

LES SYSTÈMES D'EXPLOITATION

MANIPULATION DES PROCESSUS SOUS UNIX/LINUX

🎓 3A - Cursus Ingénieurs - Dominante Informatique et Numérique
🏛️ CentraleSupélec - Université Paris-Saclay - 2025/2026



Idir AIT SADOUNE
idir.aitsadoune@centralesupelec.fr

PLAN

- Les systèmes Unix/Linux
- La redirection et la gestion de flux
- La synchronisation avec les pipes
- Les expressions régulières/rationnelles
- La gestion des processus
- La Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

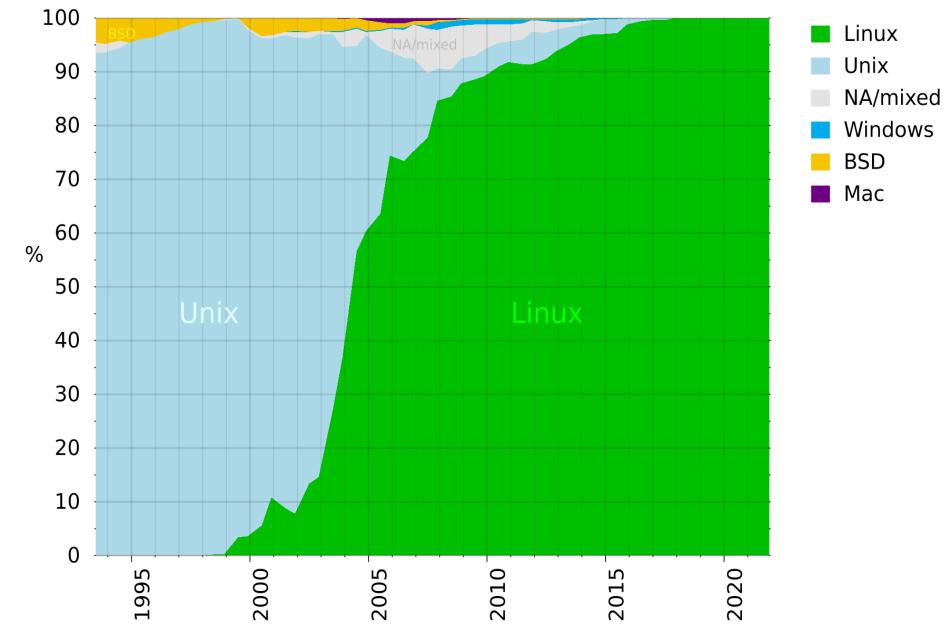
PLAN

- Les systèmes Unix/Linux
- La redirection et la gestion de flux
- La synchronisation avec les pipes
- Les expressions régulières/rationnelles
- La gestion des processus
- La Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

LES OS UNIX/LINUX

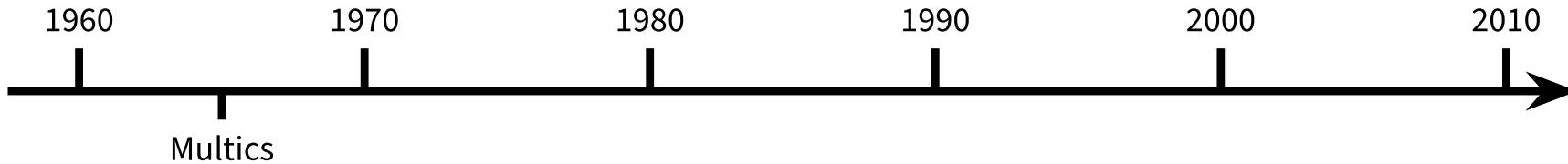
- Un **système d'exploitation** est un ensemble de **programmes informatiques** servant d'**interface** entre le **matériel** et les **applications/utilisateurs**.
 - ex. : **Windows** (... , XP, 7, ... , 10, 11), famille **Unix** (**Linux**, Mac-OS, ...).
- **Linux** (ou **GNU/Linux**) est un **OS open source** de type **Unix**
 - équipe une très **faible part** des ordinateurs personnels - **PC**
- **Linux** domine dans le **calcul intensif**
 - **100%** des calculateurs du **TOP 500**
→ **Classement 2020**
 - largement utilisé sur les serveurs, téléphones mobiles, systèmes embarqués, les superordinateurs, ...



L'HISTORIQUE D'UNIX



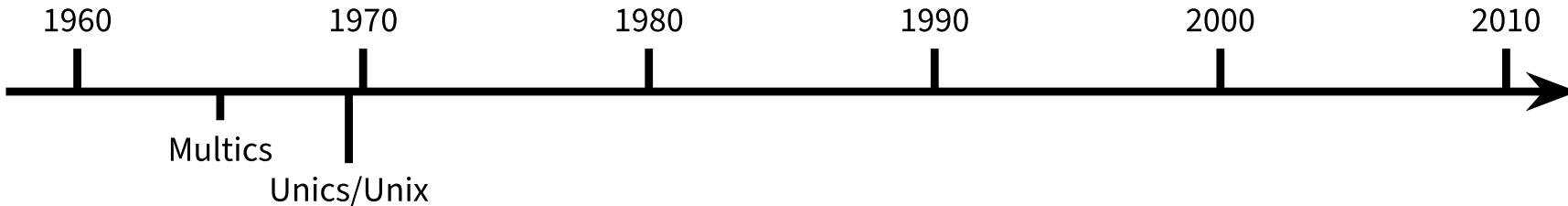
L'HISTORIQUE D'UNIX



MULTICS

- **MULT**plexed **I**nformation and **C**omputing **S**ervice
- laboratoires **Bell**, **MIT**, **General Electric**
- **temps partagé**, multi-utilisateurs
- système de fichier hiérarchique, segmentation et mémoire virtuelle
- système d'**invite de commande**, contrôle par un terminal distant

