à la Emacs

Caractères Génériques

- Certains caractères ont une signification spéciale pour le shell
  - * : 0 ou plusieurs caractères quelconques
    - t*o : désigne un fichier qui commence par t et se finit par o
  - ? : 1 (et un seul) caractère quelconque
    - t?t? : désigne toto ou titi ou ttta, ...
  - \ : enlève la signification spéciale du caractère qui suit
    - t*to : désigne t*to
- Certaines constructions ont une signification spéciale
  - [a-z] : un caractère entre a et z (a ou b ou c ... ou z)
  - [aefgij] : un caractère (et un seul) parmi a,e,f,g,i,j
    - t[oau]t[oau] : toto, ou totu, ou tata, ou tato, ...
  - {mot1,mot2} : soit mot1, soit mot2

A Propos du *Globbing*...

- Globbing = action de remplacer les caractères spéciaux du shell
  - Fonctionnement :
    1. Le shell substitue les caractères spéciaux en fonction des fichiers existants
    2. Le shell exécute la commande résultante
  - Exemple : si ./ contient : toto titi truc bidule
    - 'ls t*' se **transforme** en 'ls toto titi'
      - le shell exécute 'ls toto titi'
    - 'ls *' se transforme en 'ls toto titi truc bidule'
  - Piège classique :
    - touch t[oai]t[oai] ne créera pas toti, tota, ... !

Quelques Pièges à Eviter avec le Globbing

- Attn : le shell substitue d'abord TOUT ce qu'il peut !
  - Le commande exécutée est le résultat de la substitution
  - Exemple de catastrophe :
    - le CWD contient les fichiers : toto.a, titi.a, tutu.a
    - Que produit la commande suivante ?
      - mv *.a *.o
    - Réponse :
      1. substitution : mv *.a *.o => mv toto.a titi.a tutu.a
      2. exécution :
         - → déplace toto.a en tutu.a (écrasé)
         - → déplace titi.a en tutu.a (écrasé une 2e fois !)
      3. Résulat : toto.a et tutu.a sont (définitivement) perdus !!
    - Comment obtenir la copie tous les .a en .o alors ?
      - Pas de solution simple : il faut faire une boucle ...

Noms de Fichiers Bizarres ...

- Tous les caractères sauf '\0' et '/' sont autorisés dans les noms de fichiers
  - """";;;; est un nom de fichier valide
  - --help aussi ...
- Les fichiers dont le nom commence par . sont cachés
  - Fichiers de configuration, généralement dans $HOME
  - Ces fichiers sont normalement ignorés lors du « globbing »
    - Ex : cp * ~/bkup ne copie pas les fichiers cachés
    - Solution : cp * .?* ~/bkup
  - Ces fichiers sont « visibles » avec l'option -a (-A) de ls

Entrées/Sorties d'un Processus Unix

- Concept fondamental de flux (stream)
  - Un processus consomme des données (standard input)
  - Un processus produit un résultat (standard output)
  - Un processus produit des erreurs (standard errors)

- Comportement par défaut
  - *stdin* = console (clavier)
  - *stdout* = console (affichage terminal)
  - *stderr* = console (affichage terminal)

