MO101
LINUX ET USAGE DU SHELL

ALEXANDRE CHAPOUTOT — U2IS, ENSTA PARIS

2023-2024

PARTIE 1 : SYSTÈME LINUX

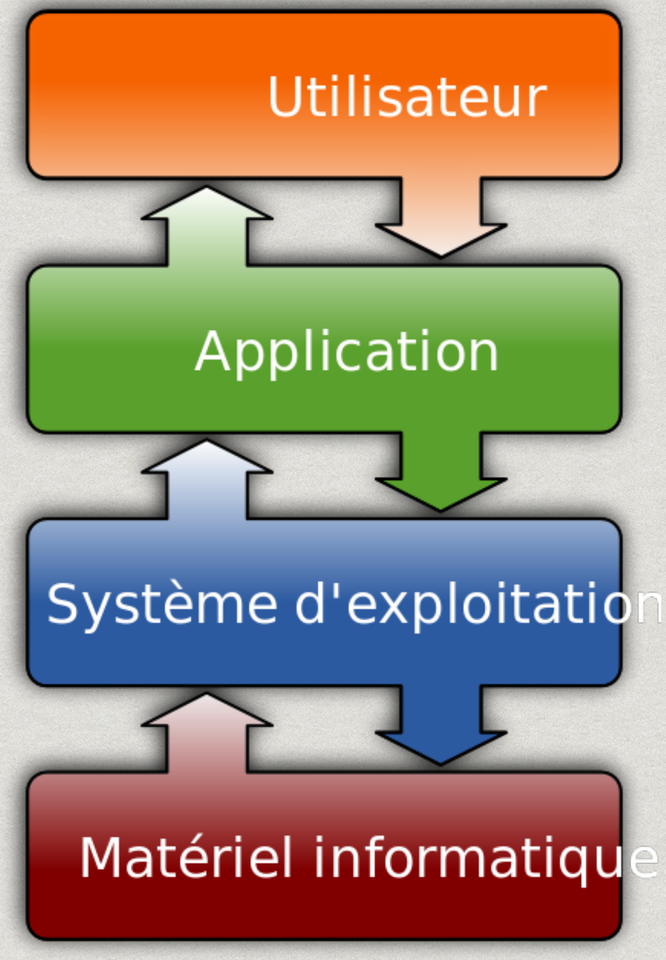
Composants électroniques

Source image :

Système d’exploitation

D’après Wikipédia

*« En **informatique**, un **système d'exploitation** (souvent appelé **OS** – de l'anglais Operating System) est un ensemble de **programmes** qui dirige l'utilisation des ressources d'un **ordinateur** par des **logiciels applicatifs** »*

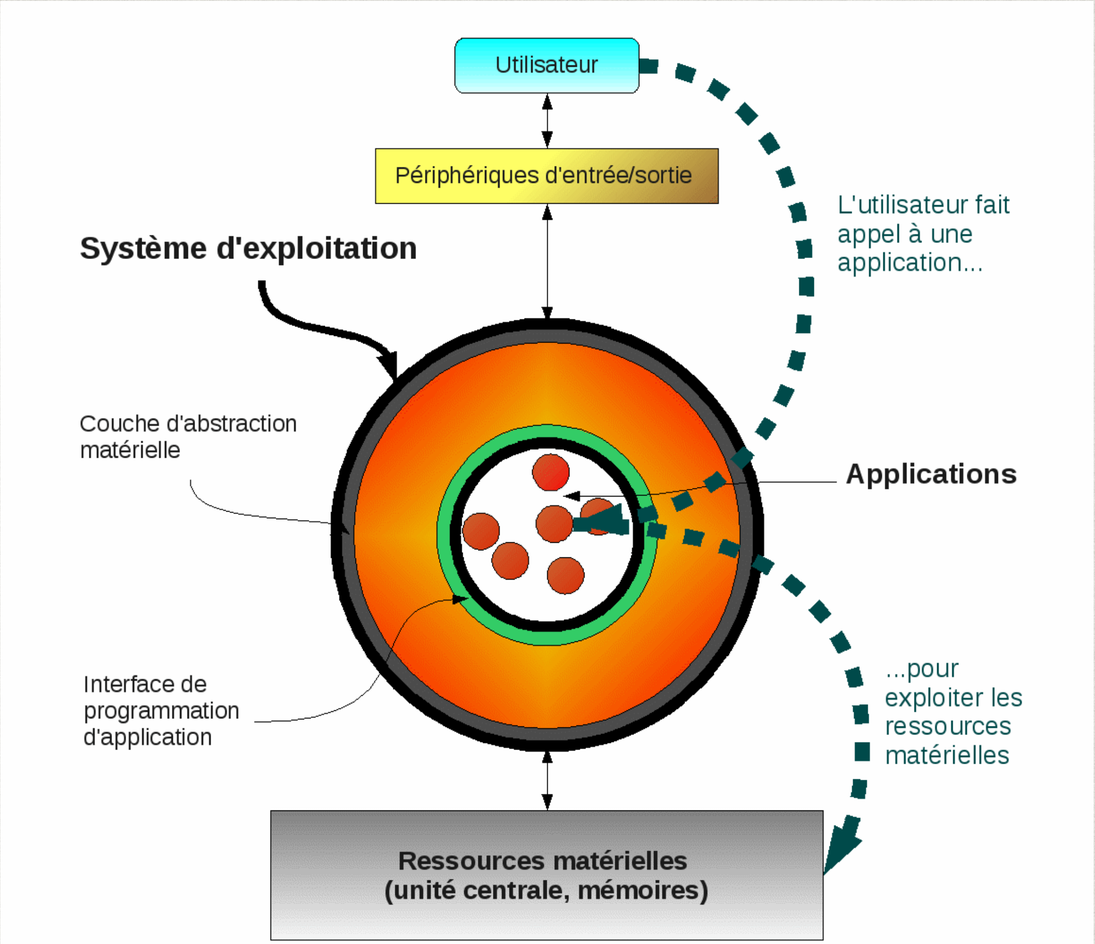


Source image : https://fr.wikipedia.org/wiki/Système_d%27exploitation

Architecture des OS

Les principaux composants des OS sont :

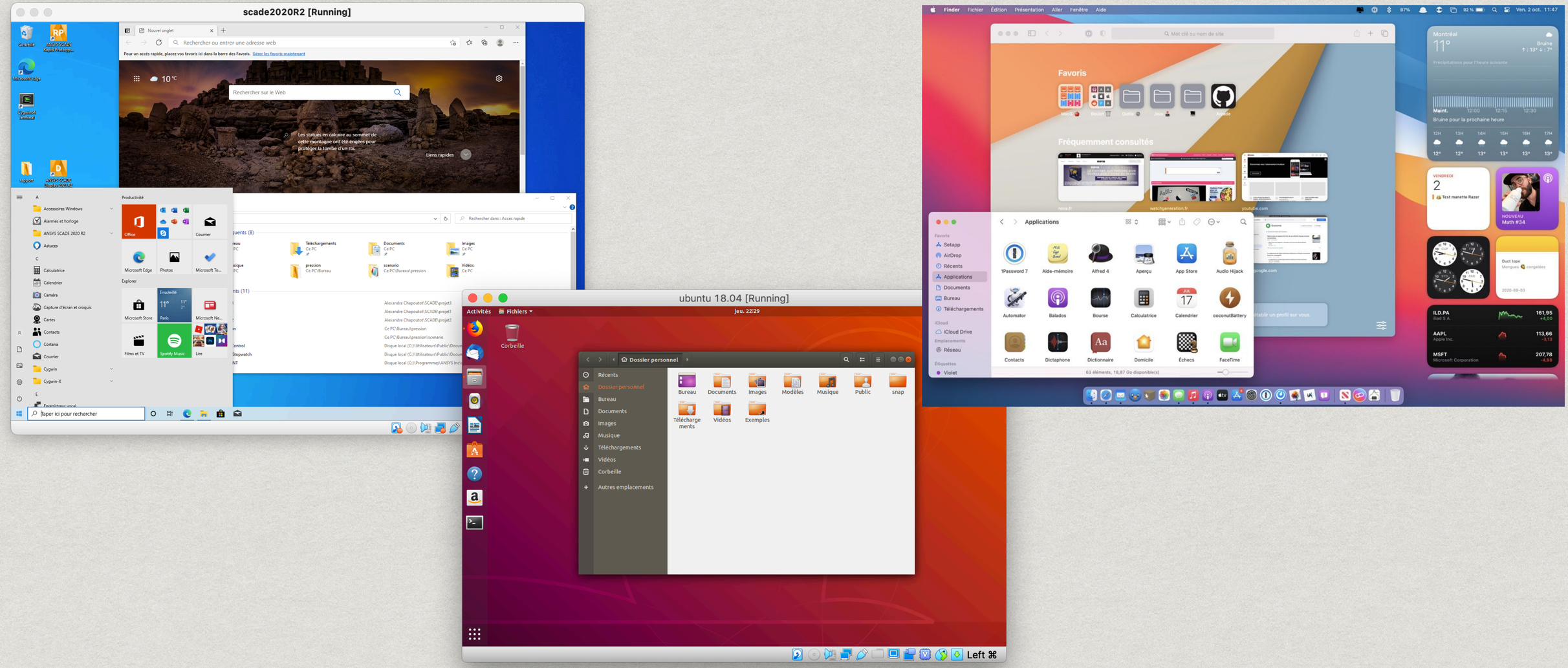
- Le **noyau**
- L'**interface utilisateur**



Les relations entre utilisateur, applications, système d'exploitation et matériel