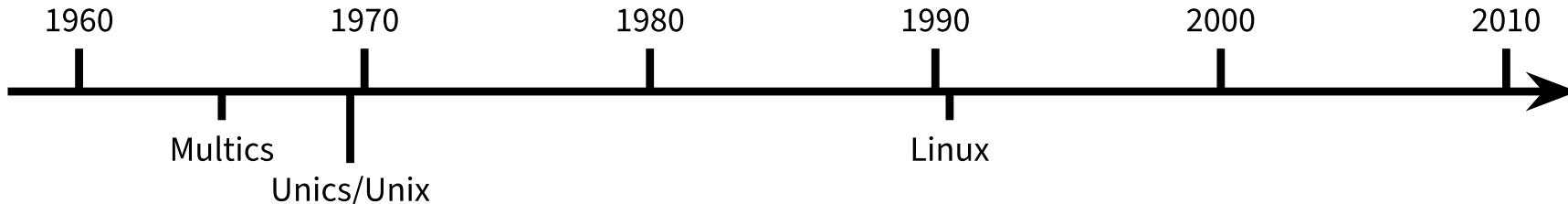
L'HISTORIQUE D'UNIX



UNICS or UNIX

- UNiplexed Information and Computing Service
- Ken Thompson, laboratoires Bell
- portable, multi-tâches, multi-utilisateurs
- système d'invite de commande utilisant le système de pipes
- principes publiés dans The Unix Programmer's manual en 1971

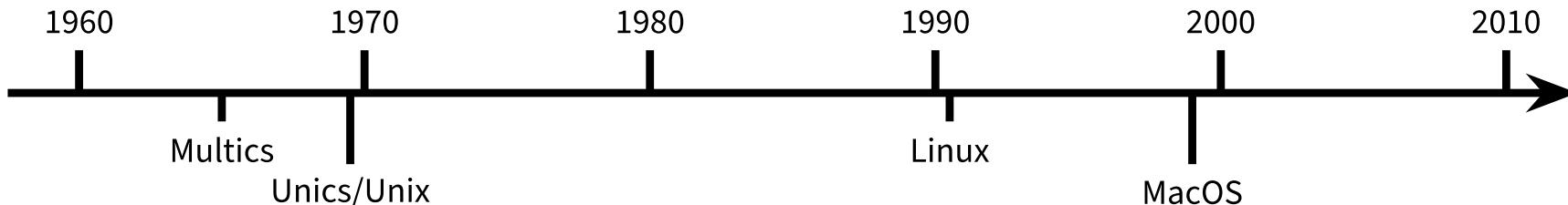
L'HISTORIQUE D'UNIX



Linux ou GNU/Linux

- créé en 1991 par Linus Torvalds
- basé sur Minix un clone d'Unix fondé sur un micro-noyau.
- noyau open-source publié sous licence GNU GPL
- plusieurs distributions : Debian, Fedora, Ubuntu, ...
- le shell Unix est toujours disponible, quelle que soit la distribution.

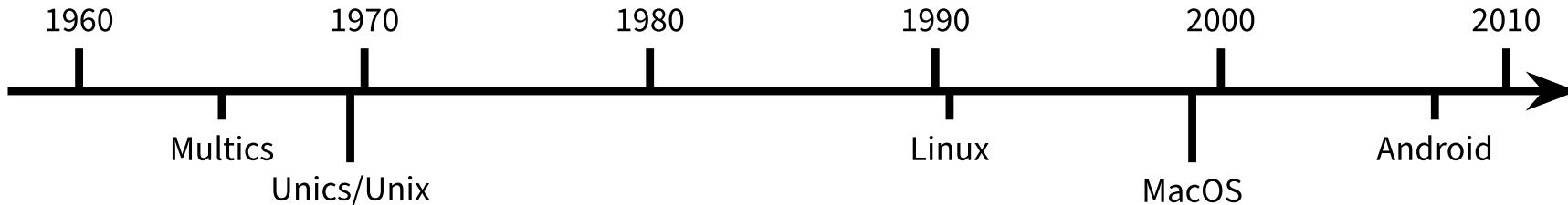
L'HISTORIQUE D'UNIX



MacOS

- un OS partiellement propriétaire annoncé par [Apple](#) en [1998](#)
- destiné exclusivement aux matériels de [Macintosh/Mac](#)
- son noyau [XNU](#) est un [noyau hybride](#) compatible avec la norme [POSIX](#).
- fondé sur le noyau [Mach](#) et sur l'implémentation [BSD](#) d'[Unix](#)
- les versions successives de [MacOS](#) ont reçu la [certification Unix](#)

L'HISTORIQUE D'UNIX



Android

- un **OS mobile** open source fondé sur le noyau **Linux**
- il a été lancé en **2007** pour les smartphones et les tablettes
- défini comme étant **une pile de logiciels** comprenant un noyau **Linux**
- l'**OS mobile** le plus utilisé (plus de **70%** de tous les appareils mobiles)
- la version **Android TV** est embarquée dans des télévisions, box TV, ...