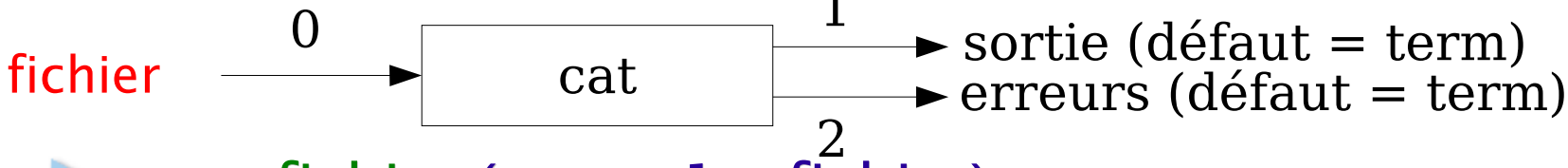
Exemple : La Commande cat

- cat [fichier1 ...]
  - Sans paramètre : envoie stdin inchangé vers stdout
    - Utile ? Dans certains cas oui !
      - Quand on a besoin d'un processus « entre » deux redirections
        - Le shell n'exécute que des commandes, pas des redirections seules !
        - Ex: cat > fichier
          - cat lit l'entrée standard (saisie clavier)
          - les caractères saisis sont envoyés dans fichier
          - La saisie s'arrête en fin de fichier : ^D en début de ligne
  - Avec paramètre(s)
    - stdin ignoré
    - lit à la suite (concatène) le(s) fichier(s) donné(s) en param.

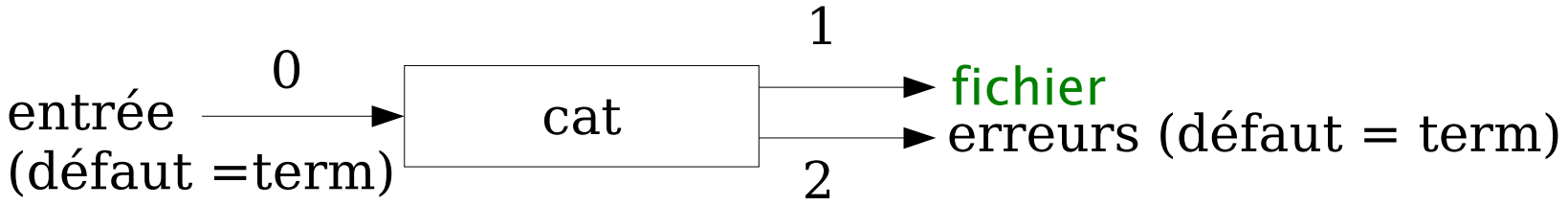
Redirections des Entrées / Sorties

Redirection = débrancher la connexion au terminal pour brancher l'E/S sur autre chose

- un fichier
  - cat < fichier (ou cat 0< fichier)



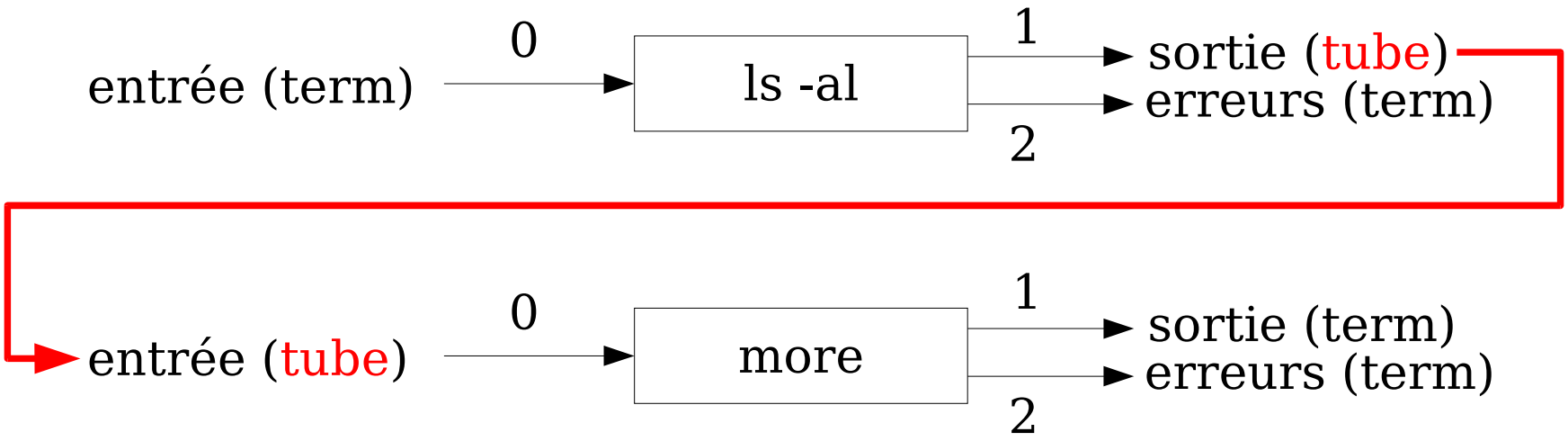
  - cat > fichier (ou cat 1> fichier)



