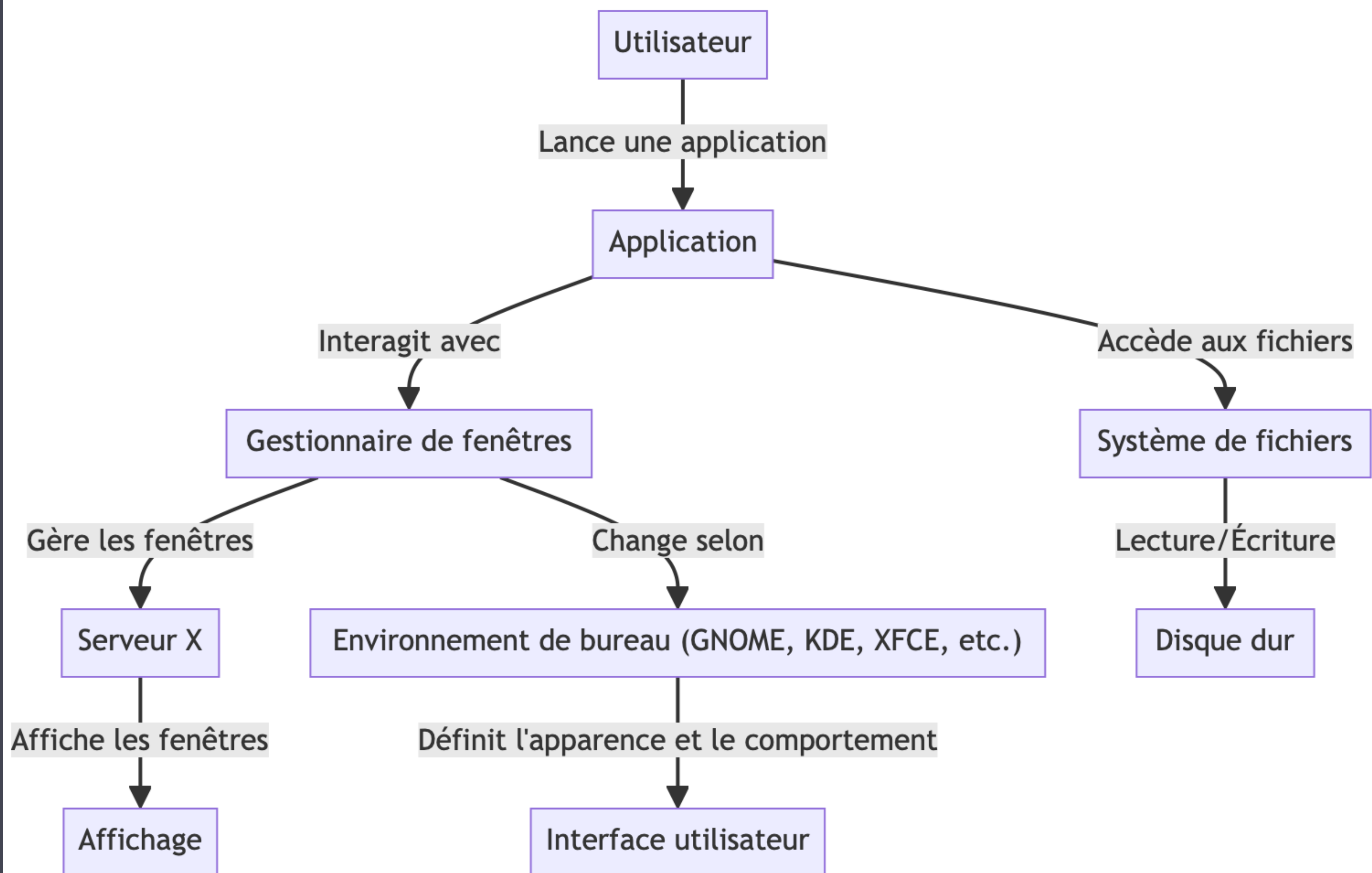


ENVIRONNEMENT DE BUREAU



DISTRIBUTIONS

LINUX



Debian



Ubuntu



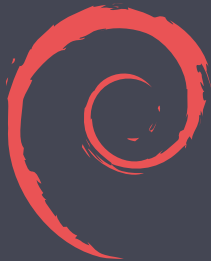


centOS





Kali Linux



Lubuntu

Distribution	Description	Usage typique	Base/Origine	Environnement de bureau par défaut
<div>Debian</div> 	Une distribution libre et universelle. Elle mise sur la stabilité et est la base de nombreuses autres distributions, dont Ubuntu.	Serveurs, stations de travail, utilisateurs qui privilégient la stabilité.	Indépendante	GNOME (mais d'autres sont disponibles)
<div>Ubuntu</div> 	Basé sur Debian, Ubuntu vise à offrir une expérience utilisateur améliorée et facile. Il est largement adopté et soutenu par la société Canonical.	Débutants, stations de travail, serveurs, cloud computing.	Debian	GNOME (depuis 17.10; précédemment Unity)
<div>CentOS</div> 	Une distribution gratuite basée sur les sources de Red Hat Enterprise Linux (RHEL). Elle offre une alternative sans coût à RHEL avec un support communautaire.	Serveurs d'entreprise, stations de travail, utilisateurs recherchant RHEL sans les coûts.	Red Hat Enterprise Linux	GNOME

<div>Kali Linux</div> 	Distribution axée sur la sécurité et le test d'intrusion. Elle vient préchargée avec de nombreux outils de piratage éthique et de test de pénétration.	Professionnels de la sécurité, tests de pénétration, recherche en sécurité.	Debian	Xfce