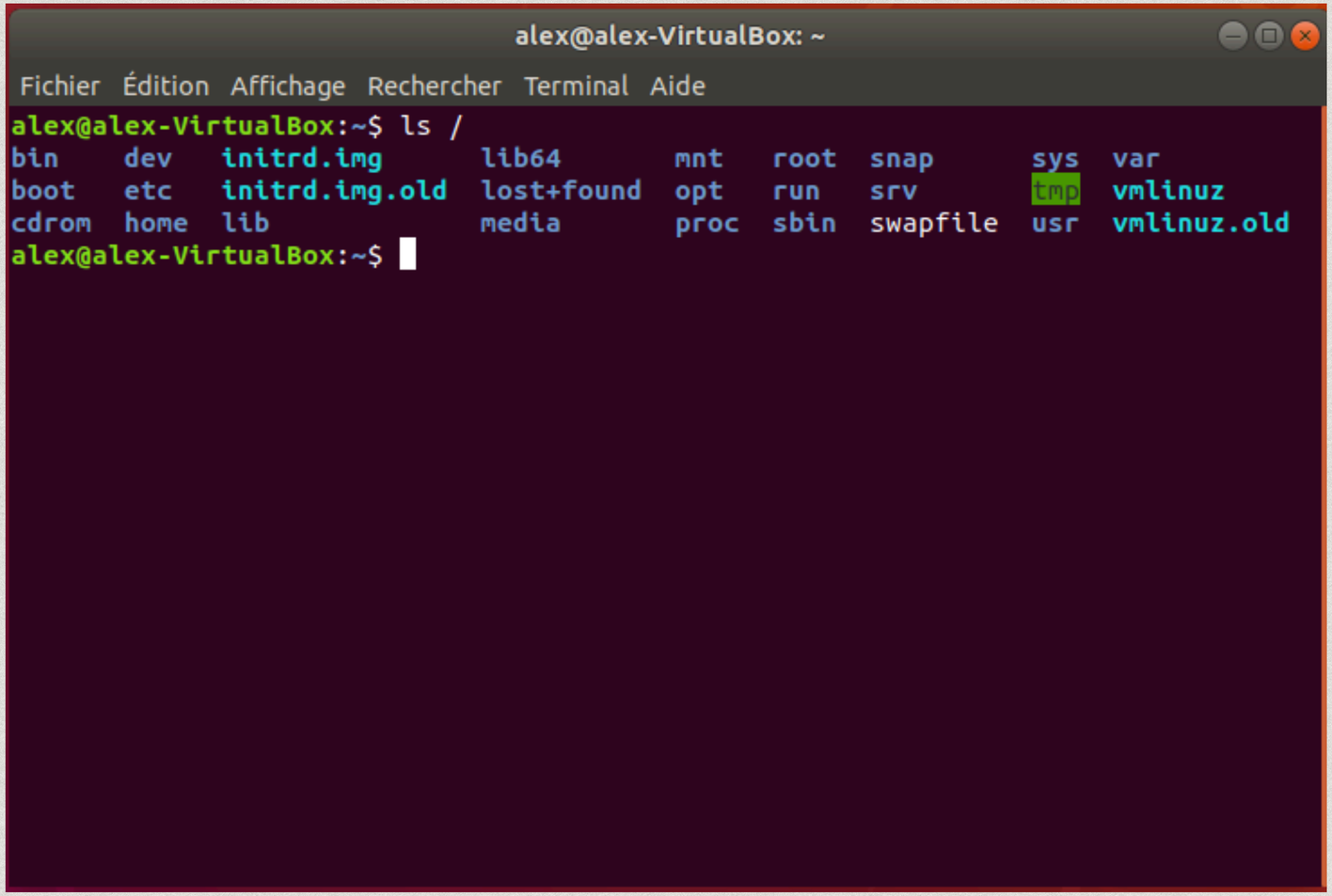
Source image : https://fr.wikipedia.org/wiki/Système_d%27exploitation

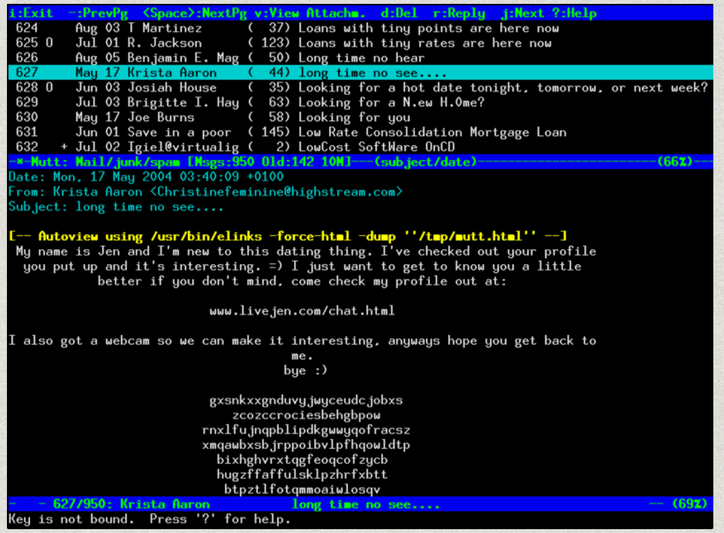
Interface utilisateur version graphique

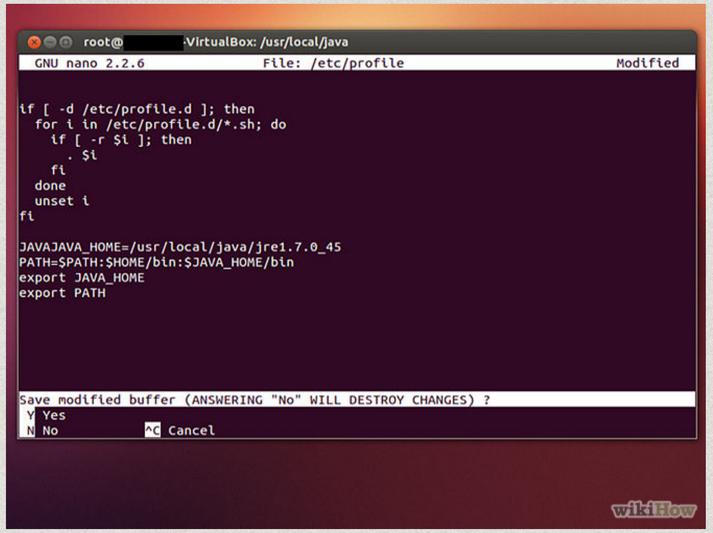


Source image : pour Mac OS https://www.macg.co/macos/2020/11/macos-big-sur-la-version-finale-est-enfin-disponible-117737

Interface utilisateur version textuelle







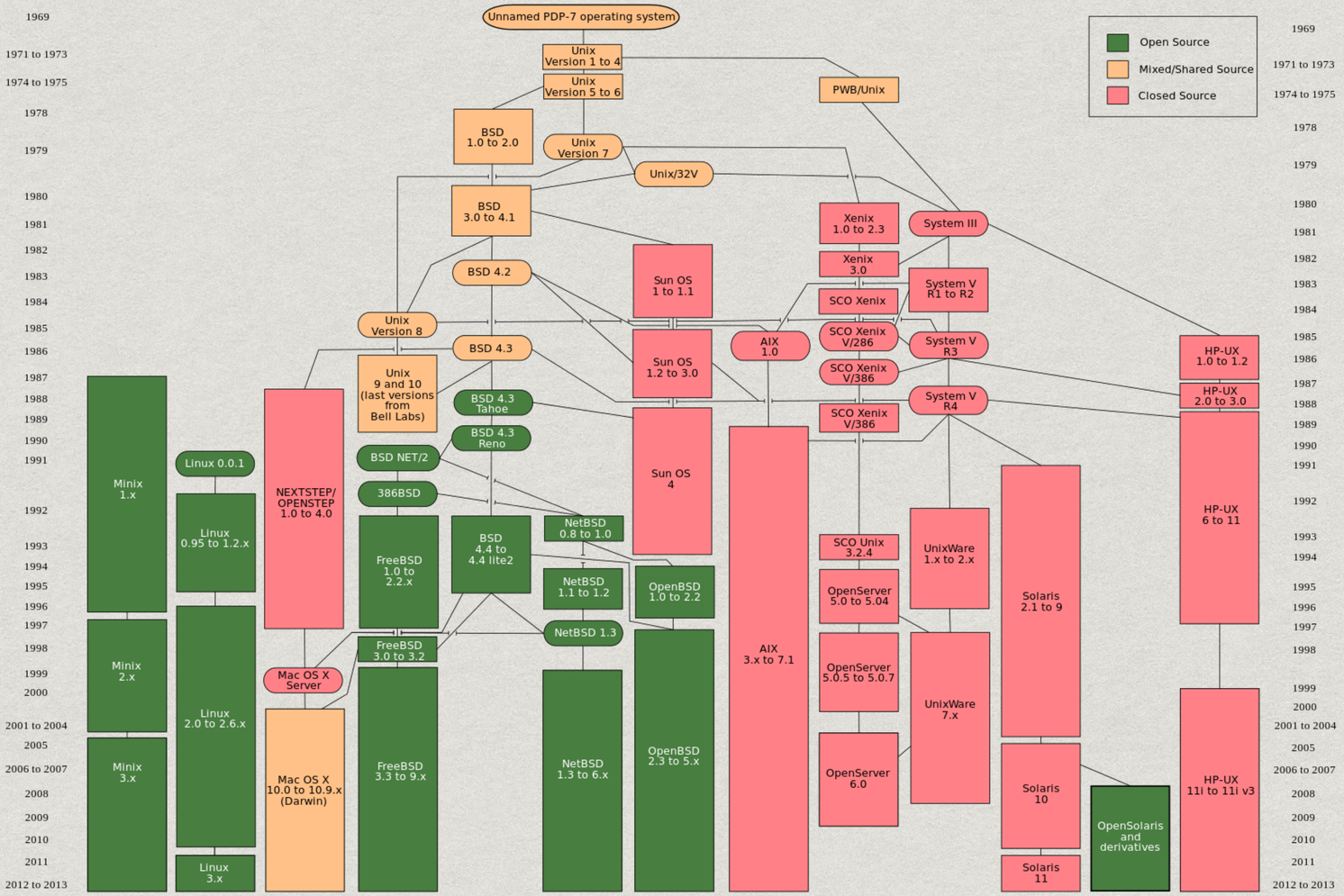
Qu’est-ce qu’UNIX ?