1969

1971 to 1973

1974 to 1975

1978

1979

1980

1981

1982

1983

1984

Système de type Unix

1985

1986

1987

1988

1989

1990

1991

1992

1993

1994

1995

1996

1997

1998

1999

2000

2001 to 2002

2003

2004

2005 to 2007

2008 to 2009

2010

2011 to 2018

2019 to 2023

Système d'exploitation PDP-7 sans nom

Unix
Version 1 à 4Unix
Version 5 à 6BSD
1.0 à 2.0Unix
Version 7BSD
3.0 à 4.1

BSD 4.2

Unix
Version 8

BSD 4.3

BSD 4.3
TahoeBSD 4.3
Reno

BSD Net/2

386BSD

FreeBSD
1.0 à 2.2xBSD 4.4-Lite &
Lite Release 2FreeBSD
3.0 à 3.2FreeBSD
3.3 à 4.xDragonFly BSD
1.0 à 6.4NetBSD
1.4 à 9.xOpenBSD
2.3 à 7.2

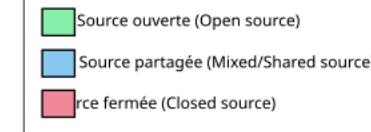
NetBSD 1.3

OpenBSD
1.0 à 2.2NetBSD
1.1 à 1.2

BSD Net/1

AIX
3.0 à 7.3

PWB/Unix

SunOS
1.1 à 1.1SunOS
1.2 à 3.0SunOS
4SCO Xenix
V/286SCO Xenix
V/386SCO Xenix
V/386SCO UNIX
3.2.4OpenServer
5.0 à 5.04OpenServer
5.0.5 à 5.0.7OpenServer
6.xUnixWare
1.x à 2.x
(System V R4.2)UnixWare
7.x
(System V R5)Solaris
2.1 à 9Solaris
10Solaris
11.0 à 11.4OpenSolaris
& dérivés
(Illumos, etc.)HP-UX
1.0 à 1.2HP-UX
2.0 à 3.0HP-UX
6 à 11.10HP-UX
11i v1 à 11i v3

1969

1971 to 1973

1974 to 1975

1978

1979

1980

1981

1982

1983

1984

1985

1986

1987

1988

1989

1990

1991

1992

1993

1994

1995

1996

1997

1998

1999

2000

2001 to 2002

2003

2004

2005 to 2007

2008 to 2009

2010

2011 to 2018

2019 to 2023

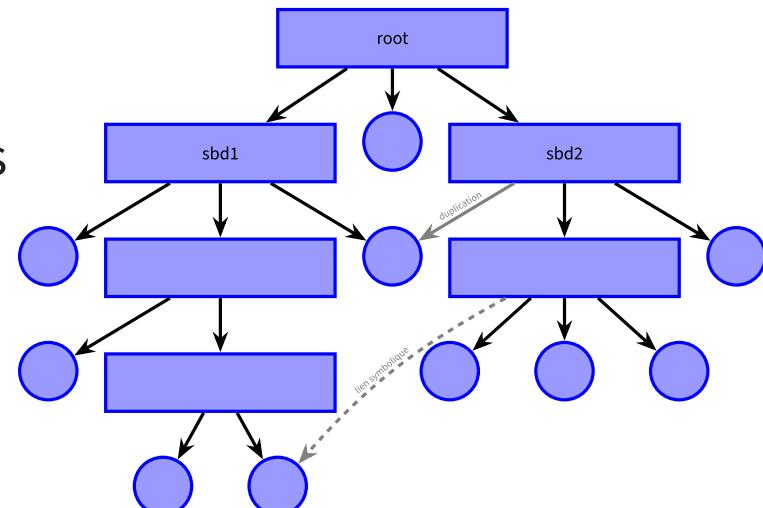
CARACTÉRISTIQUES DU SYSTÈME UNIX

- **Multi-tâches** (concurrentes et indépendantes)
- **Multi-utilisateurs** (dont l'administrateur ou le **root**)
 - système d'identification et droits d'accès aux fichiers
- **Chaînage** des processus par les tubes (**pipes**)
 - composition d'outils élémentaires pour des tâches complexes
- **Shell** est l'interface utilisateur du système d'exploitation.
 - **bash** : Bourne Again SHell (**sh** : shell historique de **Bourne**)
- L'interpréteur de commandes (**Shell**) intègre **un langage de programmation** (**variables, structures de contrôle, fonctions ...**)
 - **programmes interprétés** = fichiers de commandes = **shell-scripts**
 - création de commandes par l'utilisateur

LE SYSTÈME DE FICHIERS D'UNIX

Vu par l'utilisateur, le **système de fichier** d'**Unix** est structuré hiérarchiquement en un **arbre unique** constitué de :

- **noeuds** : répertoires (**directories**, dossiers ou **folders** sous **Windows**),
 - contiennent d'autres répertoires et des fichiers
- **feuilles** : fichiers (**files**),
 - des récipients contenant des données
 - les périphériques apparaissent également comme des fichiers