  - cat 2> fichier : redirige les erreurs vers fichier
  - Possibilité de redirections simultanées
    - cat < toto > titi 2> tutu

Tubes (*pipes*)

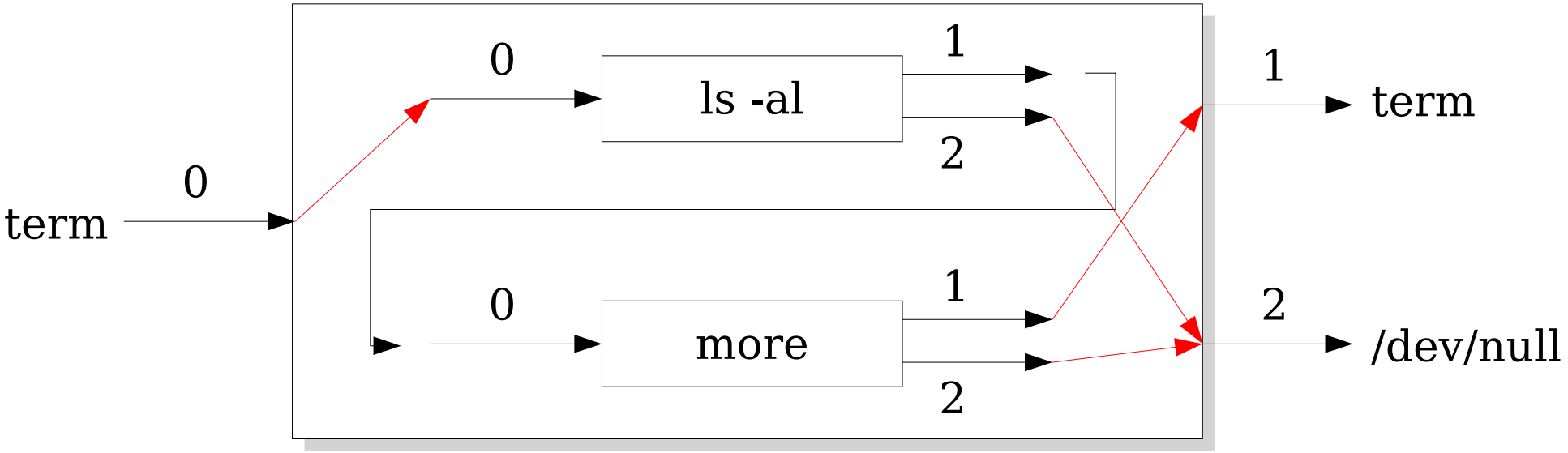
- Rediriger la sortie (standard) d'un processus vers l'entrée d'un autre processus
  - ls -al | more



  - ls -al 2> /dev/null | more
    - redirige les erreurs du processus ls vers null (trou noir)
  - ls -al | more 2> /dev/null
    - redirige les erreurs du processus more vers null

Sous Processus

- Créer un processus autour d'une chaîne de processus
  - (une | chaine | de | processus)
  - Principe des poupées russes : les E/S du processus externe peuvent être redirigées
  - Ex : (ls -al | more) 2> /dev/null



Encore des Redirections...

- Redirection en mode « append »
  - cmd >> fichier
  - Idem redirection simple mais en mode append
    - ls -al . > courant_et_parent
    - ls -al .. >> courant_et_parent
      - ajoute le résultat à la suite dans le fichier courant_et_parent
      - Rem : idem à (ls -al . ; ls -al ..) > courant_et_parent
- De nombreuses autres formes existent...
  - Exemple : Redirection « here document »
    - Lecture sur stdin jusqu'à l'apparition d'un mot clef
    - cat > fic_result << FIN
  - Duplication de descripteur (eqv dup2() en langage C)
  - ... On en reparle plus tard ...

