

La ligne de commande

Jean-Yves Didier
UFR S&T – Université d'Evry

La ligne de commande

Rôle :

Permettre à l'utilisateur de lancer des commandes ;

Appelé aussi « *shell* ». Exemple :

MsDOS ;

cmd.exe sous Windows XP ;

Les shells Unix :

Bourne shell (bash) ;

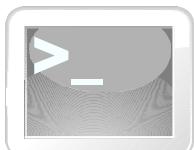
C shell (csh, tcsh) ;

Korn shell (ksh) ;

Z shell (zsh), etc.

Chacun possède ses spécificités et sa syntaxe :

Nous verrons essentiellement bash !



La ligne de commande

Le shell est également un interpréteur :

Peut lire des fichiers contenant des commandes ;

Ils sont nommés **fichiers scripts** ou **fichiers batch**.

Il est possible de **programmer** avec !

Nous verrons quel est la syntaxe de ce langage.

Exemples d'utilisation :

Démarrage d'Unix (par le biais de scripts) ;

Tâches programmées dans le temps (cron , at) ;



La ligne de commande

Fonctionnement :

Boucle infinie :

Affichage du prompt (invite de commande) : \$

Lecture d'une commande ;

Analyse syntaxique (découpage en mots) ;

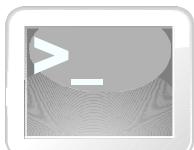
Substitution (si nécessaire) ;

Lancement de la commande en lui passant les paramètres ;

Fin de boucle

Le *shell* est sensible à la casse :

'A' et 'a' sont considérés comme différents



La ligne de commande

Lancer des commandes à distance :

Il est possible de se connecter à des machines à distance pour exécuter des commandes :

rsh (*remote shell*) ;

telnet (protocole telnet, utilisé pour lancer des commandes) ;

ssh (invite de commande sécurisée – transmission cryptée) ;

Enfin, sortir de l'invite de commande :

exit ;

logout ;

Ctrl D

La ligne de commande

Les scripts reposent sur la richesse des commandes et programme proposés :

Gestion des processus ;

Gestion des fichiers ;

Gestion des utilisateurs ;

Mais aussi des programmes jouant le rôle de filtre :

tr, grep, paste, tee, join, diff, look, sed,
cut, uniq, sort, comm, cmp, head, tail, wc,
nl, etc.

Ainsi que des commandes fondamentales :

echo, man (la plus importante de toutes !),



Les processus

Définition : un processus est un programme chargé en mémoire et est en cours d'exécution

Les processus possèdent :

- un code de retour :
 - Si 0, alors le programme a fini normalement ;
 - Sinon, le programme a fini prématurément.
- des entrées/sorties par défaut :
 - sortie standard (affichée à l'écran par défaut) ;
 - sortie d'erreur (affichée à l'écran par défaut) ;
 - entrée standard (entrée clavier par défaut).



Les processus

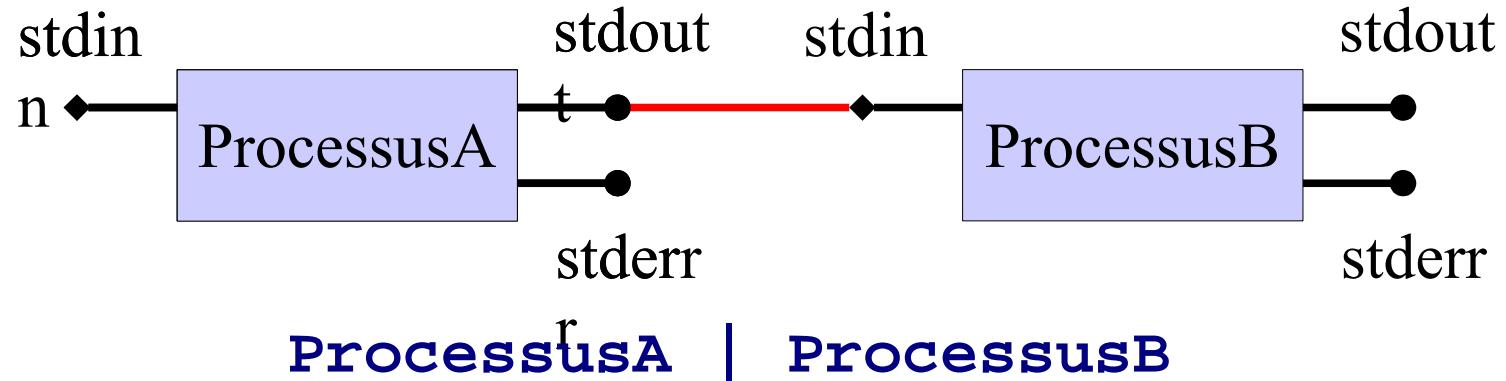
L'architecture *pipe-filter* :

- Permet de connecter les sorties des processus aux entrées des autres ;
- Mécanisme de composition des commandes ;
- Permet de créer des commandes complexes ;
- *Pipe* : canal de communication inter-processus,
- *Filter* : processus prenant des données en entrée, les traitant, donnant des résultats en sortie.

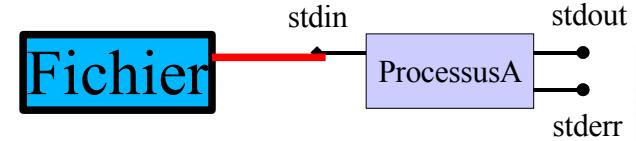
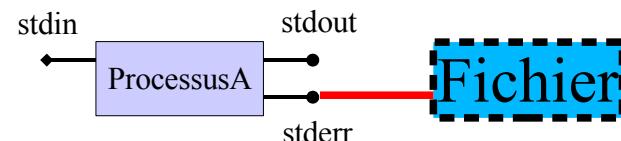
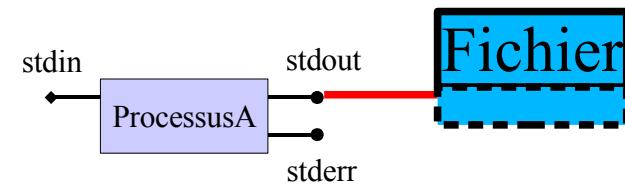
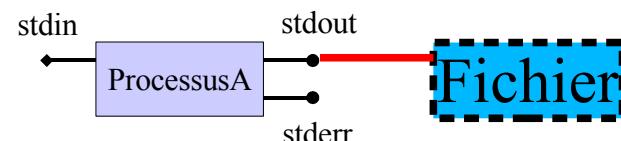


Les processus

Interconnecter les entrées sorties :



Rediriger vers des fichiers :



Les processus

Commandes (programmes) associées à la gestion des processus :

- ps , pstree ; liste les processus
- kill ; envoie un signal (utile pour les interrompre) ;
- etc.