- UNIX est un système d’exploitation multi-utilisateur et multi-tâche crée en 1969 chez AT&T qui est au fondement de nombreux autres OS

Dennis Ritchie et Ken Thompson (assis) sur PDP-11

Source image : https://en.wikipedia.org/wiki/History_of_Unix

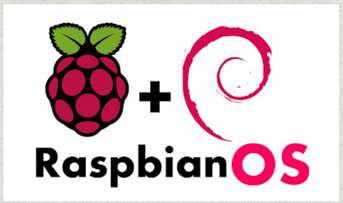
Evolution des systèmes UNIX



Source image : https://fr.wikipedia.org/wiki/Unix

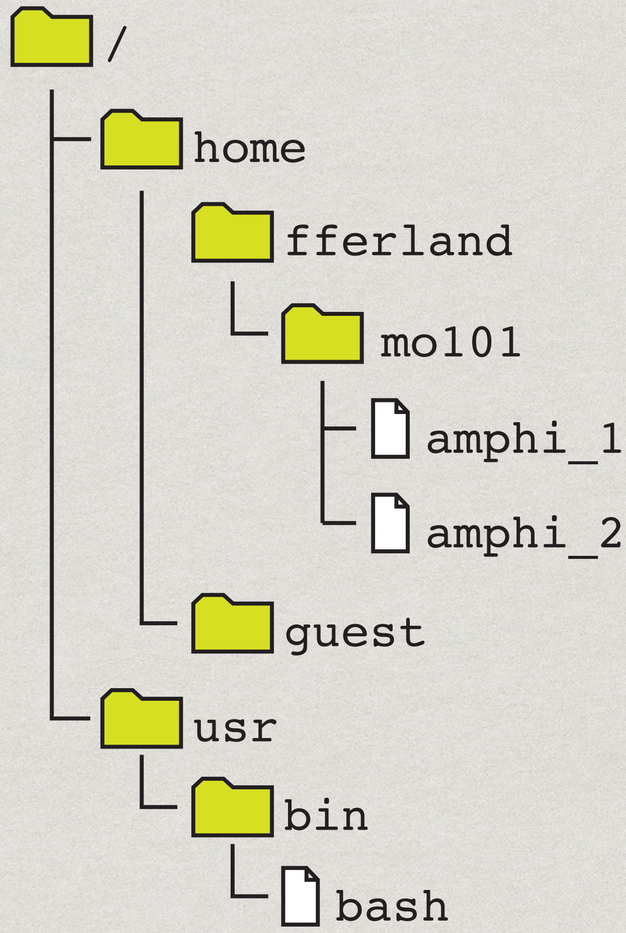
Distributions Linux

- Linux est le nom du noyau qui est inspiré du fonctionnement de UNIX
- Linux est aussi un terme générique pour nommé les « distributions Linux », l'OS dans son entièreté
- Il y a beaucoup de distributions Linux !

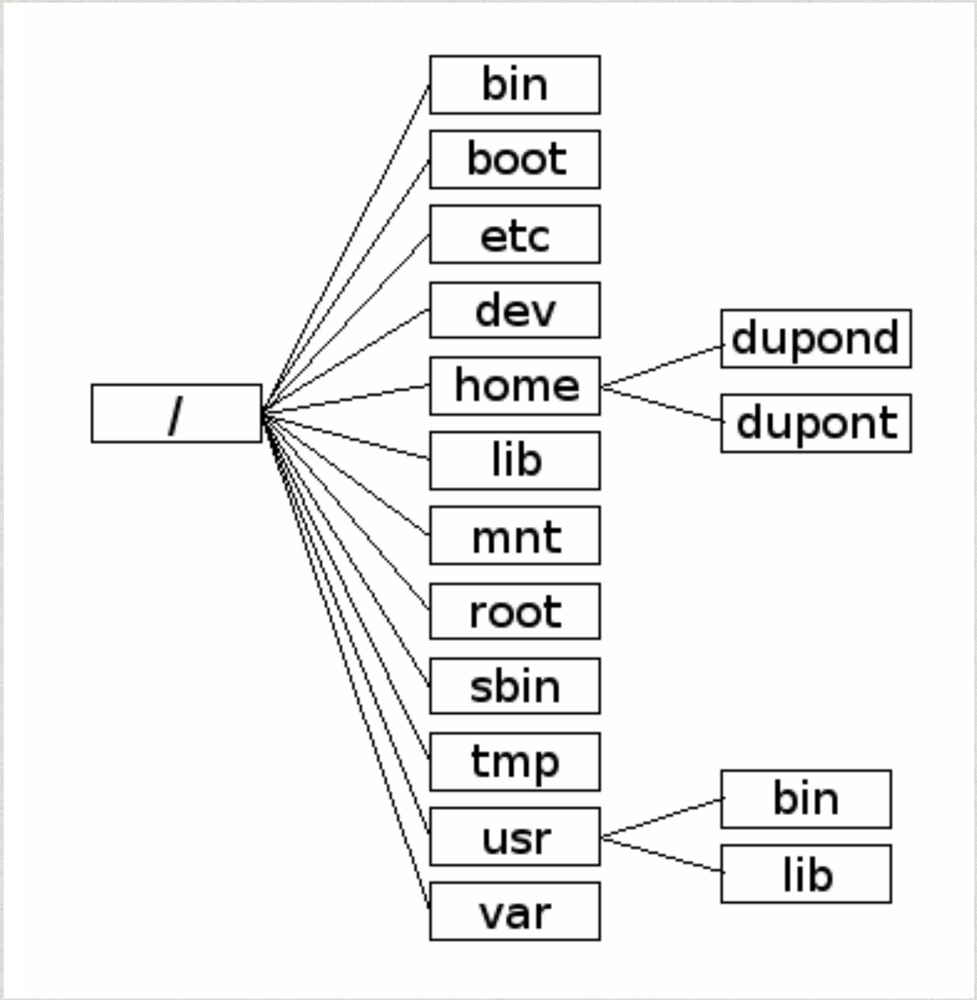
Arborescence de fichiers

- Deux concepts importants: les **fichiers** et les **dossiers**
- Organise de façon compréhensible l'espace de stockage du système
- Une arborescence unique et découplée du support physique



Arborescence de fichiers

- / est la **racine** de l'arborescence
- **bin** contient les programmes de base comme l'interprète de commandes Bash
- **boot** contient les informations pour démarrer l'OS
- **etc** contient les fichiers de configurations du système ou des applications
- **dev** contient les fichiers associés aux périphériques
- **home** contient les répertoires personnels des utilisateurs
- **lib** contient les bibliothèques partagées
- **usr** contient les applications et bibliothèques utilisateurs
- **var** contient les données écrites fréquemment comme les log



Fichiers

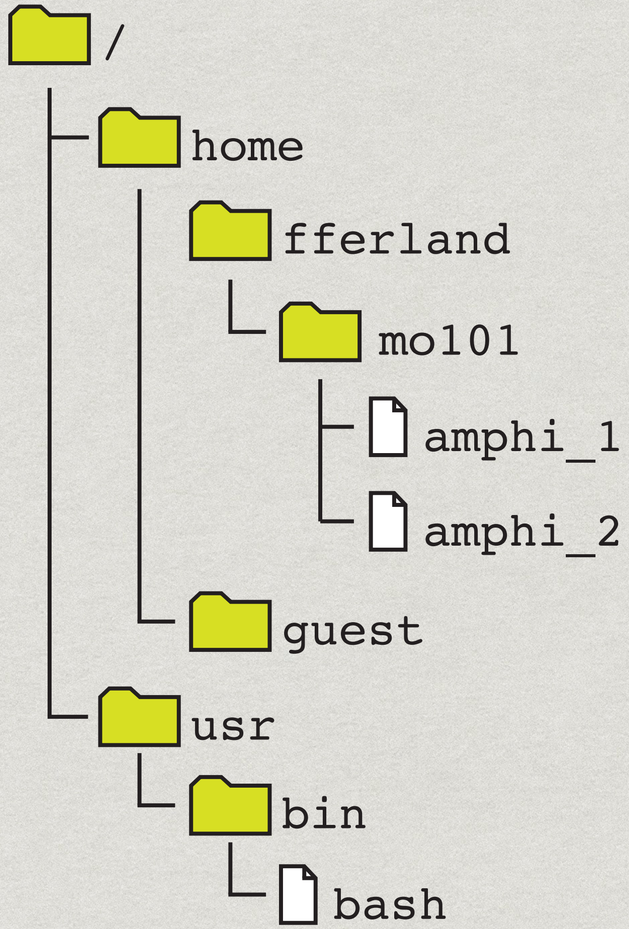
- Le plus petit élément manipulable de l'arborescence
- Deux grandes catégories: texte (brut) et binaire (images, son, exécutables, …)

Nom de fichiers

- Exemple classique: `mon_fichier.txt`
  - Un nom: `mon_fichier`
  - Une extension: `.txt`. Facultative, mais fortement recommandée
- Espaces possibles, mais généralement à éviter pour la ligne de commande
- Certains caractères interdits (ex: "`/`") ou à usage particulier (ex: "`?`", "`*`")
- Peut aussi être caché, si préfixé d'un "."