Découverte et manipulation à l'occasion du **TP 1**

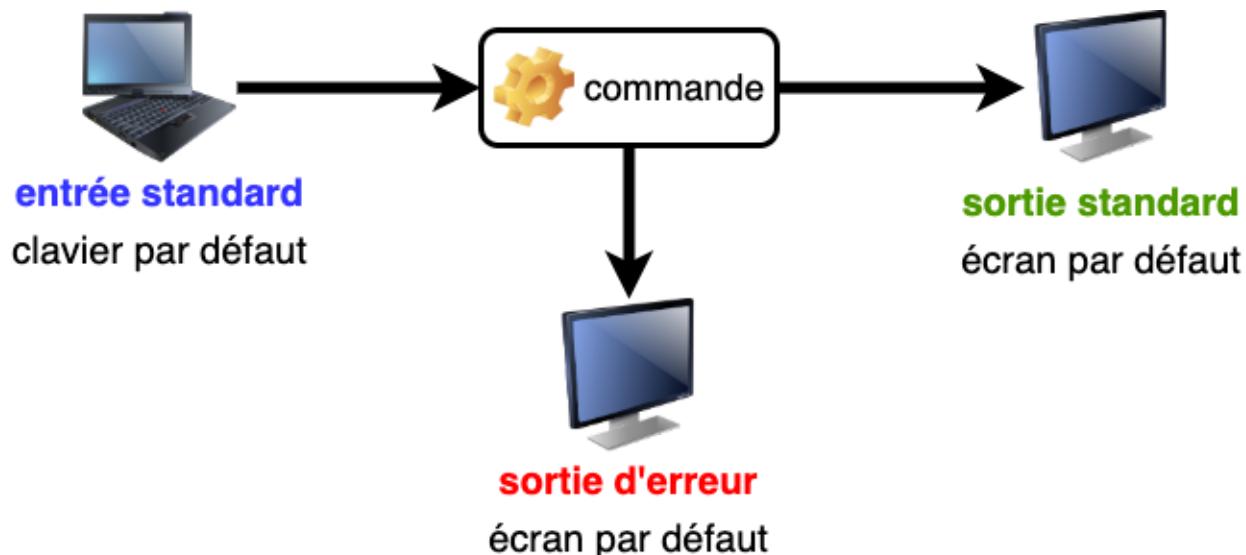
PLAN

- Les systèmes Unix/Linux
- La redirection et la gestion de flux
- La synchronisation avec les pipes
- Les expressions régulières/rationnelles
- La gestion des processus
- La Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

LES FLUX STANDARD

- En **Unix/Linux**, toute commande manipule **trois flux standards** :
 1. **entrée standard** (**stdin**, num = **0**) → données envoyées vers la commande
 2. **sortie standard** (**stdout**, num = **1**) → résultats produits par la commande
 3. **sortie d'erreur** (**stderr**, num = **2**) → messages d'erreur produits par la commande

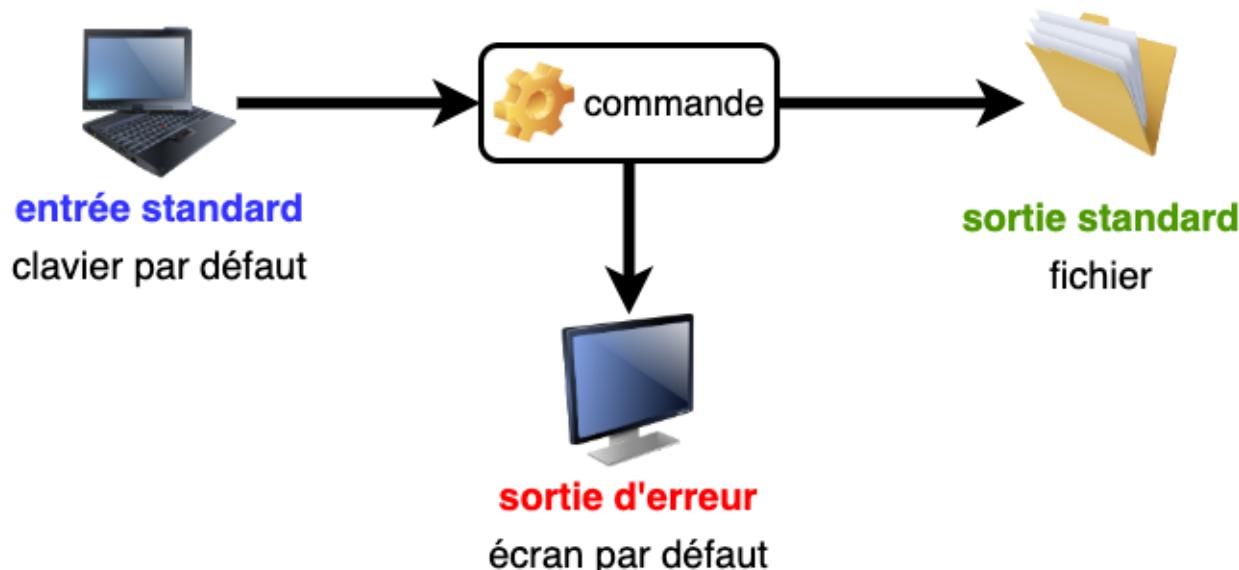


LA REDIRECTION



- Au lieu d'une saisie au clavier et d'un affichage à l'écran, stocker de façon permanente les informations d'entrée ou de sortie.
 - ⇒ rediriger les flux standards à partir ou vers des fichiers.
- Combiner des commandes pour effectuer des traitements complexes
 - ⇒ rediriger les flux standards à partir ou vers d'autres commandes.
- Grande souplesse et puissance du système Unix.

REDIRECTION VERS UN FICHIER



Un **nouveau fichier** est créé avec le contenu de la sortie

```
$ commande > fichier
```

La sortie est ajoutée à la fin d'un **fichier existant**

```
$ commande >> fichier
```

REDIRECTION VERS UN FICHIER

EXEMPLES

Le contenu du dossier courant dans un fichier

```
$ ls -l > liste.txt
```

Les 10 premières puis les 10 dernières lignes

```
$ head liste.txt > copy-liste.txt  
$ tail liste.txt >> copy-liste.txt
```

La liste des fichiers sources **Java**, puis celle des fichiers sources **C**

```
$ ls *.java > new-liste.txt  
$ ls *.c >> new-liste.txt
```

REDIRECTION VERS UN FICHIER

REMARQUE

- ✗ **Attention** → le **shell** interprète très tôt les redirections
 - ➡ ne pas rediriger la sortie vers le fichier d'entrée