Les filtres

Syntaxe des commandes :

nom [paramètre_optionnel] paramètre_requis

tr : substitution de caractères :

tr [option] ensemble1 [ensemble2]

Ex : cat toto | tr [a-Z] [n-za-mN-ZA-M]

grep : sélection de ligne :

grep [option] motif [fichier]

Ex : grep didier /etc/passwd

paste : fusion de lignes d'un fichier

paste [option] [fichier]



Les filtres

tee : duplique sa sortie sur stdout et un fichier

`tee [option] [fichier]`

Ex : `cat toto | tee titi`

join : fusion de fichiers par recouplement

`join [option] fichier1 fichier2`

Ex : `join -t ':' -1 4 -2 3 /etc/passwd /etc/group`

diff : compare des fichiers ligne par ligne

`diff [option] fichiers ...`

Ex : `diff toto tata`



Les filtres

look : lignes commençant par une expression

Ces dernières doivent être triées :

```
look chaîne [fichier]
```

sed : « *stream editor* », fait de la substitution « à la volée » en utilisant des expressions régulières

Possède son propre langage de script

```
sed [option] [script] [fichier]
```

cut : extraction de champs dans les lignes

```
cut [option] [fichier]
```

Ex : `cut -d ':' -f 1 /etc/passwd`



Les filtres

uniq : suppression des doublons

```
uniq [option] [fichier_entree [fichier_sortie]]
```

sort : tri des lignes d'un fichier

```
sort [option] [fichier]
```

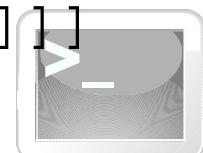
Ex : sort /etc/passwd

comm : comparaison de fichiers triés (cf diff):

```
comm [option] fichier1 fichier2
```

cmp : comparaison de fichiers binaires

```
cmp [option] fichier1 [fichier2 [decall1 [decall2]]]
```



Les filtres

head : affiche le début du fichier :

head [option] [fichier]

Ex : head -c 20 /var/log/messages

tail : affiche la fin d'un fichier

tail [option] [fichier]

Ex : tail -n 20 /var/log/messages

wc : (*word count*) statistiques sur un fichier :

wc [option] [fichier]

Ex : wc -l /var/log/messages



Les filtres

nl : comptage de lignes non vides :

nl [option] [fichier]

Ex : nl /etc/rc.d/rc.S

echo : écrire quelquechose sur l'écran :

echo [option] [chaine]

Ex : echo ''toto''

man : page de manuel (la plus importante !)

Décrit les commandes, fichiers de configuration, etc.

Ex : man man



Le système de fichiers

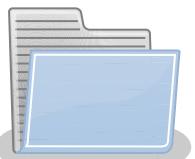
Au début était le périphérique de stockage ...

Cartes perforées, Bandes, Disquette, Disque dur,
CD/DVD-Rom, clés USB.

Chacun a un fonctionnement propre, une organisation des données différentes.

Cas des disques durs actuels :

Choisi car d'usage courant.



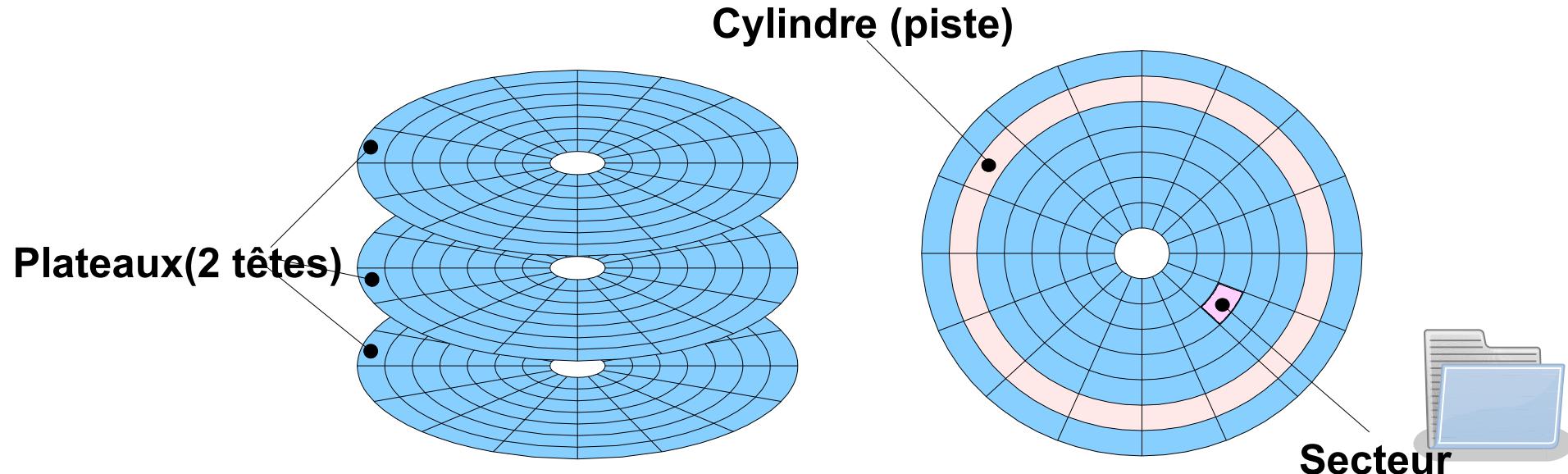
Le système de fichiers

Disques durs : géométrie

Composé de plateaux, pistes et secteurs ;

1 secteur : bloc de généralement 512 octets de données utiles ;

Repéré par n° tête, cylindre, secteur



Le système de fichiers

Comment écrire sur un disque :

Repérer un emplacement vide ;

Écrire un bloc de 512 octets de données ;

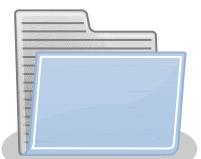
Recommencer tant que tout n'est pas écrit.

Comment repérer un emplacement vide ?

Ne pas parcourir tout le disque !!!

Utiliser une structure décrivant l'occupation du disque :
le *SGF* ou *Système de Gestion des Fichiers*.

Un disque dur peut accueillir plusieurs SGF en parallèle : dans ce cas, il est *partitionné*.



Le système de fichiers

Les partitions :

Correspond aux disques logiques sous Windows.

Peuvent être de deux types :

Principales : 4 au maximum ;

Étendues : Partitions à l'intérieur d'une partition principale.

Stockées dans une table de partitions :

Située dans le 1er secteur du disque dur : le MBR

MBR : Master Boot Record

Contient la table de partitions primaires ;

Le boot loader (système d'amorçage si présent) ;

Le tout sur ... 512 octets !!!



Le système de fichiers

Partitions nécessaires pour Linux :

2 au minimum, 3 dans la pratique :

Une partition de données avec un SGF ;

Une partition de swap (mémoire virtuelle) ;

Une partition de données utilisateur.

Une partition est associée à un système de gestion de fichiers (SGF) :

Réalisée lors de l'opération de formattage !