Quelques Commandes Utiles avec les Redirections

- grep motif [fichier1 ...]
  - recherche motif dans le(s) fichier(s) donné(s) en arg. (ou stdin à défaut)
- tee [fichier1 ...]
  - duplique les données (idée du T en plomberie) dans les fichiers donnés en paramètre et sur stdout
- tr 'xyz' 'ijk'
  - transforme les lettres du paquet 'xyz' lues sur stdin en celles du paquet 'ijk' : x->i , y->j , z->k
  - options pour supprimer totalement des lettres, ou seulement les répétitions
- more et less
  - les « pagers » usuels

Permissions (protection) des Fichiers (et Répertoires)

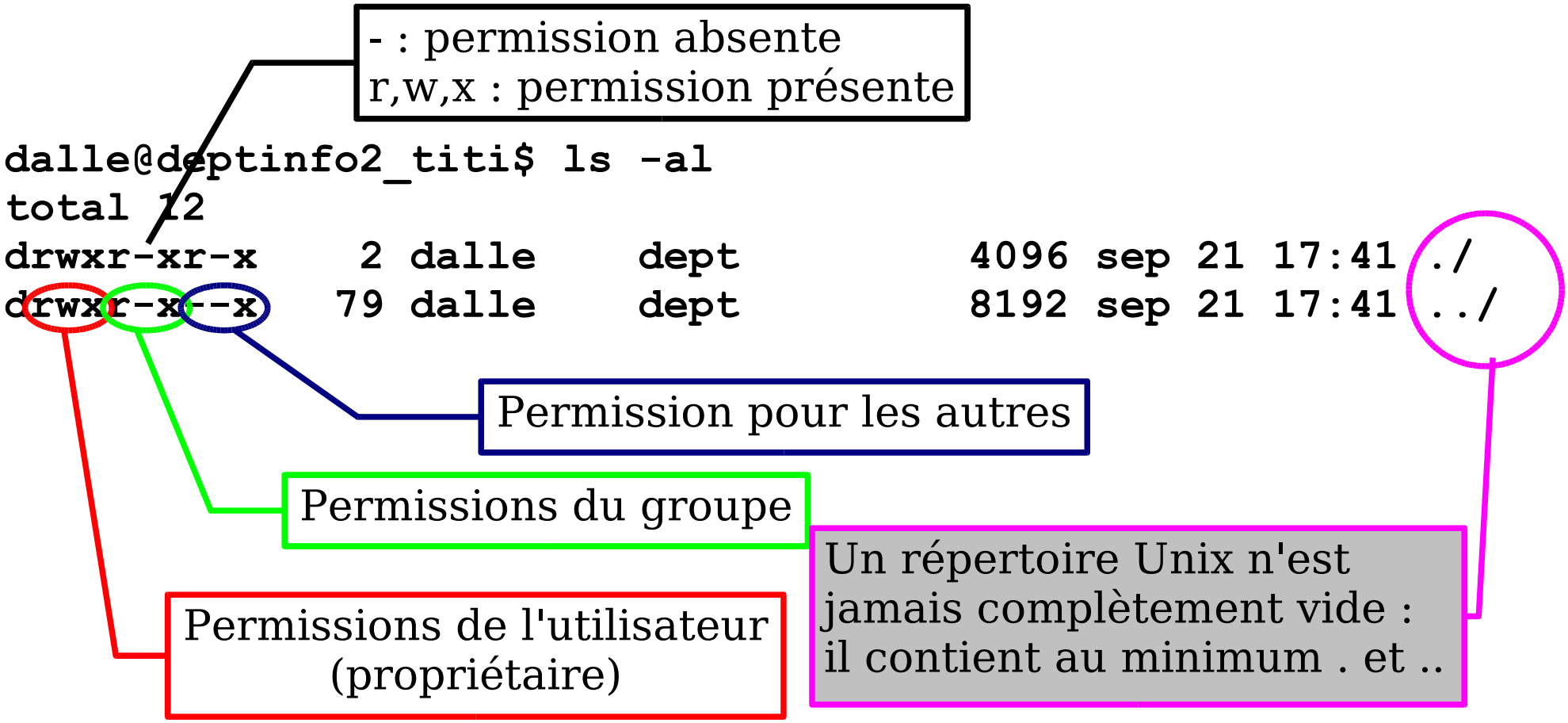
- Pour chaque fichier, Unix distingue 3 classes d'utilisateurs
  - Le propriétaire : *user*
  - Les membres du groupe : *group*
  - Le reste du monde : *other*
- Pour chaque classe, Unix distingue 3 types de permissions
  - Lecture : 'r' (comme Read)
  - Ecriture : 'w' (comme Write)
  - Exécution : 'x' (comme eXecute)
- Chacune de ces permissions est indépendante
  - 3 classes X 3 types = 9 permissions par fichier
  - Chaque permission est vraie (donnée) ou fausse (retirée)