Dossiers

- Aussi connu comme **répertoire**, ***folder***, ou ***directory***
- Permet de regrouper des fichiers sous un thème commun
- Un dossier peut contenir d'autres dossiers
- L'arborescence d'un système a pour racine le dossier "/"



Chemin absolu

```
/home/alex/docs/sujet.txt
```

Adresse d’un répertoire ou un fichier depuis la racine de l’arborescence

Chemin absolu

```
/home/alex/docs/sujet.txt
```

Chemin absolu

`/home/alex/docs/sujet.txt`

Chemin absolu

```
~/docs/sujet.txt
```

Chemins relatifs

`/home/alex/fichier.txt`

- `fichier.txt`: depuis `~/`
- fichier.`txt`: depuis `/home/alex/`
- `../fichier.txt`: depuis `~/docs/`
- `../../fichier.txt`: depuis `~/docs/images/`

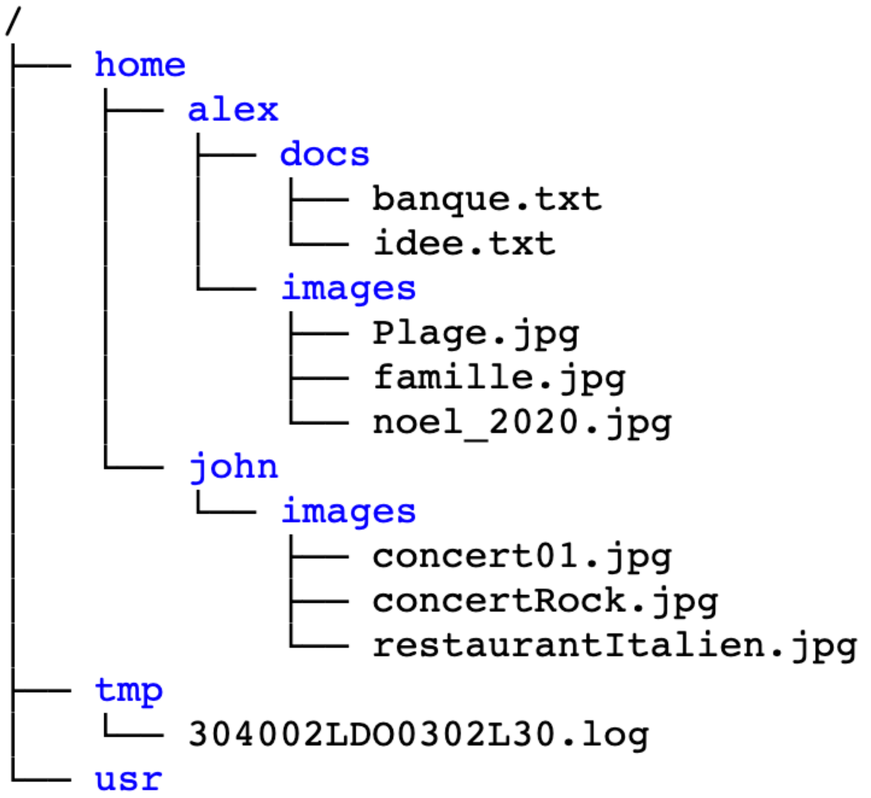
Chemins relatifs

`/home/alex/fichier.txt`

- `fichier.txt`: depuis `~/`
- fichier.`txt`: depuis `/home/alex/`
- `../fichier.txt`: depuis `~/docs/`
- `../../fichier.txt`: depuis `~/docs/images/`

Adresse d’un répertoire ou un fichier depuis un autre répertoire

Exemples



- Donnez le chemin absolu pour atteindre `concert01.jpg`

  /home/john/images/concert01.jpg

- Donnez le chemin absolu pour atteindre `docs`

  /home/alex/docs

- A partir de `tmp`, donnez le chemin relatif pour atteindre `Place.jpg`

  ../home/alex/images/Plage.jpg

- A partir du répertoire `images` chez `john`, donnez le chemin relatif pour atteindre le répertoire `tmp`

  ../../../tmp

Symboles à retenir

- `/`: Séparateur de dossier, et/ou racine de l'arborescence
- `./`: dossier en cours (généralement facultatif)
- `.fichier`: Fichier caché
- `../`: dossier parent
- `~/`: dossier personnel

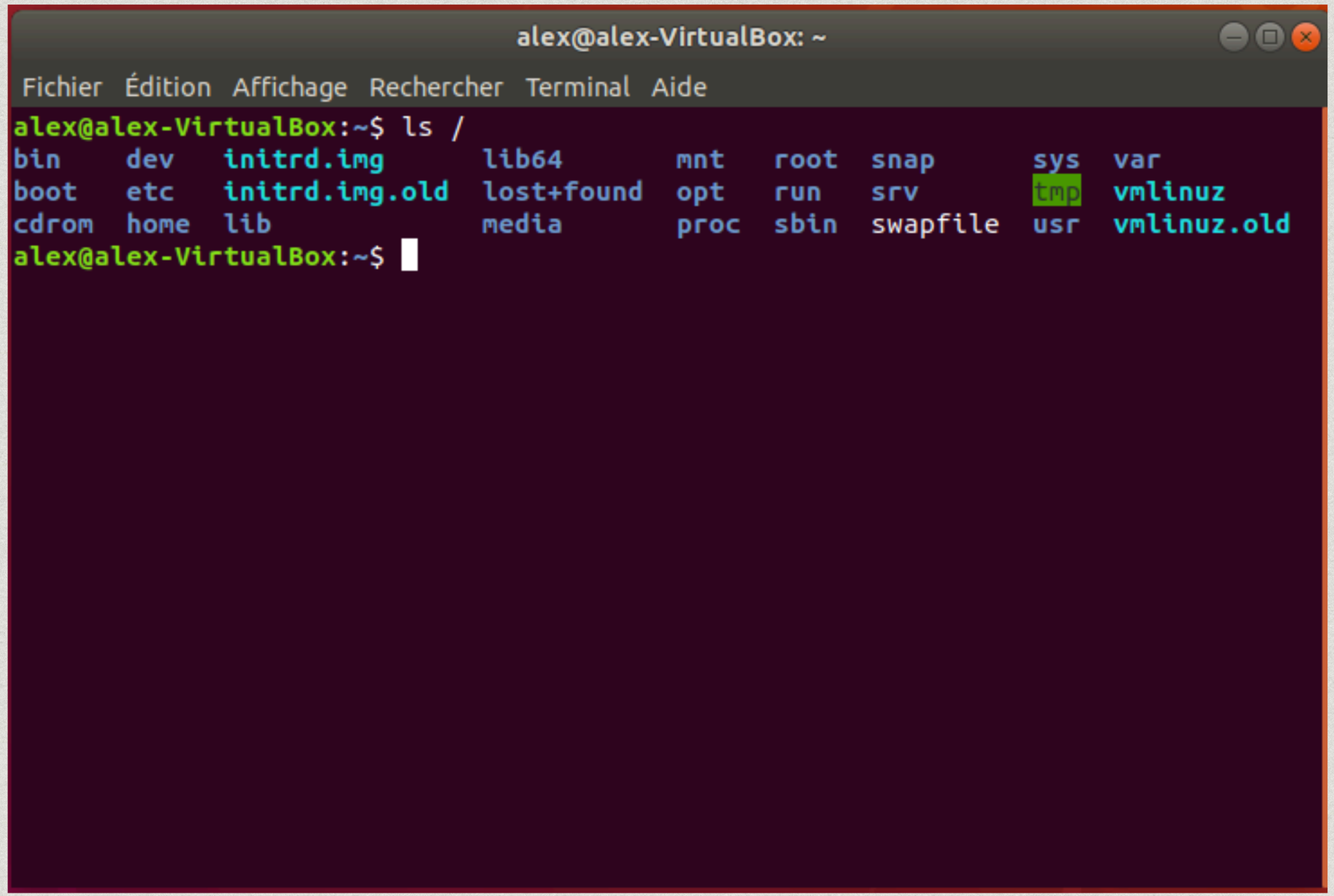
Droits d’accès

- Trois types d’accès :
  - Lecture
  - Écriture
  - Exécution ou accès au contenu
- Trois catégories d’utilisateurs :
  - Propriétaire
  - Groupe (du propriétaire)
  - Tous les autres utilisateurs