Une zone sur le disque est réservée pour décrire la structure du SGF.



Le système de fichiers

Le SGF :

Organise et optimise l'implantation des données sur le disque ;

Permet de retrouver et corriger des zones du disque endommagées ;

Offre à l'utilisateur une vue abstraite des données et permet de les localiser à partir d'un chemin d'accès ;

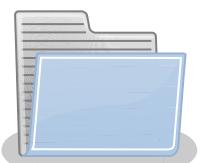
Offre d'autres fonctionnalités suivant le SGF :

Compression ;

Cryptage ;

Journalisation ;

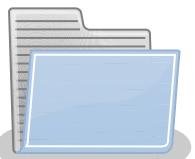
Etc.



Le système de fichiers

Quelques exemples de SGF :

SGF	FAT32	NTFS	Ext2	ReiserFS
Nom de fichier	255 car. Max.	255 car. Max.	255 car. Max.	255 car. Max.
Répertoires	non triés	B+ Tree	Non triés	B+ Tree
Journalisation	non	oui	non	oui
Adressage fichiers	32 bits	64 bits	32 bits	32 bits
Taille max fichier	4 Go	2 To	16 Go	8 To
Taille max partition	8 To	2 To	2 To	16 To
Droits utilisateurs	non	oui	oui	oui



Le système de fichiers

Vue abstraite des données du disque :

Représentation sous forme hiérarchique. Contient :

- Des répertoires (sortes de fichiers contenant des fichiers) ;
- Des fichiers.

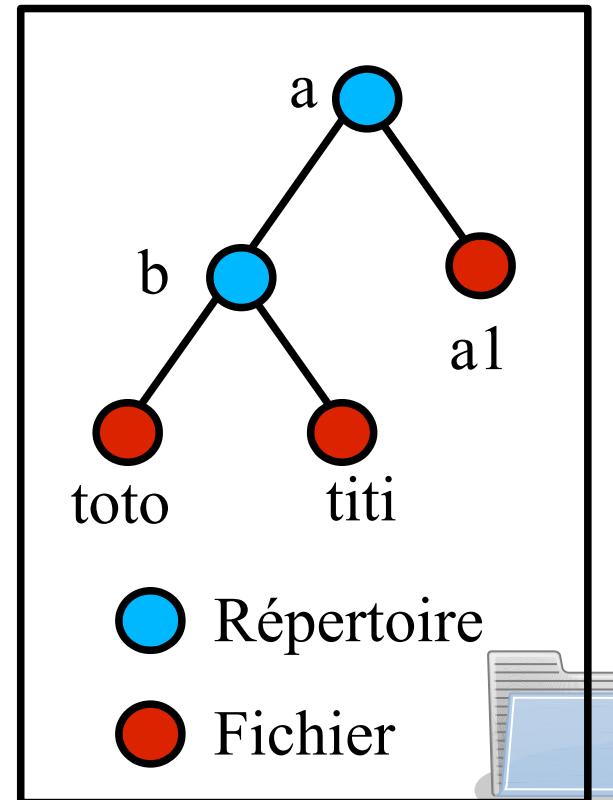
Assimilable à un arbre :

- Les répertoires sont les noeuds ;
- Les fichiers, les feuilles terminales ;
- Chacun a un nom.

Chemin absolu :

En partant de la racine :

example : a -> b -> toto



Le système de fichiers

Les commandes sur les répertoires

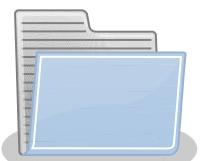
`mkdir` (MaKe DIRectory) : création ;

`rmdir` (ReMove DIRectory) : suppression ;

`pwd` (Print Working Directory) : affichage du répertoire courant ;

`cd` (Change Dir) : changer de répertoire ;

`ls` (LiSt) : liste de fichiers du répertoire.



Le système de fichiers

Les commandes sur les fichiers :

`rm` (ReMove) : effacer ;

`touch` : création fichier vide ;

`mv` (MoVe) : déplacer/renommer ;

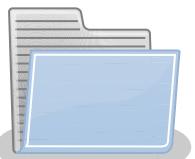
`cat` (CATalog), `more`, `less` : visualiser le contenu ;

`cp` (CoPy) : copier un fichier ;

`ln` (LiNk) : créer un lien dur ou symbolique :

Lien dur : pointe directement sur les données ;

Lien symbolique : pointe sur un autre nom.



Le système de fichiers

Ces commandes supportent les wildcards :

- '*' : remplace un ensemble (evt vide) de caractères ;
- '?' : remplace un et un seul caractère ;
- [a-z] : remplace un caractère parmi a,b,c,d,e ... z .

Exemples :

ls * .pdf : donne la liste des fichiers pdf ;

ls a* .p? ? : donne la liste des fichiers commençant par a dont l'extension fait 3 lettres commençant par p



Le système de fichiers

Navigation :

Chemin absolu : commence par '/' ;

Chemin relatif : les autres (s'expriment par rapport au répertoire courant).

Cibles particulières :

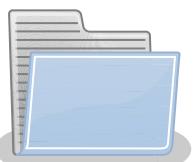
'.' : répertoire courant ;

'..' : père, répertoire contenant le répertoire courant.

Fichiers cachés :

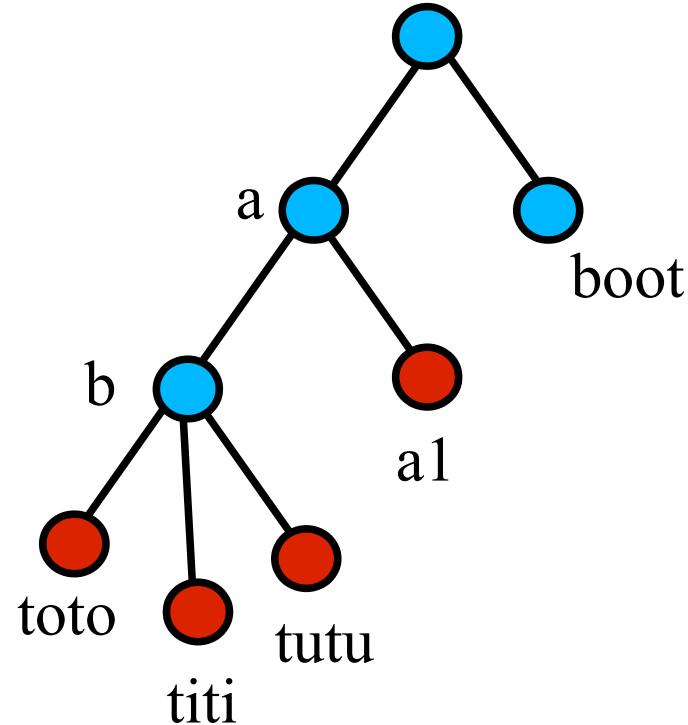
Commencent par un '.' ;

ls -a permet de les voir



Le système de fichiers

Exemples de navigation :



Répertoire courant avant : boot

Commande : cd /a/b

Répertoire courant après : b

Répertoire courant avant : b

Commande : cd ..