Dans le cas d'un répertoire:
permission de « traverser »

Affichage des Permissions

ls -l <fichier> :

- Permissions courantes = première colonne
- Ordre : ugo (user, groupe puis other)



Modification des Permissions

- Commande chmod
  - chmod <qui><+-=><perm> <fichier(s)>
    - qui : u, g, o, a (all = u+g+o)
    - perm : r,w,x
- Exemples :
  - **`chmod go+x toto titi`**
    - Ajouter (+) perm exécution (x) au groupe et aux autres (go) pour les fichiers toto et titi
  - **`chmod a-rx tutu`**
    - retirer (-) permissions lecture et exécution (rx) à toutes les catégories (a) pour le fichier tutu
  - **`chmod u=rx titi`**
    - Fixer exactement (=) les permissions pour le propriétaire (u) du fichier titi à rx

Fixer les permissions en octal

- Méthode « pour les pros »
  - un bit associé à chaque permission
    - 0 si la permission est absente
    - 1 si la permission est donnée
    - r+w+x = 3 bits => un chiffre octal (val entre 0 et 7)
      - x (exécution) vaut 0 ou 1
      - w (écriture) vaut 0 ou 2
      - r (lecture) vaut 0 ou 4
      - Permission = x*1 + w*2 + r*4
  - Exemple
    - 5 = 1*1 + 0*2 + 1*4 => exécution + lecture (rx)
  - 3 classes (user, group, other)
    - Une permission octale = 3 chiffres en octal
    - Ex : **chmod 755 toto**

Pourquoi s'embêter avec des permissions en octal ?

- Seules certaines combinaisons sont usuelles
  - Répertoires : 755, 750 ou 700
    - 7 : rwx pour le propriétaire
    - 5 : rx pour les autres (et/ou le groupe)
    - 0 : Aucun droit pour les autres (et/ou le groupe)
    - Rappel : x est indispensable pour « traverser » un rep.
  - Fichier : 644, 640 ou 600
- Certaines commandes ne comprennent que l'octal
  - umask : retrait systématique de certaines permissions
    - Ex : tester `umask 002` puis `touch titi`
  - find : recherche (profonde) de fichiers
    - Ex : `find ./ -perm -100`
      - Fichier dont la permission vaut au moins 100 = exécutables pour le propriétaire

`find` : le moteur de recherche ...