<div>Lubuntu</div> 	Une variante légère d'Ubuntu utilisant l'environnement de bureau LXQt. Elle est conçue pour les ordinateurs plus anciens ou avec des ressources limitées.	Anciens PC, ordinateurs avec des ressources limitées, utilisateurs recherchant une distribution légère.	Ubuntu	LXQt

DIFFÉRENTS SHELLS

Shell	Avantages	Inconvénients
sh	- Basique et largement disponible. - Bon pour apprendre les fondamentaux.	- Fonctionnalités limitées comparées aux shells modernes.
bash	- Très répandu et par défaut sur de nombreux systèmes. - Riche en fonctionnalités.	- Peut être compliqué pour les débutants.
csh	- Syntaxe similaire à C. - Introduit des fonctionnalités innovantes.	- Moins populaire que bash ou zsh pour le scripting.
tcsh	- Améliore csh avec des fonctionnalités supplémentaires.	- Moins couramment utilisé de nos jours.
ksh	- Combiné des fonctionnalités de sh et csh. - Bon pour le scripting avancé.	- Moins populaire que bash pour une utilisation générale.
zsh	- Très personnalisable. - Complet en fonctionnalités.	- Peut être écrasant pour les débutants.
fish	- Convivial et facile à utiliser. - Syntaxe simplifiée.	- Non compatible avec POSIX, ce qui peut poser des problèmes avec certains scripts.