Répertoire courant après : a

Répertoire courant avant : a

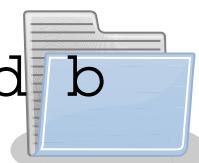
Commande : cd ../boot

Répertoire courant après : boot

Répertoire courant avant : a

Commande : cd ./b ou cd b

Répertoire courant après : b



Le système de fichiers

Recherche de fichiers :

`find` : utilitaire utilisant des expressions régulières ;

`which` : recherche de programme en utilisant la variable d'environnement PATH ;

`updatedb` : lancé par le super-utilisateur, construit une base de données des fichiers présents sur le système ;

`locate` : recherche de fichiers en utilisant la base de données précédentes.

Les utilisateurs

Unix est multi-utilisateur,

Un seul utilisateur est tout puissant !

Le super-utilisateur : root

Équivalent du compte administrateur sous windows ;

Exécute certains programmes inaccessibles aux utilisateurs ;

Peut monter et démonter tous les périphériques ;

N'utiliser ce compte que pour les opérations importantes !

Les utilisateurs sont subdivisés en groupes.

Un utilisateur peut-être dans plusieurs groupes.

Un utilisateur possède :

Les processus qu'il a lui-même lancés ;

En général un répertoire propre à lui-même ;

Des fichiers.



Les utilisateurs

Utilisateurs et fichiers :

À un fichier est associé une possession :

uid (User ID) : identifiant utilisateur ;

gid (Group ID) : identifiant de groupe ;

Modifiable à l'aide des commandes chown, chgrp.

À un fichier sont associés les droits des possesseurs :

Droit de lecture/écriture/exécution pour :

L'utilisateur qui possède le fichier ;

Le groupe qui possède le fichier ;

Les autres.

Commande associée : chmod



Les utilisateurs

Gestion des droits :

user	group	other
------	-------	-------

Ordre d'interprétation : r w x r w x r w x

Deux notations :

Mode symbolique :

chmod u+w toto

chmod g-r toto

chmod u=rwx,g-w,o= toto

Mode numérique : 3 chiffres entre 0 et 7 :

chmod 754 toto

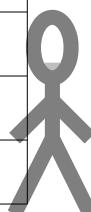
Toto est :

Lisible, modifiable, exécutable par l'utilisateur ;

Lisible, exécutable par le groupe ;

Lisible par les autres.

Chiffre	Droits
0	---
1	--x
2	-w-
3	-wx
4	r--
5	r-x
6	rw-
7	rwx



Les utilisateurs

Trois autres bits sont encore associés qui précèdent les droits normaux :

'set uid' et 'set gid' : modifient l'utilisateur apparent :

Normalement, un programme a les droits de l'utilisateur qui le lance ;

Si ces bits sont à 1 -> droits des propriétaires appliqués ;

Pour un répertoire, si le gid est 1, chaque nouveau fichier aura le gid du répertoire, sinon celui du créateur.

Le 'sticky bit' :

met en cache des programmes lancés souvent (ex: emacs);

Dans les répertoires, limite les droits (ex: répertoire /tmp).



Les utilisateurs

Le répertoire personnel (*homedir*) :

Généralement dans /home ;

Contient les fichiers propres à l'utilisateur ;

Pour l'utilisateur courant : cible ~ (tilde) :

Ex : se placer dans son homedir cd ~

Pour se déplacer dans celui d'autres utilisateurs :

Cible appelée : ~nom :

Ex : cd ~didier est équivalent à cd /home/didier



Les utilisateurs

Commandes utilisateurs :

who : pour savoir qui est connecté ;

passwd : pour changer son mot de passe (à ne pas faire à l'IUP, sous peine d'invalider son compte) ;

write : envoyer un message à un utilisateur ;

mesg : bloquer/débloquer les messages envoyés par write ;

talk : discuter avec un autre utilisateur ;

login : pour démarrer une session.



Les scripts

Qu'est ce qu'un fichier de script ?

C'est un fichier, paramétré, contenant un ensemble de commandes ;

Ces commandes sont celles étudiées ici.

Un fichier de script est interprété :

C'est le shell qui se charge de cette interprétation ;

N'importe quel shell peut être utilisé, à condition de respecter sa syntaxe.



Les scripts

Exemple d'utilisation :

Démarrage d'Unix :

Chargement et exécution du boot loader ;

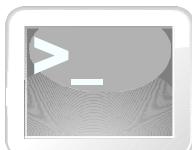
Chargement et exécution du noyau :

Initialisation des périphériques.

Lancement du père de tous les processus : init :

Lecture du fichier /etc/inittab

Contient la liste des fichiers scripts de démarrage à activer.
Lié à la notion de niveau de fonctionnement (*runlevel*).



Les scripts

Deux manières d'exécuter un script :

En lançant l'interpréteur suivi du nom du script :

Ex : sh toto.sh

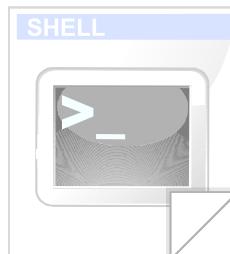
En donnant des droits d'exécution au script :

Ex : chmod 755 toto.sh ; ./toto.sh

Dans ce cas, on peut préciser l'interpréteur à utiliser sur la première ligne (*shebang*) :

Ex : #!/bin/sh

Sinon, l'interpréteur utilisé sera celui de l'utilisateur.



Les scripts

Tout langage a des variables :

Ça tombe bien, les scripts shell aussi !

Ne sont que de deux types :

Chaînes de caractères ;

Tableaux de chaînes de caractères ;

Portée :

Les variables locales : propres au shell courant ;

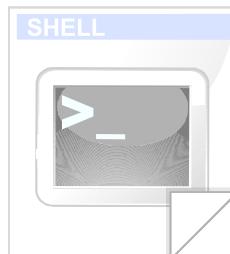
Commandes `set`, `unset`.

Les variables globales : ou variables d'environnement ;

Celles-ci sont dupliquées aux shells-fils ;

Commande `export` ;

Importer des variables : commande `source` ou `.` ;



Les scripts

Variables d'environnement courantes :

DISPLAY : écran sur lequel on travaille ;

HOME : répertoire de l'utilisateur courant (homedir) ;

HOST : nom de la machine sur laquelle est exécutée le shell ;

LOGNAME : nom de l'utilisateur courant ;

MAIL : répertoire de la boîte de l'utilisateur ;

MANPATH : endroit où se trouvent les pages de manuel ;

PATH : liste des répertoires de recherche des commandes. Séparés par ':' ;



Les scripts

Variables d'environnement courantes (suite) :

PPID : identifiant de processus du shell ;

PS[1-4] : prompts initiaux et secondaires ;

PWD : répertoire courant ;

SHELL : interpréteur de commandes utilisé ;

TERM : type de terminal ;

UID : User ID !