- Commande pour rechercher des fichiers
  - Recherche récursive (profonde)
    - En partant d'un ou plusieurs répertoires d'origine
  - Recherche possible selon de nombreux critères
    - nom de fichier (expression de globbing)
    - taille
    - permission
    - date d'accès/modification
    - type (répertoire, lien, ...)
  - Possibilité de combiner plusieurs critères
    - ET, OU, NOT
  - Possibilité d'exécuter une action systématique sur les fichier trouvés
    - Par défaut : afficher le chemin vers le fichier

`find` : syntaxe et exemples

- find origine(s) [critères] [action]
  - origines : le ou les répertoires à partir desquels explorer
  - quelques critères (consulter man pour les autres)
    - -name <glob> : recherche par nom
    - -type [fdl...] : recherche par type (fichier, répertoire, lien, ...)
    - -size <+/-taille> recherche par taille (+ signifie supérieur à)
    - ...
  - action :
    - -print : afficher (défaut)
    - -exec cmd \{\} \; (\{\} désigne le fichier trouvé)
- Exemples :
  - **`find ./ -type f -a -size +100k`**
  - **`find $HOME/tmp -name '*.gz' -exec gunzip \{\} \;`**

Liste des Processus

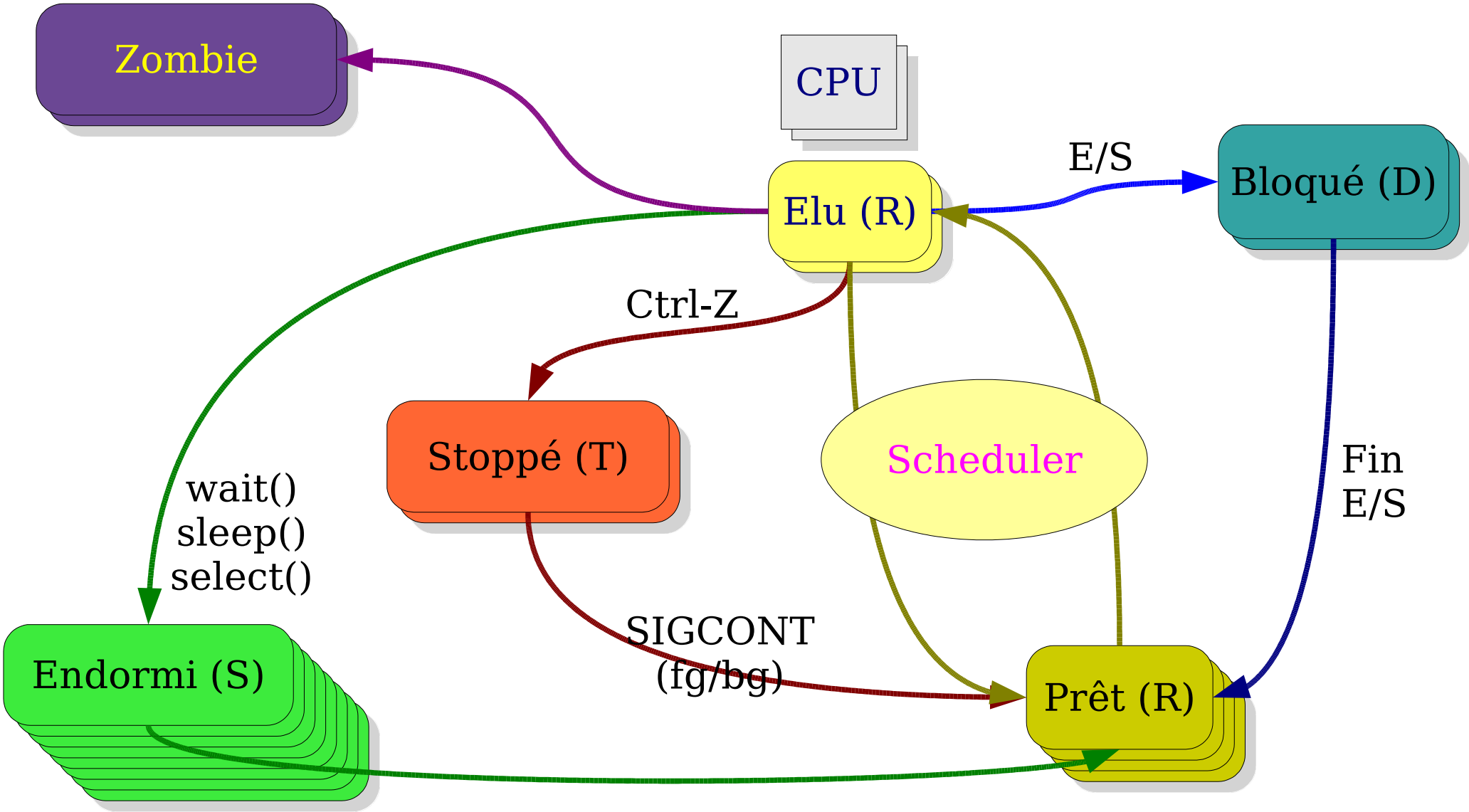
- Commande 'ps'
  - ps (sans option) : liste des processus attachés à un terminal
  - ps –x : liste de tous les processus de l'utilisateur
  - ps –aux (ou ps –edf si Unix System V) : liste de tous les processus du système
- Chaque processus a un identifiant **unique** (pid)

```
toto_v2$ ps
  PID TTY          TIME CMD
 1710 pts/1    00:00:00 zsh
 1735 pts/1    00:06:09 soffice.bin
 1755 pts/1    00:00:00 soffice.bin
 1756 pts/1    00:00:00 soffice.bin
```