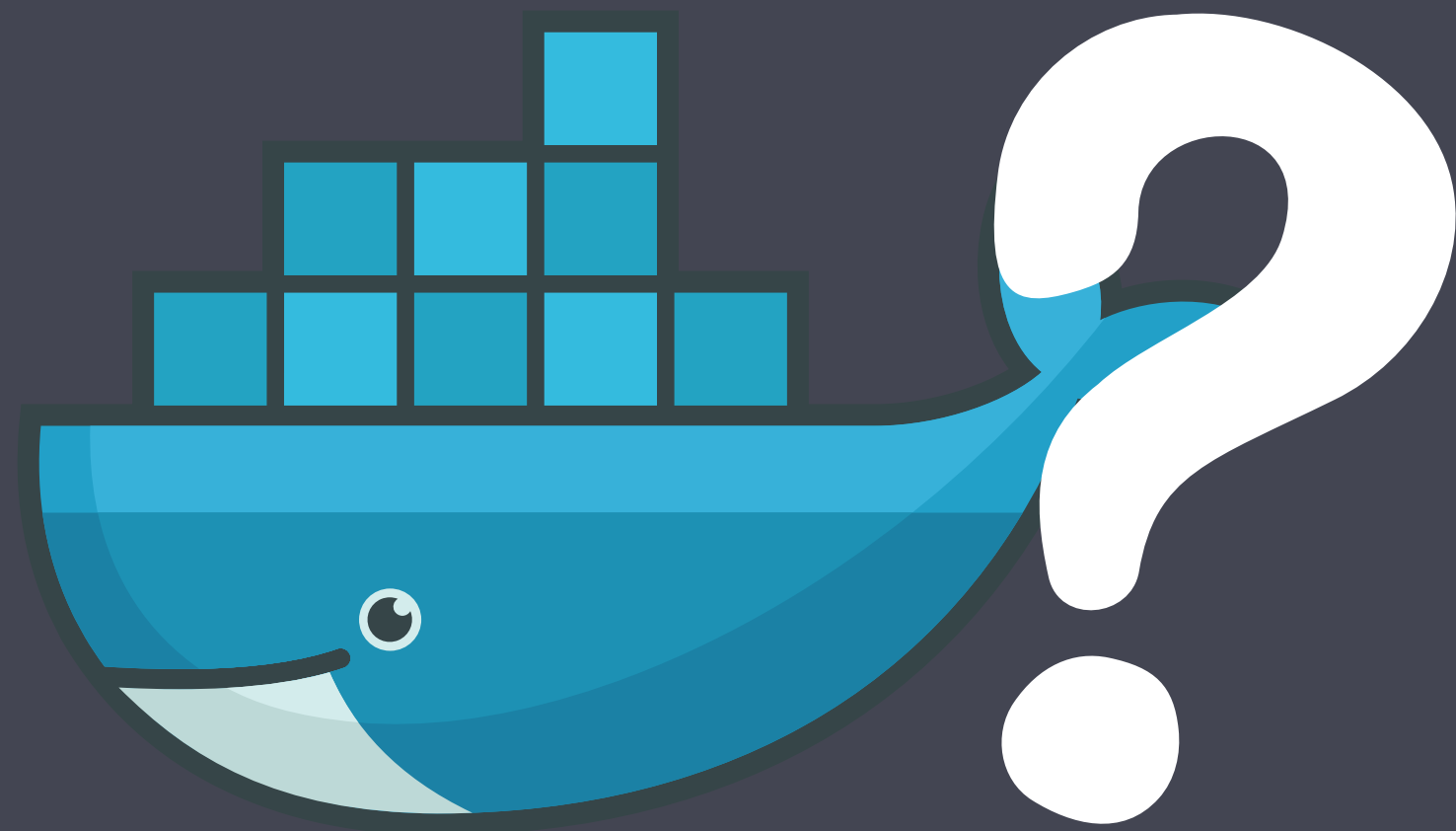
ÉDITEURS DE TEXTE

Éditeur	Avantages	Inconvénients
Vi / Vim	- Très puissant. - Léger et disponible sur la plupart des systèmes.	- Courbe d'apprentissage abrupte.
Nano	- Facile à utiliser. - Commandes affichées en bas.	- Moins de fonctionnalités que Vim ou Emacs.
Emacs	- Extrêmement extensible. - Nombreux plugins et modes.	- Courbe d'apprentissage élevée.
gedit	- Interface graphique conviviale. - Bon pour les débutants.	- Moins puissant que d'autres éditeurs.
Kate	- Fonctionnalités avancées. - Intégré à KDE.	- Dépendant de l'environnement de bureau KDE.
Sublime Text	- Interface élégante. - Extensible via plugins.	- N'est pas complètement gratuit.
VSCoDe	- Extensible via extensions. - Intégration Git et débogueur.	- Plus lourd que d'autres éditeurs en mode texte.