USER : cf LOGNAME

Utilisation des variables ailleurs (préfixe \$):

Exemple : echo \$PWD



Les scripts

Lorsque l'utilisateur se connecte :

Fichiers scripts lancés avec l'aide de login :

- /etc/profile
- ~/.bash_profile
- ~/.bash_login
- ~/.profile

Fichiers scripts lancés lorsque bash est lancé en tant que sh :

- ~/.bashrc

Les scripts

Les scripts ont des paramètres récupérables dans des variables :

Nom	Description
\$#	Nombre de paramètres
\$*	Tous les paramètres
\$0	Nom du script
\$1	Premier paramètre
...	
\$9	Neuvième paramètre

On ne peut accéder directement qu'aux 9 premiers paramètres. Pour les autres, utiliser la commande : shift



Les scripts

Les pipelines et les redirections fonctionnent toujours ;

Les commentaires sont précédés de '#' ;

Les listes :

Listes de pipelines séparés par ';', '&', '&&', '|||';

'&' : pour lancer une commande en tâche de fond ;

commande1 && commande2 :

commande2 exécutée ssi commande1 retourne 0 ;

commande1 || commande2 :

commande2 exécutée ssi commande1 retourne non 0.



Les scripts

Boucles for :

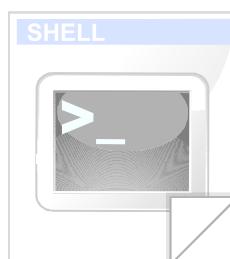
La boucle for classique :

```
for (( expr1 ; expr2 ; expr3 ) )
do
    liste d'instruction ;
done
```

Similaire au C ;

expr1 : initialisation ;
expr2 : condition d'arrêt ;
expr3 : ce qui est fait entre chaque itération.

Noter la présence des doubles parenthèses



Les scripts

La boucle for « ensembliste » :

```
for nom [in mot] ;
```

```
do
```

```
    liste d'instructions ;
```

```
done
```

nom : nom de variable ;

mot : chaîne susceptible d'être étendue

Compatible avec les wildcards, peut donner des noms de fichiers ;

Si il y a plusieurs solutions, *nom* vaudra successivement les différentes alternatives ;

Si *mot* n'est pas présent, alors *nom* prend successivement les valeurs des différents paramètres.



Les scripts

Script similaire à ls :

```
#!/bin/sh

for i in * ;
do
    echo $i ;
done
```

Script affichant ses paramètres :

```
#!/bin/sh

for i ;
do
    echo $i ;
done
```



Les scripts

Menu interactifs :

```
select nom [in mot];  
do  
    liste d'instructions;  
done
```

Affiche un menu avec les différentes valeurs de mot ;

Affiche une invite pour choisir une option ;

Exécute la liste d'instruction ;

Pour sortir, 'Ctrl d' ou l'instruction break.



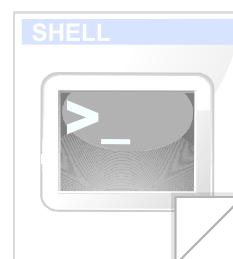
Les scripts

Les alternatives :

```
case mot in
    motif1)
        liste d'instructions1
        ;;
    motif2)
        liste d'instructions2
        ;;
    ...
*)      ...
        liste d'instructions par défaut
        ;;
esac
```

mot : valeur à comparer ;

motif : n'importe quelle expression.



Les scripts

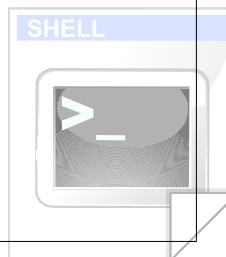
Exemple : exécuter quelques commandes au choix :

```
#!/bin/sh

select i in {ls,pwd,break} ;
do
    $i ;
done
echo "Fin de select "
```

Exemple : action sélective :

```
#!/bin/sh
select i in {bonjour} ;
do
    case $i in
        bonjour)
            echo "Hello world !"
            ;;
        *)
            break
            ;;
    esac
done
echo "Fin de select "
```



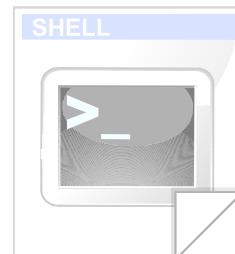
Les scripts

Les branchements conditionnels :

```
if liste;      // Dépend du code de retour de liste.  
then  
    liste d'instructions ;  
else  
    liste d'autres instructions;  
fi
```

Les boucles (tant que et jusqu'à ce que) :

while liste ;	until liste ;
do	do
liste d'instructions ;	liste d'instructions ;
done	done



Les scripts

Les tests :

Ecrire un test : 2 manières :

```
test expr  
[ expr ]
```

Différents types de comparaisons :

Sur des fichiers ;

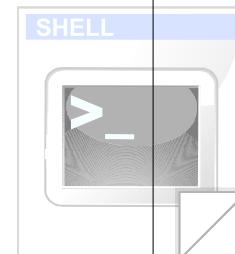
Sur des valeurs de chaînes de caractères ;

Sur des valeurs entières.



Les scripts

Expression	Description
-e fichier	Vrai si fichier existe
-d fichier	Vrai si fichier existe et est un répertoire
-f fichier	Vrai si fichier existe et est un fichier normal
-h fichier	Vrai si fichier existe et est un lien symbolique
-r fichier	Vrai si fichier existe et est lisible
-s fichier	Vrai si fichier existe et de taille non nulle
-w fichier	Vrai si fichier existe et est incriptible
-x fichier	Vrai si fichier existe et est exécutable
Fichier1 -nt fichier2	Vrai si fichier1 est modifié plus récemment que fichier2
Fichier1 -ot fichier2	Vrai si fichier1 est modifié moins récemment que fichier2
-z chaîne	Vrai si chaîne est de longueur nulle
-n chaîne	Vrai si chaîne est de longueur non-nulle
Chaine1 == chaine2	Vrai si chaine1 est égale à chaine2
Chaine1 != chaine2	Vrai si chaine1 est différent de chaine2
Chaine1 < chaine2	Vrai si chaine1 est avant chaine2 dans l'ordre alphabétique
Chaine1 > chaine2	Vrai si chaine1 est après chaine2 dans l'ordre alphabétique
Entier1 -eq entier2	Vrai si entier1 est égal à entier2
Entier1 -ne entier2	Vrai si entier1 est différent de entier2
Entier1 -lt entier2	Vrai si entier1 est strictement inférieur à entier2
Entier1 -gt entier2	Vrai si entier1 est strictement supérieur à entier2
Entier1 -le entier2	Vrai si entier1 est inférieur ou égal à entier2
Entier1 -ge entier2	Vrai si entier1 est supérieur ou égal à entier2



Les scripts

Evaluer une expression arithmétique :

2 façons de faire :

`eval expr`

`$((expr))`

Substituer à une commande sa sortie :

`commande`

` est obtenu avec les touches Alt Gr et 7.