Etats d'un Processus

- Champ STAT de la commande `ps -x`
  - `R` : Prêt à s'exécuter ou en cours d'exécution
  - `S` : Endormi en attente interruptible (non exclusive) d'un événement
  - `D` : Endormi en attente non interruptible (exclusive) d'un évènement (typiquement une E/S)
  - `Z` : Zombie
  - `T` : Arrêté ou Tracé
  - `(W` : Pas de page résidente)
  - `(N` : Gentillesse positive)
    - Le processus est lancé avec une priorité plus faible
    - nice -n <quantité> <cmd>
  - + Un état supplémentaire implicite : processus élu

Graphe des Transitions entre Etats d'un Processus



Changer l'Etat d'un Processus

- <ctrl-z> : stoppe le processus courant (-> état T)
  - En réalité : envoi d'un signal
- fg : redémarre le dernier processus stoppé **en avant plan**
  - Le processus récupère les E/S du terminal
- bg : redémarre le dernier processus stoppé **en arrière plan**
  - libère le terminal
  - Affichage du prompt : le shell est prêt pour une nouvelle commande
- <cmd> & : lance une commande directement en arrière plan
  - Equivalent à : <cmd> puis <ctrl-z> puis bg

Surveiller les processus

- La commande ps avec certaines options (à tester !)
  - ps aux / ps -edf : liste détaillée de tous les processus
  - ps -auwxf : liste détaillée avec généalogie et détail des arguments
- La commande top (hélas pas toujours disponible)

```
top - 15:19:51 up  4:03,  3 users,  load average: 0.03, 0.13, 0.20
Tasks: 107 total,   1 running, 106 sleeping,   0 stopped,   0 zombie
Cpu(s):  2.6% us,  1.0% sy,  1.3% ni, 94.7% id,  0.0% wa,  0.3% hi,  0.0% si
Mem:    775536k total,   766772k used,     8764k free,   234256k buffers
Swap:  2096440k total,        0k used,  2096440k free,   101136k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3139 root      15   0 83248  33m  51m S  2.0  4.4   3:28.55 X
 3376 odalle    20   5  233m  92m 141m S  1.7 12.2   5:50.79 soffice.bin
 3351 odalle    15   0 25908  13m  20m S  0.3  1.7   0:04.66 gnome-terminal
 5342 odalle    17   0  1996  788 1576 R  0.3  0.1   0:00.16 top
    1 root      16   0  1996  532 1320 S  0.0  0.1   0:00.94 init
    2 root      34  19     0    0    0 S  0.0  0.0   0:00.26 ksoftirqd/0
    3 root       5 -10     0    0    0 S  0.0  0.0   0:00.09 events/0
    4 root       5 -10     0    0    0 S  0.0  0.0   0:00.01 khelper
```