REDIRECTIONS & PIPES

Outil/Commande	Description	Exemple d'utilisation
`pipe` ()`		Permet de passer la sortie d'une commande à une autre commande.
`grep`	Recherche des chaînes de caractères selon un motif donné.	`echo "Hello World"``
`echo`	Affiche une chaîne de caractères à l'écran.	`echo "Bienvenue dans le monde Linux!"` (Affiche le message à l'écran)
`redirections`	Dirige la sortie d'une commande vers un fichier ou une autre commande.	`ls -l > liste_fichiers.txt` (Sauvegarde la liste des fichiers dans `liste_fichiers.txt`)
	`>` écrase le fichier ou crée un nouveau fichier.	`echo "Ceci est un test" > test.txt` (Crée ou écrase le fichier `test.txt` avec le message)
	`>>` ajoute à la fin du fichier existant.	`echo "Ceci est une autre ligne" >> test.txt` (Ajoute le message à la fin de `test.txt`)
	`<` utilise le fichier comme entrée pour la commande.	`sort < liste_fichiers.txt` (Trie le contenu de `liste_fichiers.txt`)

MACHINE VITRUELLE
CONTENEURS



DOCKER



vm

PARAMÈTRES DE SCRIPT

Syntaxe	Description	Exemple d'utilisation	Résultat (si le script est exécuté comme : <code>./script.sh arg1 arg2 arg3</code>)
<code>\$0</code>	Le nom du script lui-même.	<code>echo \$0</code>	<code>./script.sh</code>
<code>\$1,</code> <code>\$2,</code> <code>...</code>	Ces paramètres représentent les arguments passés au script. <code>\$1</code> est le premier argument, <code>\$2</code> est le second, et ainsi de suite.	<code>echo \$1</code> <code>echo \$2</code>	<code>arg1</code> <code>arg2</code>
<code>\$#</code>	Donne le nombre d'arguments passés au script.	<code>echo \$#</code>	<code>3</code>
<code>\$*</code>	Affiche tous les arguments passés au script comme une seule chaîne.	<code>echo \$*</code>	<code>arg1 arg2 arg3</code>
<code>\$@</code>	Affiche tous les arguments passés au script, chaque argument étant traité comme une chaîne distincte.	<code>for arg in "\$@"; do echo \$arg; done</code>	<code>arg1</code> <code>arg2</code> <code>arg3</code>
<code>\$?</code>	Donne le code de sortie de la dernière commande exécutée. Un <code>0</code> signifie que la commande a réussi, et toute autre valeur indique une erreur.	<code>./une_autre_commande</code> <code>echo \$?</code>	(Affiche le code de sortie de <code>./une_autre_commande</code>)
<code>\$\$</code>	Donne le PID (Process ID) du script en cours d'exécution.	<code>echo \$\$</code>	(Affiche le PID du script)
<code>\$!</code>	Donne le PID du dernier processus exécuté en arrière-plan.	<code>sleep 10 &</code> <code>echo \$!</code>	(Affiche le PID de la commande <code>sleep</code>)

BOUCLES & SCRIPTS

Type de Boucle	Description	Exemple
<code>`for`</code>	Exécute des commandes pour chaque élément dans une liste.	<code>`for i in 1 2 3; do echo \$i; done`</code>
<code>`for ((...))`</code>	Boucle style C, souvent utilisée pour des séquences numériques.	<code>`for ((i=0; i<3; i++)); do echo \$i; done`</code>
<code>`while`</code>	Exécute des commandes tant qu'une condition est vraie.	<code>`i=1; while [[\$i -le 3]]; do echo \$i; i=\$((i+1)); done`</code>
<code>`until`</code>	Exécute des commandes tant qu'une condition est fausse.	<code>`i=1; until [[\$i -gt 3]]; do echo \$i; i=\$((i+1)); done`</code>

Voici quelques points à retenir :

- Les boucles ``for`` peuvent être utilisées pour parcourir des séquences ou des listes d'éléments.
- Les boucles ``while`` et ``until`` sont conditionnelles et continuent de s'exécuter tant qu'une certaine condition est respectée.
- La syntaxe des boucles doit être respectée scrupuleusement, en particulier l'utilisation de ``do`` et ``done`` pour encadrer les commandes exécutées dans la boucle.

ATELIER PRATIQUE

GUIDÉ

ATELIER PRATIQUE EN AUTONOMIE