Lire des entrées clavier :

`read nom1 nom2 ...`

Lit au clavier et stocke les valeurs dans les variables *nom1*, *nom2*, etc.



Les scripts

Les fonctions :

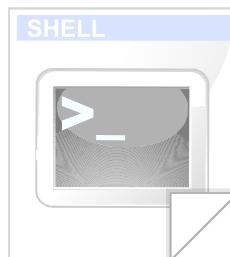
Un script peut également inclure des fonctions :

Déclaration :

```
[function] nomfonction( )  
{  
    ... liste d'instructions  
}
```

Paramètres :

Définis de la même manière que pour les scripts.



Les scripts

Exemple de script :

```
#!/bin/sh

inetd_start() {
    if [ -x /usr/sbin/inetd ]; then
        echo "Starting Internet super-server daemon"
        /usr/sbin/inetd
    fi
}

inetd_stop() {
    killall inetd
}

inetd_restart() {
    inetd_stop
    sleep 1
    inetd_start
}

case "$1" in
'start')
    inetd_start
    ;;
'stop')
    inetd_stop
    ;;
'restart')
    inetd_restart
    ;;
*)
    echo "usage $0 start|stop|restart"
esac
```



Les scripts

Exemple de script avec fonction :

```
#!/bin/bash

noname()
{
    echo "Paramètres de $0 :"
    while [ -n "$1" ];
    do
        echo $1
        shift
    done
}

echo "Paramètres de $0 :"
while [ -n "$1" ];
do
    echo $1
    shift
done

noname toto tata tutu
```

Les scripts

La syntaxe des scripts constitue un véritable langage de programmation ;

Ils combinent :

Primitives du langage ;

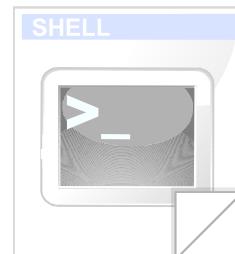
Commandes disponibles du système.

Les scripts sont utiles aux systèmes de type Unix :

Le démarrage est assuré par des scripts ;

Les tâches programmées passent par des scripts.

Le script : une façon d'automatiser un certain nombre de traitements.



L'interface graphique

Le shell n'est pas la seule manière d'interagir avec les systèmes de type Unix.

Il est possible de le faire également graphiquement.

Principe de base :

Le système et l'interface graphique sont **indépendants** !

Les interfaces graphiques sous les différentes variantes d'Unix sont interopérables et ce depuis 1984 : c'est le système **X Window** !

La variante X11 est opérationnelle depuis 1987 ;
Actuellement, X11R7.3 (septembre 2007).



L'interface graphique

Principe :

Architecture réseau de type client/serveur ;

La machine où l'on veut visualiser les informations héberge le serveur ;

Les applications graphiques sont les clients.

Les clients envoient des requêtes d'affichage au serveur :

Ex : dessine moi une fenêtre, un mouton ...

Conséquence :

Les applications peuvent être lancées sur une machine distante et affichées sur un terminal qui est une autre machine.

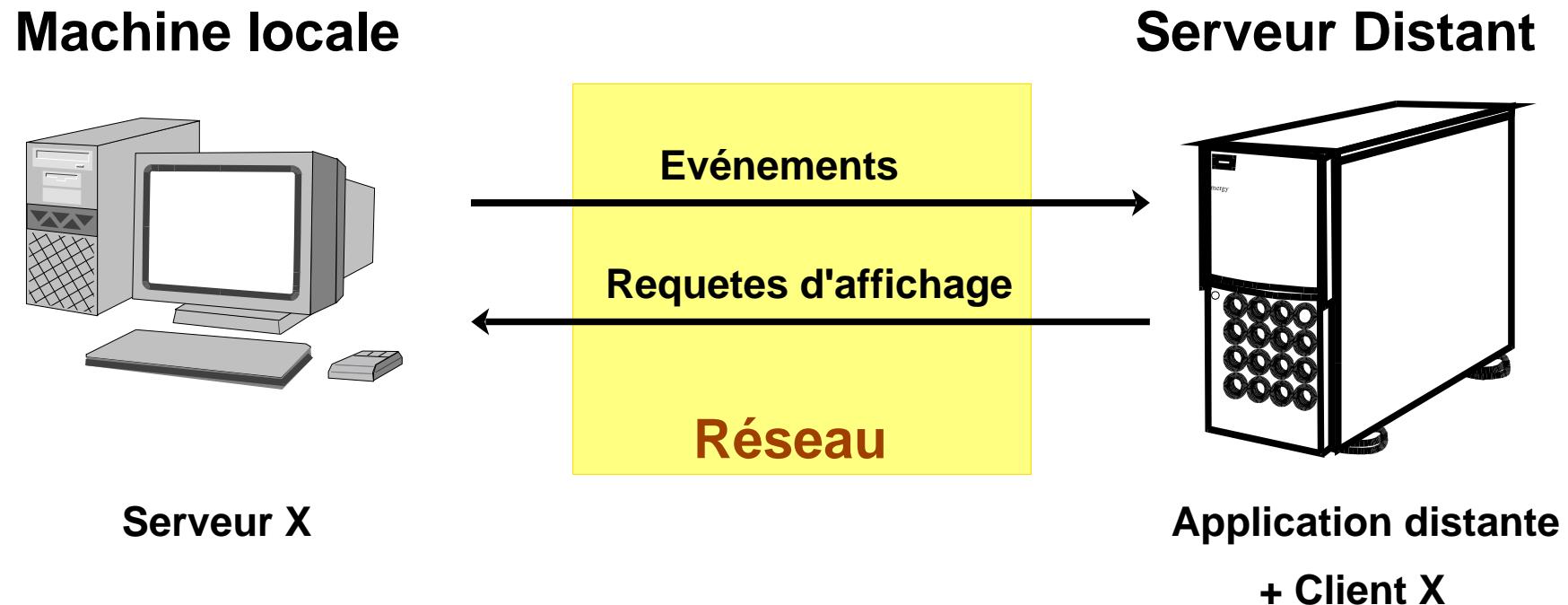


L'interface graphique

Ex : Les Tps Unix à l'IUP :