Processus vs. Jobs

- Processus : Tout programme en cours d'exécution
  - La liste est loooongue ...
    - Autres utilisateurs
    - Processus de service (personne en particulier)
  - Problème 1 : besoin d'interaction sur un petit nombre
    - Ses propres processus
    - Généralement ceux qu'on vient de créer
  - Problème 2 : processus fortement liés
    - Cas des pipelines : ls -al | tr -s ' ' | more
- Solution : notion de job/group
  - Un groupe (=job) créé pour chaque pipeline
  - Numérotation des jobs interne au shell
    - Numérotation indépendante
      - préfixe %

Signaux

- Signal = mini-message
  - Un nombre entier
  - Lors de la réception du signal, le processus exécute une action prédéfinie
    - terminaison simple
    - terminaison avec copie mémoire (core dump)
    - stop
    - continuer
    - Un traitement spécifique prévu par le programme
- De 32 à 64 signaux différents
  - Chacun correspond à un événement particulier
  - Chacun a un nom précis ('man 7 signal' sur linux)
  - Certains sont déclenchés par une frappe clavier
    - <ctrl-c>, <ctrl-z>, ...

Envoyer un Signal

- Processus en avant plan
  - Certaines combinaisons de touches envoient un signal
    - <ctrl-c> => SIGINT
    - <ctrl-z> => SIGTSTP => SIGSTOP
    - <ctrl-\> => SIGQUIT
    - <ctrl-t> => SIGINFO
- Signal vers autres processus : **kill** <pid> ou kill %<job>
  - tue le processus de numéro <pid> ou le job <job>
  - En réalité : envoie signal de terminaison (SIGTERM = 15)
    - Pb : SIGTERM peut être inopérant
    - Possibilité de demander l'envoi d'un autre signal
      - SIGKILL (signal n° 9) : imparable
      - Ex: kill -9 3245 (ou kill -KILL 3245)
      - kill -CONT : redémarrage après <ctrl-z>

Variables

- Le shell permet la définition de variables
  - mavar="toto"
- Le caractère '$' permet de substituer la valeur d'une variable
  - Ex: echo $mavar => echo toto
- Certaines variables sont prédéfinies (ou supposées)
  - HOME : répertoire HD de l'utilisateur
  - PATH : répertoires où se trouvent des exécutables
  - USER : identifiant de l'utilisateur (login)
  - PRINTER : nom de l'imprimante par défaut
  - MAIL : Zone de dépôt du courriel de l'utilisateur
  - PS1 : message affiché par le prompt du shell
  - ...

Variables d'Environnement

- Chaque processus possède des variables **d'environnement**
  - transformer une variable (norm.) en variable d'env. :
    - export variable (export variable=valeur)
  - lister la valeur des variables d'environnement
    - env
- Les processus sont créés par **clonage**
  - Chaque nouveau processus récupère la copie des variables d'environnement de son père
    - Seulement les variables d'environnement !
  - Le shell est le père des commandes lancées
    - Chaque commande lancée récupère les variables d'environnement définies dans le processus shell père

Configuration du shell

Lorsqu'un processus shell est créé (login, nouveau terminal, ...) :

- Le shell lit des fichiers de commande prédéfinis
  - fichier $HOME/.profile
  - shell bash : $HOME/.bash_profile $HOME/.bash_rc
  - shell zsh : $HOME/.zlogin, $HOME/.zshrc
- Comme la lecture est systématique
  - Bon endroit pour placer des commandes de configuration
    - Configuration de la variable d'environement PATH
    - Configuration de variables diverses (PRINTER, ...)
    - Exécution de commandes d'intialisation
- Format d'un fichier de commande ?
  - Simple fichier texte : *shell-script*