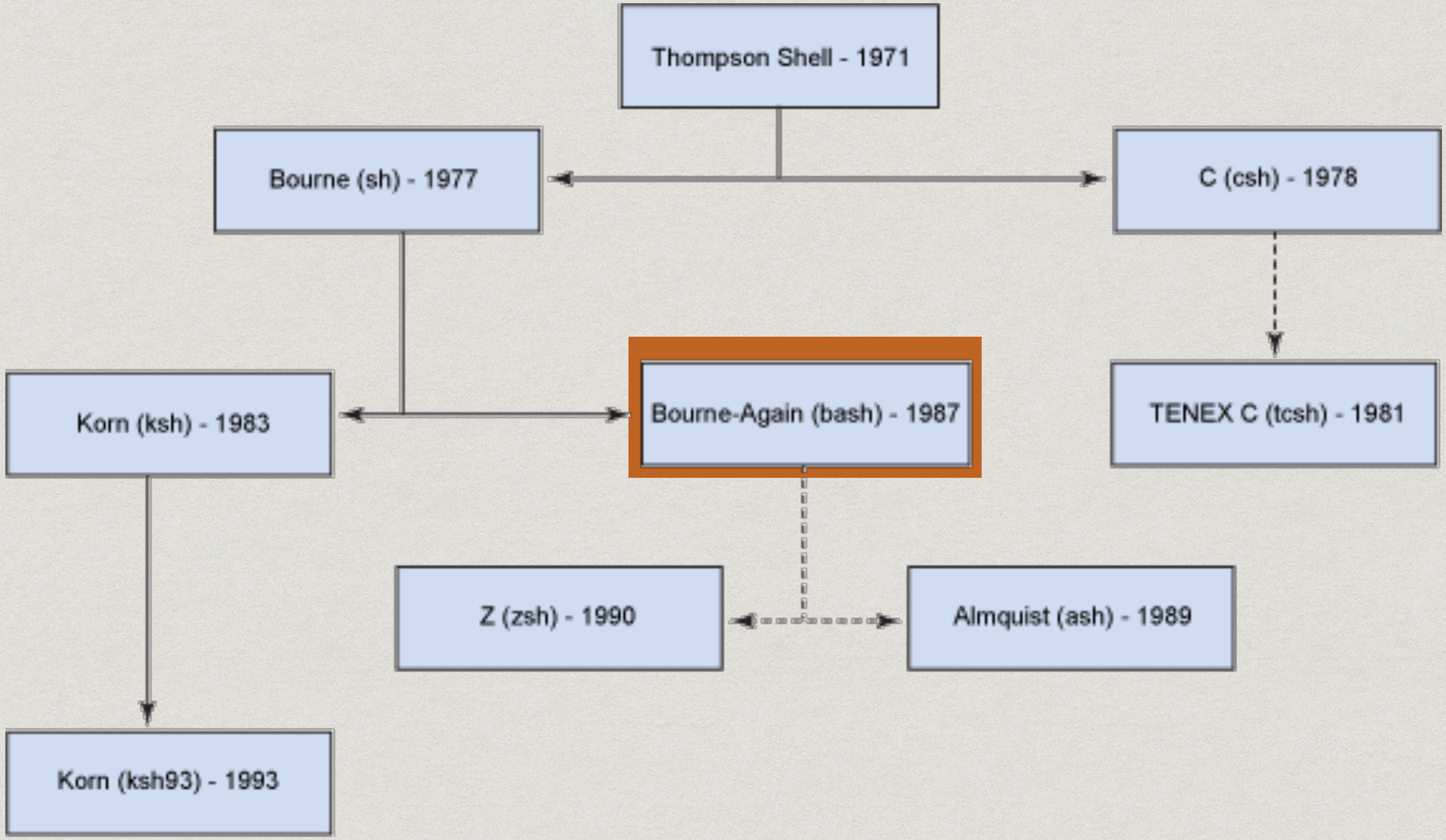
```
alex@MacAlex examples % ls -l
total 0
drwxr-xr-x  2 alex  staff  64 23 jul  2020 dir1
drwxr-xr-x  2 alex  staff  64 23 jul  2020 dir2
drwxr-xr-x  2 alex  staff  64 23 jul  2020 dir3
```

PARTIE 2 : LIGNE DE COMMANDE

Terminal


alex@alex-VirtualBox: ~
Fichier Édition Affichage Rechercher Terminal Aide
alex@alex-VirtualBox:~$ ls /
bin dev initrd.img lib64 mnt root snap sys var
boot etc initrd.img.old lost+found opt run srv tmp vmlinuz
cdrom home lib media proc sbin swapfile usr vmlinuz.old
alex@alex-VirtualBox:~$

«Shell» ou interprète de commandes



Invite de commande

```
chapoutot@alex-VirtualBox:~$_
```

- `chapoutot` : le nom de l'utilisateur
- `alex-VirtualBox` : le nom du poste
- `~` : Le dossier courant
- `$` : Droits d'accès (normal ou administrateur) et début de commande

⭐ Format du prompt décrit par la variable PS1="\u@\h:\w\$

Exemples de commandes

```
chapoutot@alex-VirtualBox:~$ wget https://perso.ensta-paris.fr/
~chapoutot/teaching/shell-liesse2021.tgz

chapoutot@alex-VirtualBox:~$ tar -xvzf shell-liesse2021.tgz

chapoutot@alex-VirtualBox:~$ find racineLiesse2021 -type t -name “T*“
```

Différents types de commandes

- Dans le terminal il y a deux types de commandes:
  - les **commandes builtin,** celles fournies par Bash
  - les **commandes systèmes,** celles fournies par la distribution Linux

Différents types de commandes

- **variable** `PATH` chaîne de caractères qui contient des noms de répertoires séparés par des `:` où il faut chercher les commandes utilisées usuellement

- La commande `which` recherchent parmi les répertoires contenant des commandes l'adresse de la commande passée en argument
- La commande `type -t [-a]` permet de savoir comment est exécuté une commande (builtin ou système)

Différents types de commandes

- Documentation d’une commande système
  - `man ls`
- Documentation d’une commande builtin
  - `help cd`

Configuration du Bash

- Fichier de configuration `.bashrc` à la racine du dossier personnel, *i.e.*, à l'adresse
  - `/home/$USER/.bashrc`
- Bash gère les **alias** permettant d'avoir des raccourcis de commande.
  - `alias rm='rm -i'`
  - `alias ll='ls –color=auto -l -h'`

Configuration du Bash

- **Alias recommandés pour tous les systèmes**
  - `alias rm='rm -i'`
  - `alias ls='ls –colore=auto'`
  - `alias la='ls -a'`
  - `alias ll='ls -l -h'`
  - `alias grep='grep –color'`

Configuration du Bash

- Après modification du fichier `.bashrc` il faut
  - soit quitter et relancer le terminal
  - soit charger à nouveau le fichier de configuration
    - `source ~/.bashrc`

Commandes utiles

- Se déplacer dans l'arborescence
  - **`cd`** (Change Directory)
- Lister le contenu d'un répertoire
  - **`ls`** (List Segment)
- Créer un répertoire
  - **`mkdir`** (MaKe DIRectory)
- Supprimer un répertoire vide
  - **`rmdir`** (ReMove DIRectory)
- Afficher le chemin absolue du répertoire courant
  - **`pwd`** (Print Working Directory)
- Changer les droits d'accès
  - **`chmod`** (CHange Files MODes)
- Chercher des éléments de l'arborescence
  - **`find`**

Construction de commandes

```
mkdir -p dossier/sous-dossier
```

Construction de commandes

Exécutable **Arguments**

Construction de commandes

Exécutable

Arguments

Option(s)

Commandes et options

- Les options peuvent modifier grandement le comportement d'une commande
- Par exemple pour la commande « ls »
  - **-S** : trie les fichiers par taille
  - **-t** : trie les fichiers par date de modification (+récent en premier) puis par ordre lexicographique
  - **-R** : affiche le contenu des dossiers de manière récursive

Sélection de fichiers

- Il est parfois utile de sélectionner plusieurs fichiers en une seule fois grâce à des **patterns**
  - Caractère `?` : Un et un seul caractère
  - Symbole `*` : Aucun ou plusieurs caractères
  - [] : intervalle de caractères, p. ex., [A-Z], [a-z], [0-9]
  - {} : alternatives séparées par des virgules, p. ex., A*.{txt, jpg}

Exemples de sélection de fichiers

- ABC
- Test
- Test1
- Test2
- Test30

Exemples de sélection de fichiers

- ABC
- Test
- Test1
- Test2
- Test30

```
ls *
```

Exemples de sélection de fichiers

- ABC
- Test
- Test1
- Test2
- Test30

```
ls A*
```

Exemples de sélection de fichiers

- ABC

- Test
- Test1
- Test2
- Test30

```
ls Test*
```

Exemples de sélection de fichiers

- ABC
- Test
- Test1
- Test2
- Test30

```
ls Test?
```

Exemples de sélection de fichiers

- ABC
- Test
- Test1
- Test2
- Test30

```
ls ?est?*
```

Exemples de sélection de fichiers

- Test30
- Test300
- Test301

```
ls Test30?
```

Exemples de sélection de fichiers

- Test30
- Test300
- Test301

- Vide01
- Vide02
- Wagon.txt

```
ls *{id,go}*
```

Modifications des droits d’accès

- La commande **chmod**
  - chmod [ugo][+-][rwx] [-R] file

- [ugo] choisir le groupe d’utilisateurs
- [+-] ajouter ou enlever des droits
- [rwx] designer les droits à modifier
- [-R] de manière récursive

```
alex@dhcpuei20 images % ls -l
total 0
-rw-r--rw-  1 alex  staff  0 15 avr 15:17 concert01.jpg
-rw-r--r--  1 alex  staff  0 15 avr 15:17 concertRock.jpg
-rw-r--r--  1 alex  staff  0 15 avr 15:17 restaurantItalien.jpg
alex@dhcpuei20 images % chmod o+x-r concert01.jpg
alex@dhcpuei20 images % ls -l
total 0
-rw-r---wx  1 alex  staff  0 15 avr 15:17 concert01.jpg
-rw-r--r--  1 alex  staff  0 15 avr 15:17 concertRock.jpg
-rw-r--r--  1 alex  staff  0 15 avr 15:17 restaurantItalien.jpg
alex@dhcpuei20 images %
```

PARTIE 3 : MANIPULATION DE FICHIERS

Deux types de fichiers

- **Fichiers textuels**
  - Lisible directement depuis le terminal
  - Nécessite un encodage particulier
- **Fichiers binaires**
  - Difficile à visualiser sans transformation (texte ou graphique)

Fichier binaire

Exemple d’un document PDF

Encodage du texte

- Objectif : permet d’associer un caractère à une valeur numérique
- Plusieurs types: ASCII, ISO 8859-1, UTF-8, …

ASCII

- Inventé en 1960, conçu pour la télé-impression
- Chaque caractère est enregistré sur 7 bits (0-127)
- Inclut des codes de contrôle pour terminaux
- N’inclus que l’alphabet latin, et sans accents

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

UTF-8 (Unicode)