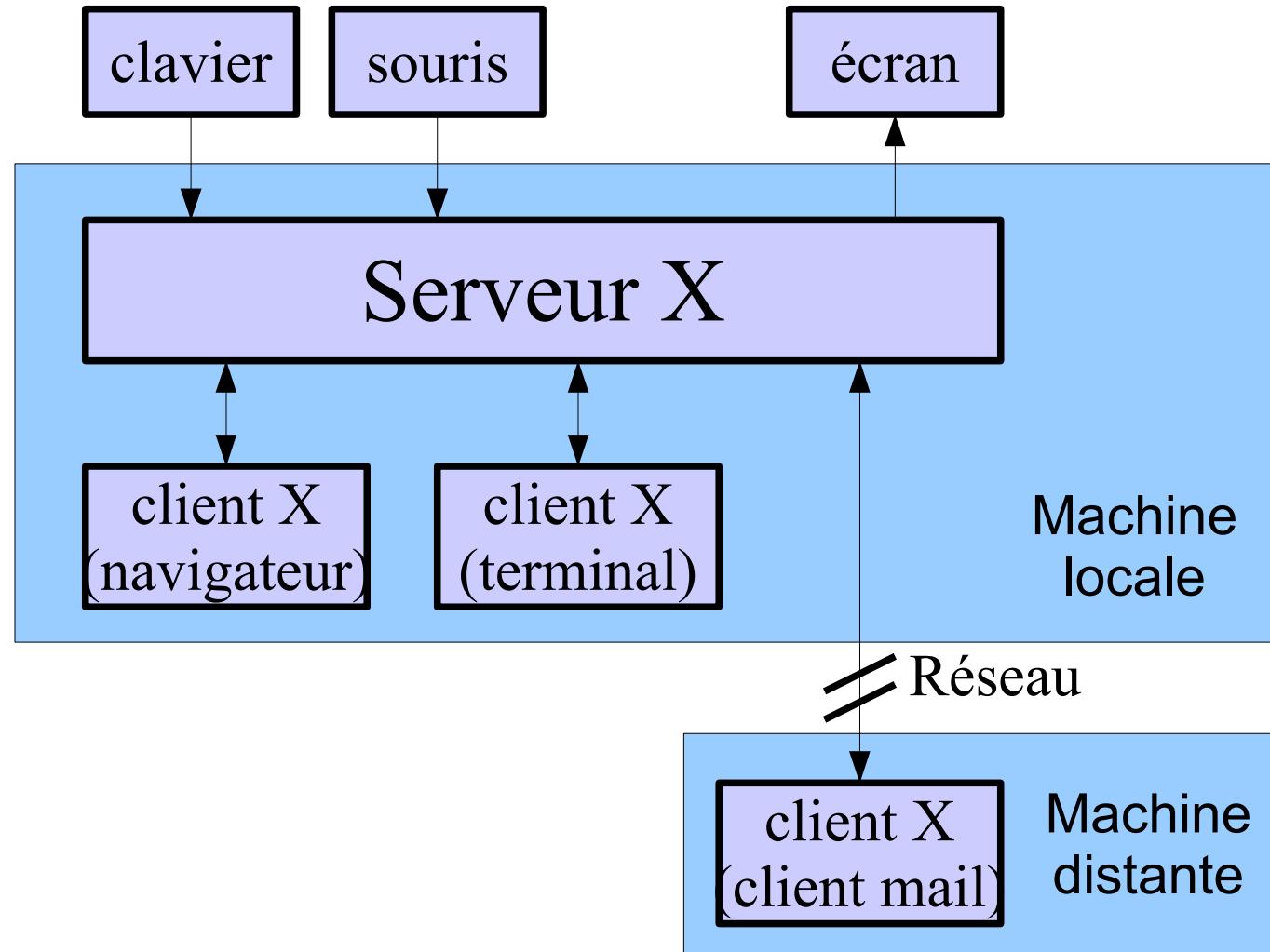
Machine locale : la vôtre, serveur X : Xwin32 ;

Machine distante : ens-unix, client X : kwrite, konsole.



L'interface graphique

Une autre vue « applicative » :



L'interface graphique

Avantages :

L'exécution est transparente vis à vis du réseau ;
Protocole robuste (éprouvé depuis 1987) ;
Implémentation de référence maintenue.

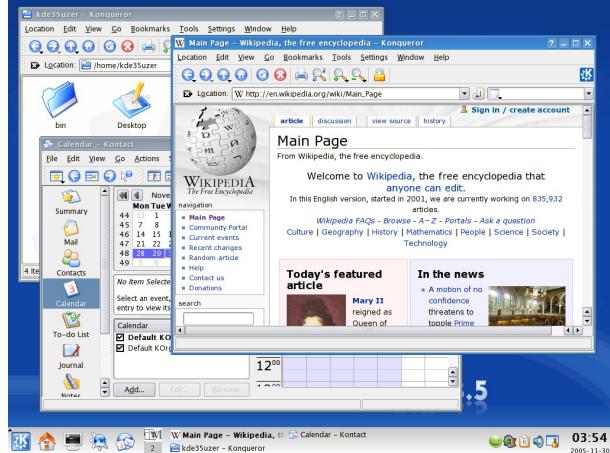
Inconvénients :

L'architecture client/serveur (latence, etc...) !
Un client X ne peut pas être détaché d'un serveur pour
être réattaché à un autre ;
L'interface utilisateur, la présentation des fenêtres n'est
pas spécifiée : un serveur X mais de multiples
aspects !!!



L'interface graphique

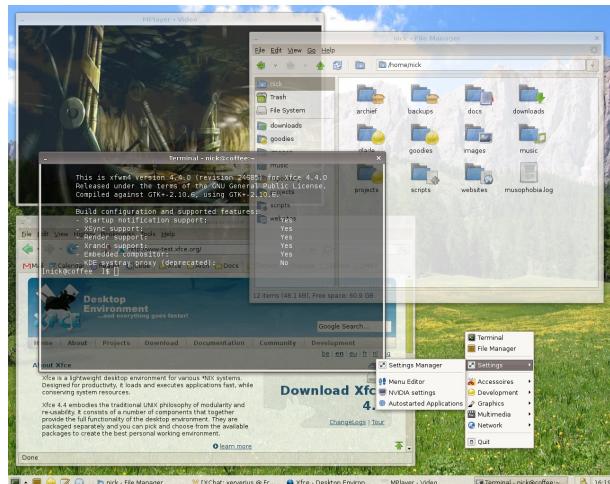
Exemples d'environnements :