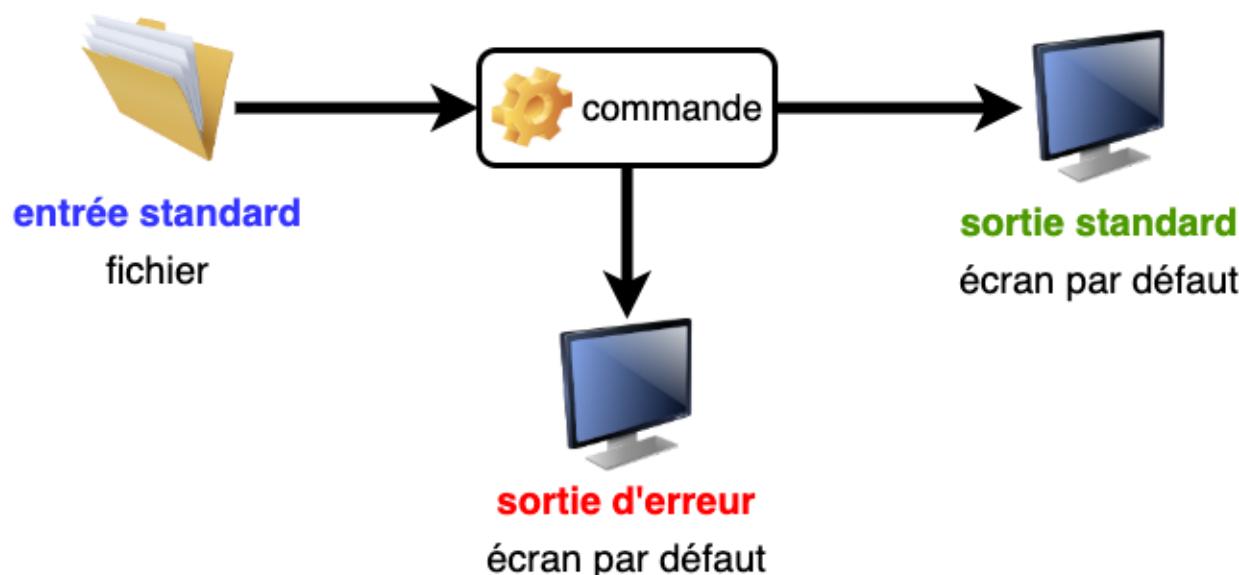
```
$ cat -n fichier.txt > fichier.txt
```

- ✓ **Solution**

- ➡ utiliser un fichier tampon

```
$ cat -n fichier.txt > tmp ; mv tmp fichier.txt
```

ENTRÉE DEPUIS UN FICHIER



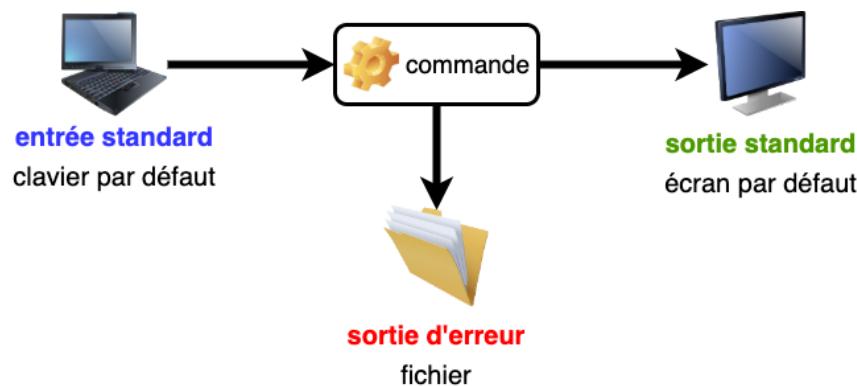
Le fichier doit exister au préalable

```
$ commande < fichier
```

Exemple → lecture des données d'entrée d'un script depuis un fichier

```
$ ./trier.sh < entrees.txt
```

LA SORTIE D'ERREURS VERS UN FICHIER



Un fichier est créé avec le contenu de la sortie d'erreurs

```
$ commande 2> fichier
```

La sortie d'erreurs est ajoutée à la fin d'un fichier existant

```
$ commande 2>> fichier
```

Exemple → sauvegarde des diagnostics d'une compilation

```
$ gcc programme.c 2> erreurs.txt
```

REGROUPEMENT DES FLUX



affiche à la console le contenu de **fichier_existant** et un **message d'erreur**

```
$ cat fichier_existant fichier_inexistant
```

affiche à la console un **message d'erreur**

```
$ cat fichier_existant fichier_inexistant > fichier.txt
```

n'affiche plus rien

```
$ cat fichier_existant fichier_inexistant > fichier.txt 2>&1
```

QUELQUES FICHIERS SPÉCIAUX

- Le répertoire **dev** contient des **fichiers spéciaux** gérant les **flux** entre l'**UC** et les **périphériques** (terminaux, imprimantes, disques, ...)
- **/dev/tty** : le terminal attaché à la connexion

```
$ tty  
/dev/ttys000
```

- **/dev/null** : fichier **poubelle** (vide) ou trou noir !
- **Exemple** → empêcher le flux d'erreur de s'afficher à l'écran.

```
$ commande 2> /dev/null
```

PLAN

- Les systèmes Unix/Linux
- La redirection et la gestion de flux
- La synchronisation avec les pipes
- Les expressions régulières/rationnelles
- La gestion des processus
- La Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

SYNCHRONISATION DE PROCESSUS

Deux méthodes pour **synchroniser deux processus** :