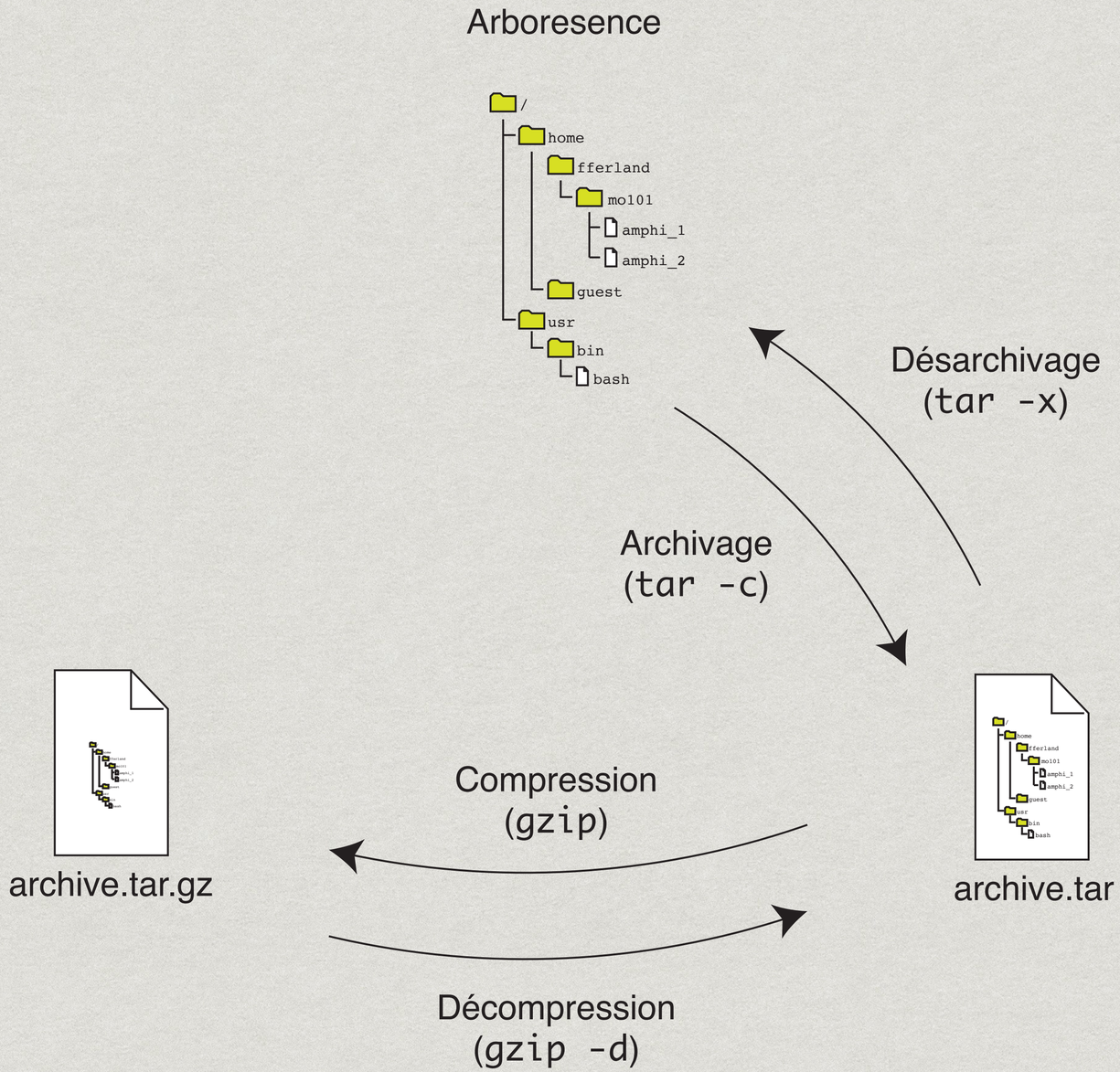
- L’ASCII est encodé sur 7 bits, mais stocké sur des octets (8 bits, 0-255)
- L’UTF-8 se sert du bit additionnel pour différentier un caractère ASCII traditionnel et un caractère étendu
  - 0-127: ASCII
  - 128-255: Points de contrôle et caractères additionnels
- Un point de contrôle indique que le(s) prochain(s) octet(s) est un caractère étendu
  - Plus d’un million de caractères disponible, incluant plusieurs alphabets et émoticônes (*emoji*)

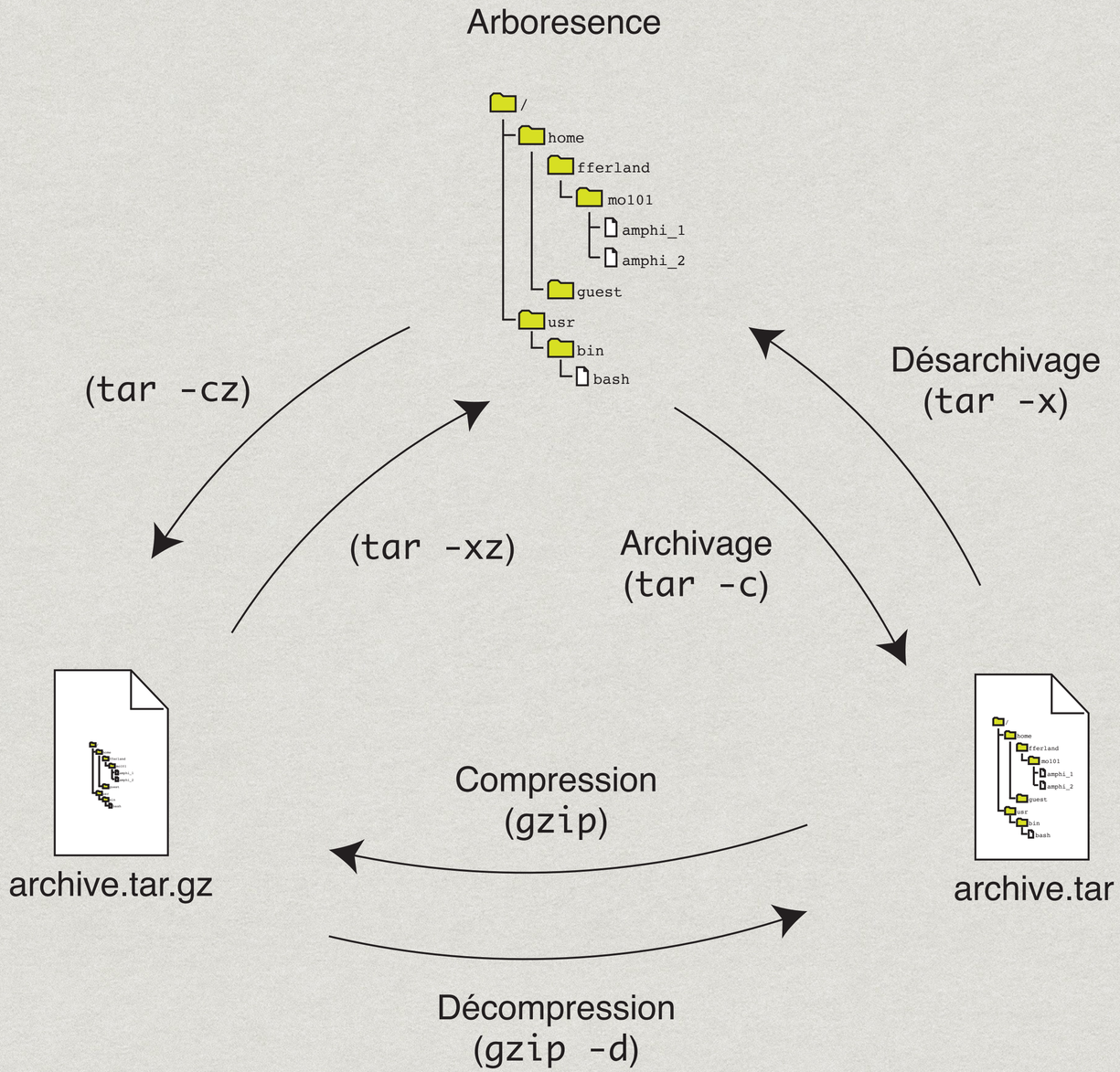
Archivage

- Permet d’enregistrer une arborescence en un seul fichier
- Utile pour transporter ou sauvegarder à long terme des fichiers
  - Particulièrement pour vos remises de travaux !
- L’arborescence peut être récupérée par le processus inverse: le désarchivage
- Ne compresse pas les fichiers - l’archive prend tout autant de place que l’arborescence originale

Compression

- Réduit la taille d'un ensemble de données en exploitant les motifs répétitifs.
- Est sans perte d'information (*lossless*) - À ne pas confondre avec des techniques comme le MP3, JPEG et autres algorithmes à perte d'information, perceptible ou non
- Plusieurs algorithmes disponibles, gz et bz2 étant les plus populaires sous UNIX/Linux.