KDE



GNOME



XFCE

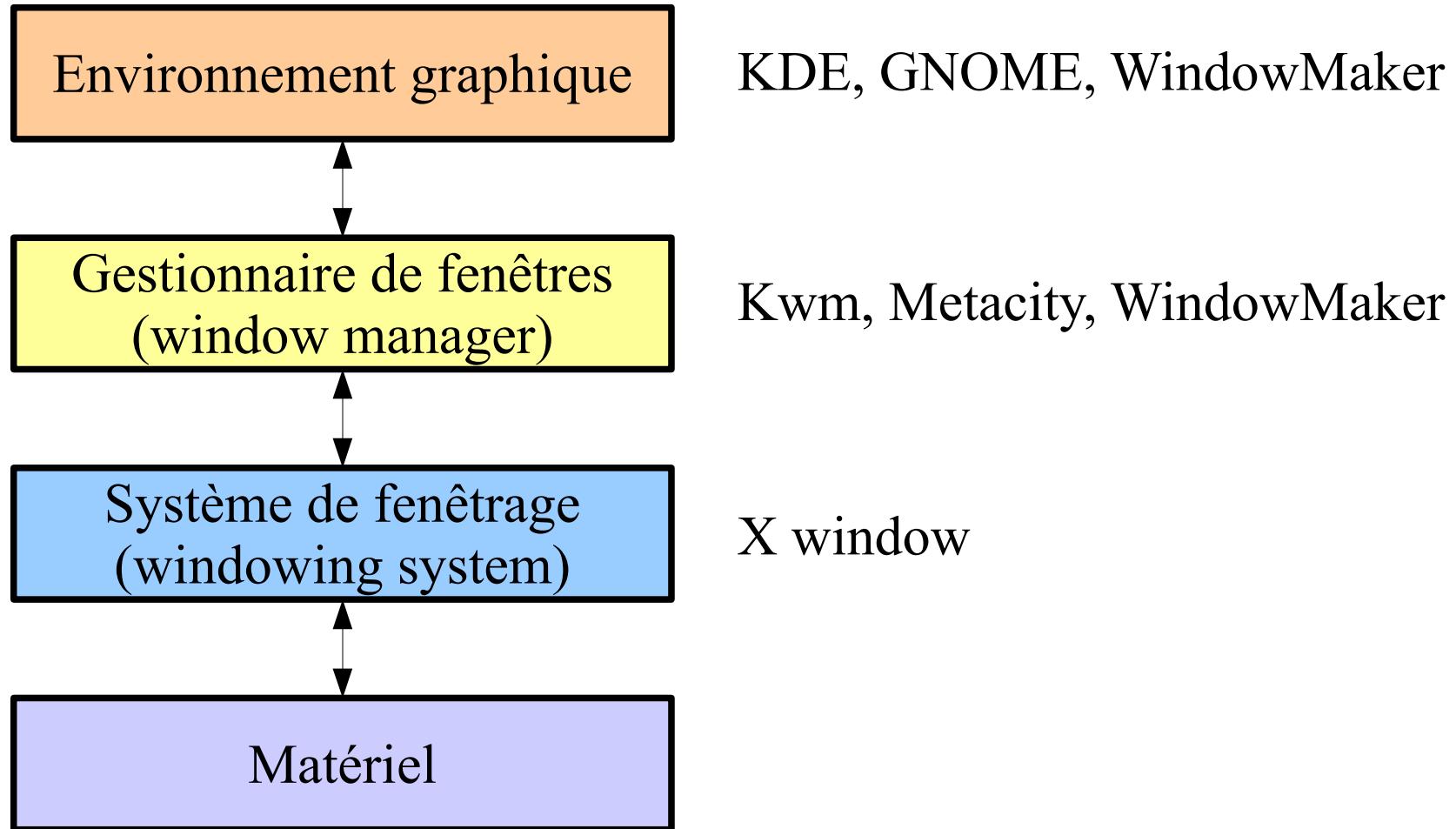


WindowMaker



L'interface graphique

X Window n'est qu'une couche :



L'interface graphique

Système de fenêtrage :

Logiciel permettant à l'utilisateur d'interagir avec plusieurs applications simultanées via le clavier et la souris ;

Contient les primitives de dessin de bas niveau (Xlib).

Gestionnaire de fenêtres :

Logiciel chargé de la gestion et du placement des fenêtres ;

Est lui-même un client du système de fenêtrage.

Environnement graphique :

Bureau, thème employé, applications spécifiques ;

Homogénéise la présentation des fenêtres ;

Associé à des bibliothèques graphiques de haut niveau :

KDE <-> Qt, GNOME <-> GTK, WindowMaker <-> WINGS



L'interface graphique

Lancement de l'interface graphique en local :

2 manières :

Runlevel 4 ou 5 des distributions :

Lancement d'un gestionnaire de session entièrement graphique (ce que l'on appelle un « display manager »)

Runlevel 3 :

Le script `startx` dans `/usr/X11R6/bin`

Fichiers importants :

`/etc/X11/xorg.conf` (ou `XFree86-config`) :

Fichier de configuration du serveur X ;

`$HOME/.xinitrc`

Configuration du gestionnaire de fenêtre et environnement de bureau.



L'interface graphique

Lancement d'applications à distance :

Par le biais du protocole XDMCP (utilisé à l'IUP) ;

Par le biais de telnet (commande à distance) :

Sur la machine distante : configurer la variable d'environnement DISPLAY:

```
export DISPLAY=AdresseIPLocale:0
```

AdresseIPLocale est l'adresse IP de votre machine

Sur la machine locale : accepter la connexion :

```
xhost +nom
```

nom : nom de la machine distante ou nom d'utilisateur.

Par le biais de ssh, en utilisant un tunnel :

```
ssh -X login@machine.distante
```

Vérifier que le serveur permette de faire des tunnels de ce type.



La norme POSIX

Norme IEEE 1003.1-1988 – ISO/IEC 9945

POSIX : Portable Operating System Interface

Objectif :

Standardisation des APIs des variantes d'Unix ;

Spécifie les interfaces utilisateurs et logicielles des SE
au travers de 17 documents.

Il existe une suite de certification POSIX :

PCTS : POSIX Conformance Test Suite

De nombreux SE s'y conforment :

Solaris, OpenSolaris, MacOSX, Windows NT, etc ...

La norme POSIX

Documentation structurée en 3 parties :

POSIX : APIs fondamentales ;

POSIX : Commandes et utilitaires ;

POSIX : Test de conformité.

Spécification écrite au travers de plusieurs documents.

La norme POSIX

POSIX.1 : API Système (language C) ;
POSIX.2 : Le Shell et ses outils ;
POSIX.4 : Threads et temps réel ;
POSIX.6 : Sécurité du système ;
POSIX.7 : Administration système ;
POSIX.8 : Réseau ;
POSIX.12 : Interfaces graphiques.

La norme POSIX

Conformité POSIX.1 d'une application :

Application strictement conforme POSIX.1

Bibliothèques
POSIX.1

Bibliothèques
C standard
(110 fonction)

Système d'exploitation

La norme POSIX

Avantages :

- API standardisée ;
- Facilite le portage du code C d'un SE à un autre ;
- Décrit un comportement standard.

Inconvénient :

- La certification POSIX est coûteuse ;
- Beaucoup de systèmes d'exploitation se conforment partiellement à POSIX.

Le noyau Linux

Le noyau est Open Source :

Chacun peut le modifier à sa guise ;

Chacun peut faire profiter la communauté de ses modifications.

Compiler le noyau : étapes :

Télécharger sur www.kernel.org ;

Décompresser le noyau (dans /usr/src en général) ;

Configurer le noyau (par un Makefile !). Options :

config : configuration interactive ;

menuconfig : version menu texte ;

xconfig : version graphique.