1. Méthode séquentielle avec fichier intermédiaire

```
$ cmd_1 > fichier  
$ cmd_2 < fichier  
$ rm fichier
```

2. Traitement à la chaîne en connectant les deux processus par un **Pipe**

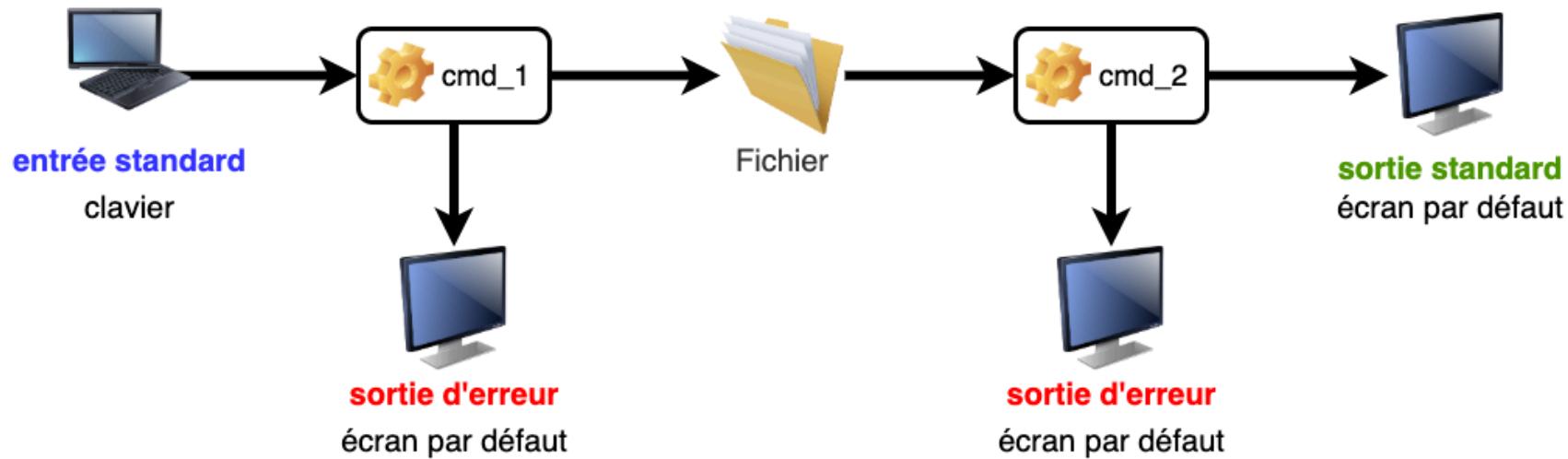
```
$ cmd_1 | cmd_2
```

- zone **mémoire**
- communication **synchronisée** entre les 2 processus
- plus **rapide** que le traitement séquentiel

MÉTHODE SÉQUENTIELLE

Méthode séquentielle avec fichier intermédiaire

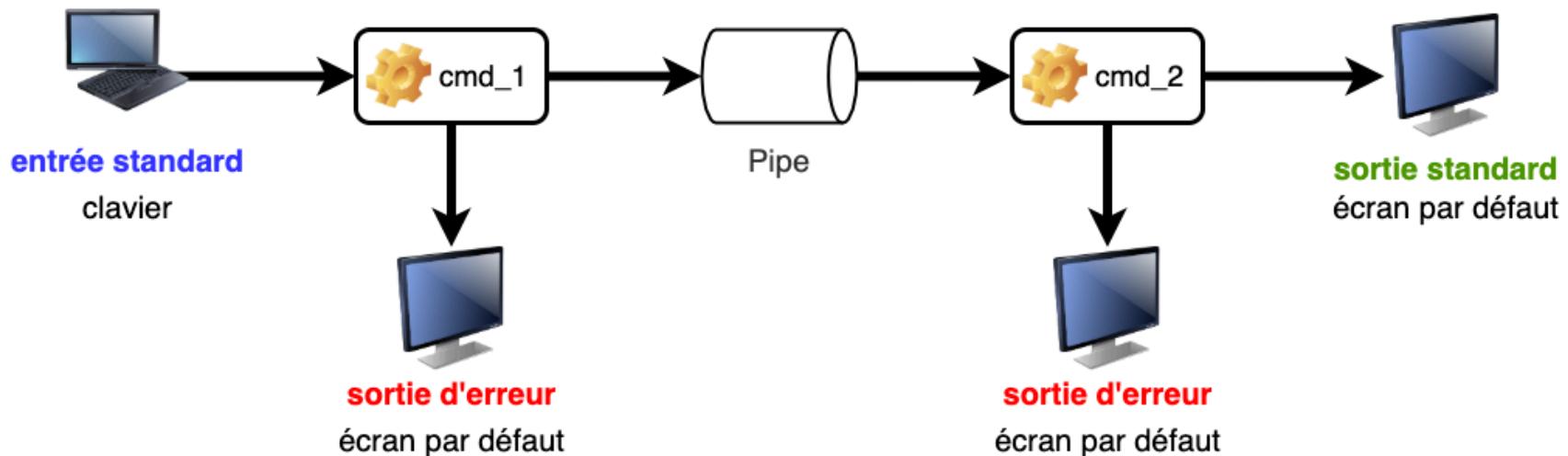
```
$ cmd_1 > fichier  
$ cmd_2 < fichier  
$ rm fichier
```



CHAÎNAGE AVEC TUBE

Traitement à la chaîne en connectant les deux processus par un Pipe

```
$ cmd_1 | cmd_2
```



CHAÎNAGE AVEC TUBE

EXEMPLE I

Affichage paginé de la liste des fichiers du répertoire courant

1. Exemple utilisant la méthode séquentielle

```
$ ls -l > liste.txt  
$ more liste.txt  
$ rm liste.txt
```

2. Exemple utilisant le chaînage avec tube

```
$ ls -l | more
```

CHAÎNAGE AVEC TUBE

EXEMPLE II

Affichage de la **12e** ligne du fichier **toto.txt**

1. Exemple utilisant la méthode séquentielle

```
$ head -n 12 toto.txt > tmp  
$ tail -n 1 tmp  
$ rm tmp
```

2. Exemple utilisant le chaînage avec tube

```
$ head -n 12 toto.txt | tail -n 1
```

CAS DE PLUSIEURS REDIRECTIONS

Avec **une seule commande**, l'ordre des redirections est indifférent

```
$ commande < entree > sortie
```

```
$ commande > sortie < entree
```