Arboresence
/
home
fferland
mo101
amphi_1
amphi_2
guest
usr
bin
bash
Désarchivage
(tar -x)
Archivage
(tar -c)
archive.tar.gz
Compression
(gzip)
Décompression
(gzip -d)
archive.tar


Arboresence
/
home
fferland
mo101
amphi_1
amphi_2
guest
usr
bin
bash
(tar -cz)
Désarchivage
(tar -x)
(tar -xz)
Archivage
(tar -c)
Compression
(gzip)
archive.tar.gz
archive.tar
Décompression
(gzip -d)

A retenir archivage/compression

- Une seule commande pour archiver/compresser
  - **`tar -cvzf nom-archive.tar.gz dir file …`**
  - **`tar -cvjf nom-archive.tar.bz2 dir file …`**

- Une seule commande pour décompresser/désarchiver
  - **`tar -xvzf nom-archive.tar.gz`**
  - **`tar -xvjf nom-archive.tar.bz2`**

Les commandes utiles

- Créer un fichier vide (mais pas que)
  - **`touch nom-fichier`**
- Copier un fichier/répertoire
  - **`cp [-r]`**
- Déplacer un fichier/répertoire ou renommer
  - **`mv`**
- Supprimer un fichier/répertoire
  - **`rm [-rif]`**

- Afficher le contenu de fichiers
  - **`cat`**
- Lecteur de fichiers textes
  - **`less`**
- Connaitre le type de fichier
  - **`file`**

Les commandes utiles - 2

- Fichiers formatés en colonnes : extraction
  - **`cut [-d delim] [-f col]`**
- Fichiers formatés en colonnes : composition
  - **`paste [-d delim]`**
- Numéroter les lignes
  - **`nl`**
- Afficher le début d'un ficher
  - **`head -n`**
- Afficher la fin d'un fichier
  - **`tail -n`**
- Affichez des statistiques sur les fichiers textes
  - **`wc [-lmw]`**

PARTIE 4 : COMBINAISON DE COMMANDES

Processus

- Une **instance** d'un programme
  - Il peut y avoir plusieurs instances d'un même programme, chacune est un processus
- Chaque commande, qui inclut un programme, est également un processus
- Plusieurs processus peuvent fonctionner en parallèle

Processus, quelques états

- **Actif**
  - Le processus occupe de la mémoire et du temps de processeur
- **En pause**
  - Le processus ne fait plus de travail, mais demeure en mémoire
- **Interrompu**
  - Le processus libère toutes ses ressources et arrête son exécution. Il est ensuite détruit.

Processus, hiérarchie

- Les processus ont des relations parent-enfant(s)
- Une commande devient le processus enfant de la ligne de commande (bash)
- Interrompre le processus parent veut également dire interrompre tous ses processus enfants

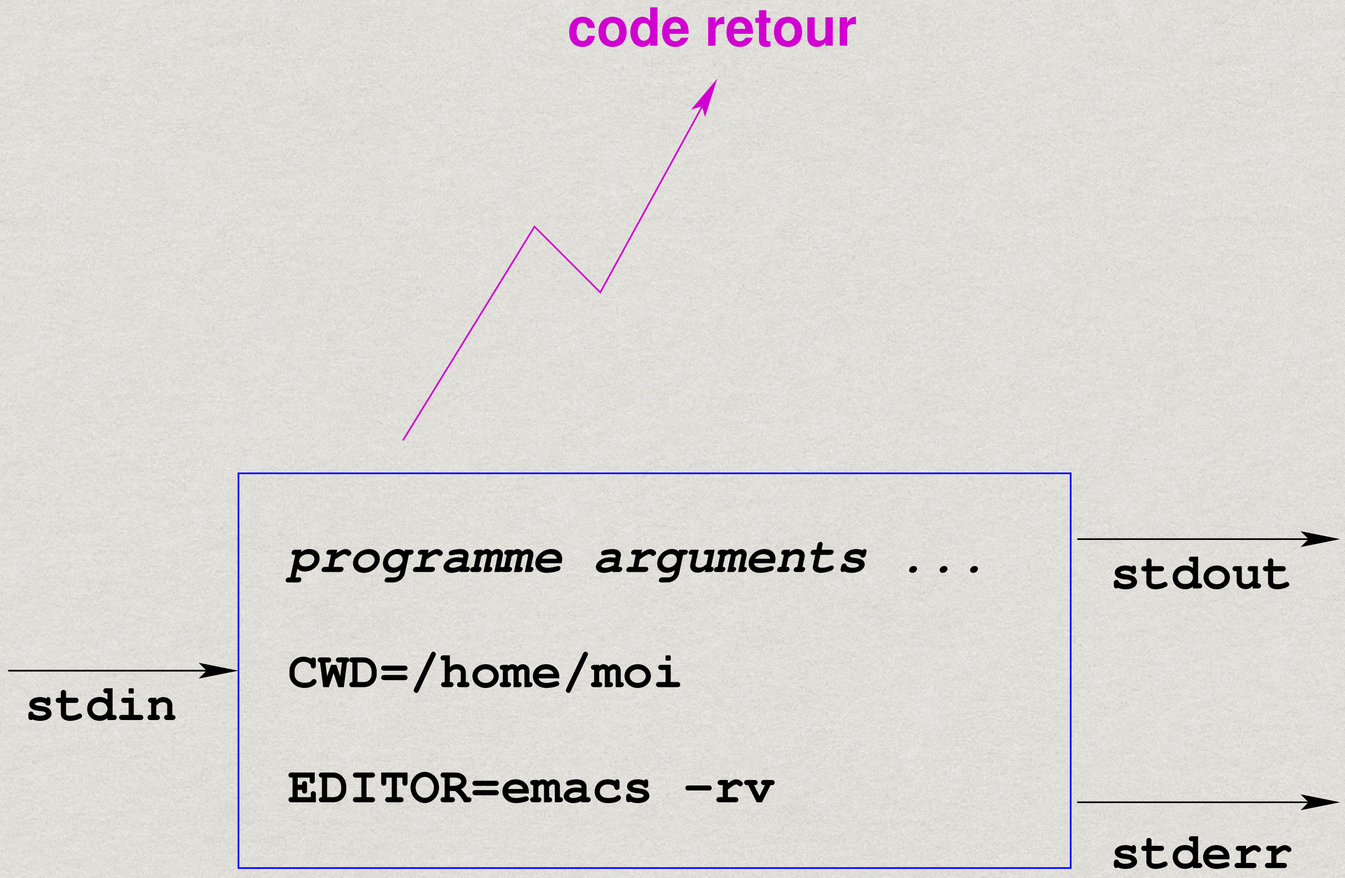
Processus, avant- et arrière-plan

- À la ligne de commande, une nouvelle commande prend l'avant-plan, c.-à-d., l'entrée (le clavier) et la sortie (l'écran du terminal)
- On peut mettre en pause un processus actif et en avant-plan (**`Ctrl-z`**)
  - Ceci rend l'avant-plan à la ligne de commande
- On peut également réactiver un processus en pause, mis en arrière-plan (**`bg`**)
  - Le processus continue son exécution, mais le contrôle est redonné à la ligne de commande. Par contre, l'affichage est partagé
- On peut remettre en avant-plan le dernier processus mis en pause (**`fg`**)

Entrée et sortie standard

- Chaque programme, et donc commande, a une entrée et une sortie standard
- Par défaut, l'entrée est le clavier, et la sortie l'écran du terminal
- Autant la sortie que l'entrée peuvent être redirigées vers un fichier ou un autre programme

Entrée et sortie standard



Source de l'image: C. Queinnec et V. Ménissier-Morain

Redirection de la sortie

- Un symbole: **>**
  - Exemple de commande:
    - `echo "test" > fichier_test`
- Un symbole: **>>**
  - Permet d'ajouter du contenu à la fin d'un fichier existant ou le crée s'il n'existe pas
- Un symbole : **2>**
  - Redirection de la sortie d'erreur
- Un symbole : **1>&2**
  - Ecrire sur la sortie erreur
    - `echo  "salut" 1>&2`

Le mise en séquence

- Un symbole : `;`
- Exécute une commande puis une autre
- Exemple
  - `echo “salut“; echo “les amis“`

- **Remarque** : pas de gestion des erreurs

Le mise en séquence 2

- Un symbole : **`&&`**
- Exécute une commande et si succès exécute l'autre
- Exemple
  - `echo "salut" && echo "les amis"`
  - `false && echo "les amis"`
- **Remarque** : succès si code de retour (`$?`) est nul

Le mise en séquence 3

- Un symbole : ||
- Exécute une commande et si échec exécute l'autre
- Exemple
  - `echo "salut" || echo "les amis"`
  - `rmdir dir || echo "Repertoire pas vide"`
- **Remarque** : échec si code de retour (`$?`) est différent de 0

Le tuyau

- Un symbole: |
- Redirige la sortie d'une commande vers l'entrée d'une autre
- Exemple de commande:
  - `sort fichier | uniq`
  - `cat fichier | wc -l`

Les commandes utiles

- Traduction de caractères
  - **`tr str1 str2`**
- Supprime les lignes multiples d'un fichier trié
  - **`uniq`**
- Trouver les chaines de caractères dans les fichiers binaires
  - **`strings`**
- Trier le contenu
  - **`sort [-n] [-d]`**
- Inverser les lignes d'un fichier
  - **`rev`**

PARTIE 5 : EXPRESSIONS RATIONNELLES

Expressions rationnelles

- En anglais **RegEx** (Regular Expressions)
- Définition de classes (motifs) de chaînes de caractères pour:
  - **filtrer** les chaînes (éliminer)
  - **extraire** de l'information (sélectionner)
  - **transformer** les chaînes (quand combiner avec d'autres fonctions)

Systèmes de RegEx

- Il y a plusieurs systèmes de RegEx dont
  - **BRE** (Basic Regular Expressions)
  - **ERE** (Extended Regular Expressions)
  - **PCRE** (Perl Compatible Regular Expressions)

- Pour les commandes Linux comme `grep` ou `sed`, par défaut BRE mais activation de ERE par option (`-E`)

ERE / BRE

- Les **méta-caractères** communs
  - N’importe quel caractère ‘`.`’
  - Un caractère parmi l’ensemble ‘`[ ]`’
  - Un caractère parmi le complémentaire de l’ensemble ‘`[ ^ ]`’
  - Début de chaîne ‘`^`’
  - Fin de chaîne ‘`$`’

ERE / BRE

- Les **méta-caractères** avec une syntaxe différente
  - Les groupements ou expressions marquées `()` ou `\(\)`
  - Accès à un groupe enregistré `\1, \2, …, \9`
  - Alternance `|` ou `\|`