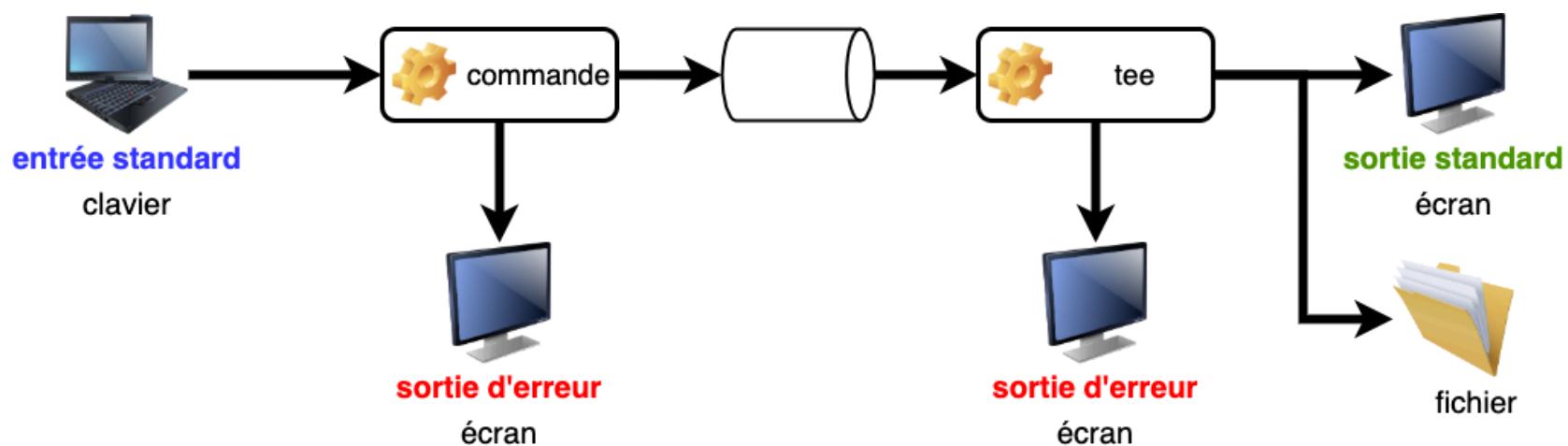
Avec **deux commandes** et **un tube**, ne pas détourner le flux

```
$ commande_1 < entree | commande_2 > sortie
```

DUPLICATION DE FLUX

La commande **tee** duplique le flux de son entrée standard vers le fichier passé en argument et la sortie standard.

```
$ commande | tee fichier
```



Exemple → conserver une trace du résultat intermédiaire d'un tube

```
$ cmd_1 | tee f_intermediaire | cmd_2
```

PLAN

- Les systèmes Unix/Linux
- La redirection et la gestion de flux
- La synchronisation avec les pipes
- Les expressions régulières/rationnelles
- La gestion des processus
- La Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

EXPRESSIONS RÉGULIÈRES OU RATIONNELLES

Recherche de chaînes de caractères correspondant à un pattern.

- syntaxe particulière pour décrire des motifs génériques
(expression régulière ou rationnelle)
- utilisées par les éditeurs **ex**, **vi** et **sed**, les filtres **grep** et **awk**,
ainsi que les langages **perl**, **Python**, **php**, **JavaScript** ...

LES CARACTÈRES SPÉCIAUX

- représente **un caractère quelconque** et un seul
- \ sert à **protéger le caractère qui le suit** pour empêcher qu'il ne soit interprété
- * représente **un nombre d'occurrences quelconque** (zéro ou plus) du caractère ou de la sous-expression qui précède

Remarque

ne pas les confondre avec les **wildcards** (* et ?), utilisés pour les **noms de fichiers**, qui sont interprétés par le **Shell**.

LES CARACTÈRES SPÉCIAUX

EXEMPLES

a^* un nombre quelconque de fois le caractère a
y compris une chaîne vide

aa^* un ou plusieurs fois le caractère a

$.^*$ un nombre quelconque de caractères quelconques
y compris une chaîne vide

$..^*$ au moins un caractère

$\backslash..^*$ un point suivi d'un nombre quelconque de caractères

$\backslash\backslash^*$ un nombre quelconque (y compris zéro) de backslash

LES ANCRES

Les ancre (anchor) permettent de spécifier qu'un motif est situé en début ou en fin de ligne.

^ en début de motif, représente le début de ligne

\$ en fin de motif, représente la fin de ligne

LES ANCRES

EXEMPLES

a une ligne commençant par un a

$^a.*b$ une ligne commençant par le template $a.*b$

$^$$ une ligne vide

$^.*$$ une ligne quelconque, y compris vide

$^..*$$ une ligne non vide

ENSEMBLE DE CARACTÈRES

- Un et un seul caractère choisi parmi un ensemble de caractères spécifiés entre crochets : `[ensemble_de_caracteres]`
- À l'intérieur d'un tel ensemble, les caractères spéciaux sont :
 - `[-]` : utilisé pour définir des intervalles
 - `[^]` : en tête pour spécifier le complémentaire de l'ensemble
 - `[]` : délimite la fin de l'ensemble, sauf s'il est placé au début
- On peut faire référence à des classes de caractères
 - `[:lower:]`, `[:upper:]`, `[:alpha:]`, `[:digit:]`, `[:alnum:]`

ENSEMBLE DE CARACTÈRES

EXEMPLES

[a0+]	un des caractères <i>a</i> , <i>0</i> ou <i>+</i>
[a-z]	une lettre minuscule
[a-z:;?!]	une lettre minuscule ou une ponctuation double
[0-9]	un chiffre
[^0-9]	n'importe quel caractère qui n'est pas un chiffre
[]-]	un <i>]</i> ou un signe <i>-</i>
[:digit:]	au lieu de [0-9]
[-+.[:digit:]]	un chiffre, un <i>.</i> , un <i>+</i> ou <i>-</i>

LE FILTRE **grep**

- **grep** : global regular expression print
- affiche les lignes qui contiennent un motif passé en paramètre

```
$ grep [options] motif [liste_de_fichiers]
```

où **motif** est une expression régulière

- Principales options :
 - **-i** : ignore la casse (majuscule/minuscule)
 - **-v** : inverse la sélection (affiche les lignes sans le motif)
 - **-l** : la liste des fichiers contenant le motif
 - **-n** : les lignes contenant le motif précédées de leur numéro
 - **-c** : les noms des fichiers et le nombre de lignes qui contiennent le motif

LE FILTRE **grep**

EXEMPLES

affiche la ligne de l'utilisateur **lefrere** dans le fichier de mots de passe

```
$ grep lefrere /etc/passwd
```

affiche les lignes commençant par **//** (commentaires)

```
$ grep '^//' application.java
```

affiche les lignes qui ne sont pas des commentaires

```
$ grep -v '^ *//' application.java
```

affiche la liste des sous-répertoires du répertoire courant

```
$ ls -l | grep ^d
```

PLAN

- Les systèmes Unix/Linux
- La redirection et la gestion de flux
- La synchronisation avec les pipes
- Les expressions régulières/rationnelles
- La gestion des processus
- La Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

GÉNÉRALITÉS

- **Processus** → tâche élémentaire identifiée par un **numéro (pid - process identifier)**
- **ps** afficher les processus de l'utilisateur associés au terminal

```
$ ps [options]
```

- **Principales options :**
 - **-e** → affiche tous les processus de tous les utilisateurs
 - **-U user_list** → sélectionne les processus appartenant à cette liste
 - **-f** → affiche une liste complète d'informations sur chaque processus
- **Principales informations** affichées par **ps** :

UID	PID	PPID	TTY	VSZ	CMD
user id	processus id	parent id	terminal	taille	commande

- Affichage **interactif** des processus avec la commande **top**

```
$ top
```

Contrôle et signaux

Caractères de contrôle (`Ctrl` ^) interprétés par le shell

- gestion des processus attachés au terminal et des flux d'E/S

`Ctrl L` clear efface l'écran

`Ctrl S` stop blocage de l'affichage à l'écran

`Ctrl Q` start déblocage de l'affichage à l'écran

`Ctrl D` eof fermeture du flux d'entrée (fin de session en shell)

`Ctrl C` int interruption du processus

`Ctrl Z` susp suspension du processus en cours

- La commande `stty` affiche les fonction des caractères de contrôle.

```
$ stty -a
```

- Un caractère de contrôle ne peut agir que sur le processus en interaction avec le terminal auquel il est attaché.

LA COMMANDE **kill**

- Intervenir sur un autre **processus** (ex. application qui ne répond plus)
➡ le désigner par son **numéro** et lui **envoyer un signal**
- **kill** envoie par défaut **un signal de terminaison**

```
$ kill -s TERM pid
```

- sinon **un signal de mise à mort**

```
$ kill -s KILL pid
```

PROCESSUS EN ARRIÈRE PLAN

Système **UNIX** multi-tâches :

- commandes longues non-interactives **en arrière-plan (tâche de fond)**
- **garder la main** pour d'autres commandes (mode **asynchrone**)

```
$ commande &
```

Gestion des processus en arrière-plan :

jobs affiche la liste des processus en arrière-plan

fg passe le job courant en premier plan

fg num passe le job **num** en premier plan

bg passe le job courant en arrière-plan

PROCESSUS EN ARRIÈRE PLAN

EXEMPLES

- **top au premier plan**
 - on perd la main dans la fenêtre initiale

```
$ top
```

- terminer ce processus par **ctrl C**
- **top en arrière plan**
 - on conserve la main dans la fenêtre initiale

```
$ top &
```

- terminer le processus **top** par **fg** puis **ctrl C** ou par **kill -s** suivi de **KILL pid**

Remarque : si on a oublié le **&**, on utilise **ctrl Z** pour suspendre le processus, puis **bg** pour le passer en arrière-plan

CODE DE RETOUR

Toute commande **UNIX** renvoie un code numérique en fin d'exécution

- valeur de retour (cf. `exit()` dans `main` en C)
- statut de fin (`return status`) accessible via `$?`

✓ **Code de sortie = 0** → la commande s'est bien déroulée

```
$ cd /bin  
$ echo $?  
0
```

✗ **Code de sortie ≠ 0** → la commande s'est mal déroulée

```
$ cd /introuvable  
$ echo $?  
1
```

COMBINAISON DE COMMANDES

```
$ cmd_1 && cmd_2
```

- La première commande est exécutée.
- Si elle réussit ($\$? = 0$), la seconde commande est exécutée.
- **Exemple** → Si la **compilation** d'un code source **Java** s'effectue **sans erreurs**, alors on lance son **exécution**.

```
$ javac Application.java && java Application
```

PLAN

- Les systèmes Unix/Linux
- La redirection et la gestion de flux
- La synchronisation avec les pipes
- Les expressions régulières/rationnelles
- La gestion des processus
- La Synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

CE QU'IL FAUT RETENIR

- Unix est un OS multi-tâches, multi-utilisateurs
- Unix est à la base de plusieurs OS modernes
- Le shell bash est une interface utilisateur utilisant un interpréteur de commandes
- Une grande souplesse et une puissance basées sur la redirection et le Pipe
- Mécanisme de recherche basé sur les expressions régulières
- Commandes de gestion et de synchronisation des processus

MERCI

[Version PDF des slides](#)

[Retour à l'accueil](#) - [Retour au plan](#)