ERE / BRE

- Les **méta-caractères** multiplicateur d'occurrences
  - `*` (0 à l'infini)
  - `+` ou `\+` (1 à l'infini)
  - `?` ou `\?` (0 ou 1 fois)

ERE / BRE

- Les classes de caractères

Standard POSIX	Equivalence	Signification
[:upper:]	[A-Z]	Majusucules
[:alpha:]	[[:upper:][:lower:]]	Majuscules et minuscules
[:alnum:]	[[:alpha:][:digit:]]	Chiffres, alphabétiques
[:digit:]	[0-9]	Chiffres
[:blank:]	[\t]	Espace et tabulation
[:space:]	[\t\n\r\f\v]	Tous les espaces

RegEx vs Shell

Shell	RegEx (ERE)	Signification
`?`	`.`	N'importe quel caractère
`*`	`.*`	Suite possiblement vide de caractères
`{a,bc}`	`(a|bc)`	Ensemble de chaines (alternative)

Attention : `?, . n'ont pas le même sens entre le Shell et les RegEx`

Exemples (ERE)

- "foo" : reconnait toutes les chaines de caractères contenant foo
- "(foo|foobar)" : toute chaîne de caractères qui contient le mot foo ou le mot foobar
- "^foo" : toute chaîne de caractères qui commencent par foo
- "foo$" : toute chaîne de caractères qui finit par foo
- "^foo$" : seul le mot foo est reconnu, rien ne peut le précéder ni le suivre dans la chaîne de caractères
- "f.o" : '.' désigne n'importe quel caractère, toute chaîne de caractéres qui contient f puis un caractère puis o
- "f[mnopq]o" : tous les caractères de la liste mnopq entre crochets autorisés à cet emplacement

Exemples (ERE)

- "f[m-p]o" : [m-p] tous les caractères de l’intervalle de caractères entre m et p
- "f[ˆa-IR-W0-9]o" : [ˆa-IR-W0-9] reconnaît tous les caractères autres que ceux de la liste de séquences indiquées
- "a*" : reconnaît 0 fois ou plus de a
- "(..)+" : reconnait 1 fois ou plus de .. soit un nombre pair non nul de caractères
- "f[o]+" ou "foo*" : reconnaît f suivi d'au moins 1 fois o
- "foo(bar)?" ou foo(|bar) : reconnait foo suivi de bar 0 ou 1 fois (alternative)

Exemples (ERE)

- Que détecte cette regex: ‘`dog`’
- Que détecte cette regex: ‘`^cat$`’
- Que détecte cette regex: ‘`^$`’
- Que détecte cette regex: ‘`^(From|Subject):`’
- Que détecte cette regex: ‘`^[+-]?[0-9]*\.?([0-9]+)$`'
- Que fait cette expression rationnelle '`^([^:]*:){2}[0-9]{4,}`' appliquée au fichier `/etc/passwd` ?

Exemples (ERE)

- Supposons un fichier texte,
  - donnez une regex qui trouve les lignes contenant quatre caractères « `a` »

    (a.*){4}

  - donnez une regex qui trouve les lignes contenant quatre occurrences du mot « `toto` »

    (toto.*){4}

  - donnez un regex qui trouve les lignes contenant quatre occurrences d'une même chaine non vide

    (..*)(.*\1){3}
-

PARTIE 6 : OUTILS EXPRESSIONS RATIONNELLES

La commande `grep`

- `grep` = g/re/p
- Utilitaire permettant de faire de la recherche de chaines de caractères
- Principales options
  - `-E` pour utiliser les ERE
  - `–color` pour afficher en couleur la chaine trouvée
  - `-o` affiche seulement les chaines trouvées
  - `-n` affiche le numéro de ligne des chaines trouvées
  - `-c` compte le nombre d'occurrences
  - `-w` cherche des mots

La commande `grep`

- Format d’appel de la commande
  - grep [options] pattern [fichier …]

La commande `sed`

- `sed` = Stream Editor
- Utilitaire permettant d'éditer un flux de caractères
- Principales options de la commande
  - `-n` n'affiche pas les lignes du flux
  - `-e commande` ajoute `commande` à la liste des traitements
  - `-i[ext]` permet des modifications en place avec sauvegarde sur `ext` renseigné.

La commande `sed`

- Format d'une commande
  - `[address[,address]]command[arguments]`
- Une adresse
  - Si un numéro correspond au numéro de ligne
  - Si `$` correspond à la dernière ligne
  - Si expression rationnelle (BRE) `/motif/` (adresse contextuelle) correspond aux lignes qui contiennent ce motif
  - Si **range** `adr1,adr2` correspond à toutes les lignes entre ligne qui est associée à `adr1` et une ligne associée à `adr2`

La commande `sed`

- Format d'une commande
  - `[address[,address]]command[arguments]`
- Principales commande
  - `!command` effectue la commande sur les lignes qui ne sont pas sélectionnées par les adresses
  - `d` supprime les lignes concernées
  - `p` affiche les lignes concernées
  - `s/motif/remplacement/` substitution de texte

Exercices `grep`

- Dans le fichier **`/etc/passwd`**
  - Trouvez les lignes contenant **`bash`**
  - Trouvez les utilisateurs dont le login contient le caractère **`s`**

Exercices `grep`

- Dans le fichier **`annuaire.txt`**
  - Trouvez les entrées contenant Mila
  - Trouvez les entrées dont le numéro de téléphone termine par 1
  - Trouvez les entrées dont le nom de famille commence par M
  - Trouvez les entrées dont le nom de famille commence par B et le numéro de téléphone termine par 7
  - Trouvez les entrées qui ont "Bertrand" comme nom de famille et le numéro de téléphone commence par 03
  - Trouvez les entrées de l'annuaire dont le nom de famille est "Durand" et dont le prénom est soit Jules, Louis ou Emma

Exercices `grep`

- Combinaison find et grep, à partir du répertoire **`racineLiesse2021`**
  - Affichez toutes les lignes qui contiennent le mot **`ode45`** dans les fichiers dont le nom contient la chaîne de caractères **`nonstiff-d`**

Exercices `sed`

- Dans le fichier **`annuaire.txt`**
  - Substituez tous les prénoms **`Gabriel`** par **`Gab`**
- Dans le fichier **`naissance.txt`**
  - Supprimez la première ligne du fichier
  - Supprimez toutes les lignes qui contiennent le prénom Nathan
  - Affichez les lignes 15 à 28 du fichier
  - Supprimez les lignes 5 à 10 du fichier

Exercices `sed`

- Dans le fichier **`index.html`**
  - Extraire la balise titre de la page HTML
  - Extraire toutes les balises `li` de la page HTML
  - Que font les commandes suivantes ?
    - **`sed -e 's/<\(.*\)>/<\U\1>/' index.html`**
    - **`sed -e 's/<\(.*\)>/<\u\1>/' index.html`**
  - Mettez en majuscule le titre (définit entre les balises `<h1>` et `</h1>`) de la page HTML

PARTIE 7 : SCRIPTS SHELL

Scripts bash

- Consiste à créer un programme à partir de commandes
- Permet d'automatiser des tâches répétitives
- Fichier texte contenant une suite de commandes auquel on a donné le droit d'exécution
- Constructions : mise en séquence, tuyau, structures de contrôle, fonctions

Un exemple

- Ajouter le texte « * Fichier 2021 *" comme entête à tout fichier ayant « 2021" dans son nom

Un exemple

```
#! /bin/bash

FICHIERS=`ls . | grep 2021`
N=0

for F in $FICHIERS; do
    echo "Modification de $F ..."
    echo '*Fichier 2021*' | cat - $F > temp;
    mv temp $F;
    N=$((N + 1))
done

echo "$N fichier(s) modifié(s)"
```

Script Shell

- Par convention la première ligne d'un script bash commence par "#! " suivi du chemin vers l'interprète de commandes, i.e., le shell. (shebang)
- Il est possible de manipuler des variables dans les scripts shell (également dans le terminal).
  - On donne une valeur à la variable "foo" en utilisant la syntaxe suivante `“foo=expression“` (sans espace autour du symbole =)
  - Le contenu de la variable "foo" est obetnu en ajoutant un symbole "$" devant le nom de la variable, c'est-à-dire, en écrivant "$foo".
- Un script shell est capable de gérer des paramètres sur la ligne de commande et il existe différentes variables associées à ces paramètres.

Variables du Shell

- `$#` nombre d'arguments sur la ligne de commande
- `$@` liste de tous les arguments de la ligne de commande
- `$0` nom du script
- `$1, $2, …, $9, ${10}, …` premier, second, etc argument de la ligne de commande

Un exemple

```
ferland@salle:~/mo101_3$ ls

a2021  b2021  test.sh

ferland@salle:~/mo101_3$ cat a2021

Ligne 1
Ligne 2

ferland@salle:~/mo101_3$ ./test.sh

Modification de a2021 ...
Modification de b2021 ...
2 fichier(s) modifié(s)

ferland@salle:~/mo101_3$ cat a2021

*Fichier 2021*
Ligne 1
Ligne 2
```

Un exemple : amélioration

```
#! /bin/bash

FICHIERS=`ls . | grep 2021`
N=0

for F in $FICHIERS; do
    if [ `head -1 $F | grep -c ‘Fichier 2021’` -eq 0 ] then
        echo "Modification de $F ..."
        echo '*Fichier 2021*’ | cat - $F > temp;
        mv temp $F;
        N=$((N + 1))
    fi
done

echo "$N fichier(s) modifié(s)"
```

CONCLUSION

Conclusion

- Le terminal est l'interface d'utilisation textuelle de l'OS Linux (il y a des outils similaires sous Windows, p. ex., powershell)
- Il donne accès à des outils puissants pour traiter les fichiers et les données
- Bash est un langage de programmation dont les constructions de base sont les commandes systèmes