



université
PARIS-SACLAY



CentraleSupélec

SYSTÈMES D'EXPLOITATION

MANIPULATION DES PROCESSUS SOUS UNIX

🎓 3A - Coursus Ingénieurs - Dominante Informatique et Numérique

🏛️ CentraleSupélec - Université Paris-Saclay - 2024/2025



Idir AIT SADOUNE

idir.aitsadoue@centralesupelec.fr

PLAN

- Le système Unix
- Redirection
- Tubes ou pipes
- Expressions régulières ou rationnelles
- Gestion des processus
- Synthèse

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

PLAN

- > Le système Unix
- > Redirection
- > Tubes ou pipes
- > Expressions régulières ou rationnelles
- > Gestion des processus
- > Synthèse

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

SYSTÈMES D'EXPLOITATION

- Ensemble de **programmes informatiques** servant d'**interface** entre le **matériel** et les **applications utilisateurs**.
 - **ex. :** **Windows** (XP, 7, ...,10), famille **Unix** (**Linux**, **Mac-OS**, ...).
- **Linux** domine dans le **calcul intensif**
 - plus de **97 %** des calculateurs du **TOP 500** → **Classement 2019**

L'HISTORIQUE D'UNIX



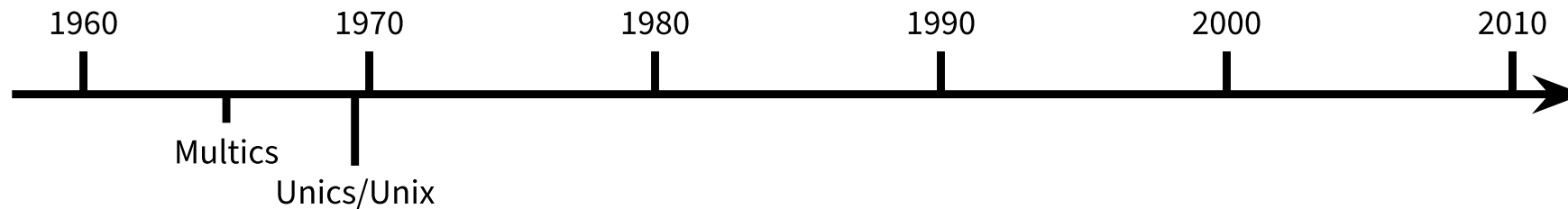
L'HISTORIQUE D'UNIX



MULTICS

- **MULT**iplexed **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

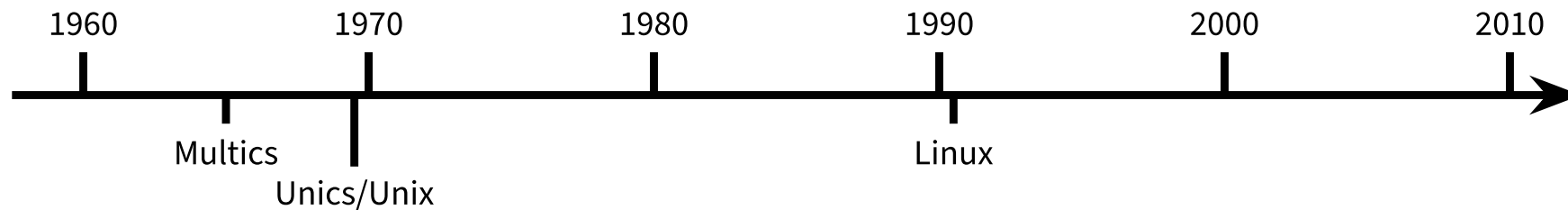
L'HISTORIQUE D'UNIX



UNICS or UNIX

- **UN**iplexed **I**nformation and **C**omputing **S**ervice
- **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**

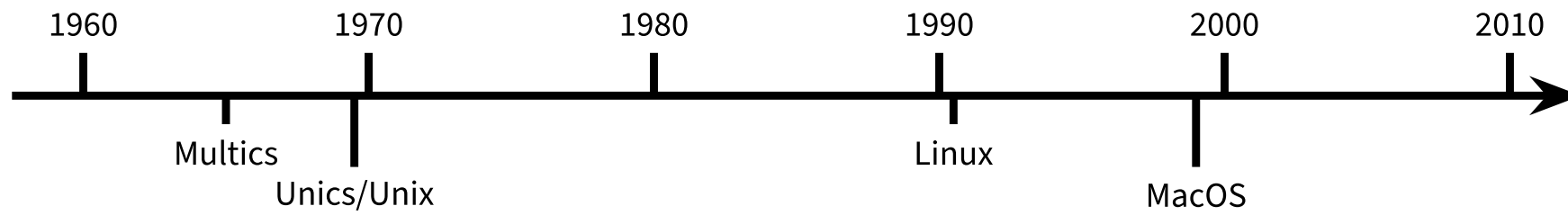
L'HISTORIQUE D'UNIX



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, ...

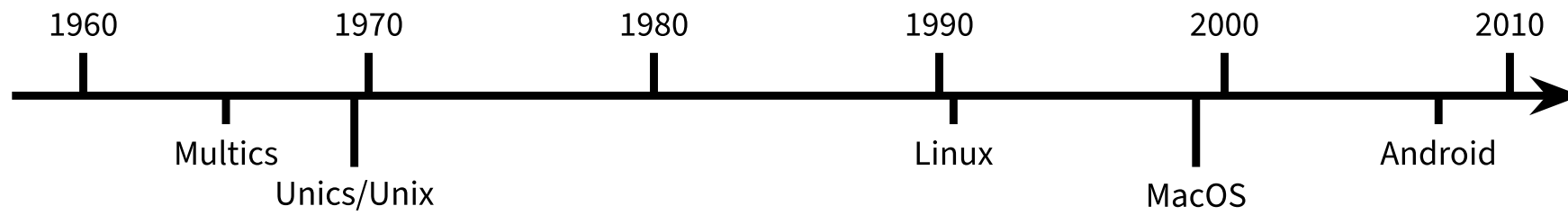
L'HISTORIQUE D'UNIX



MacOS

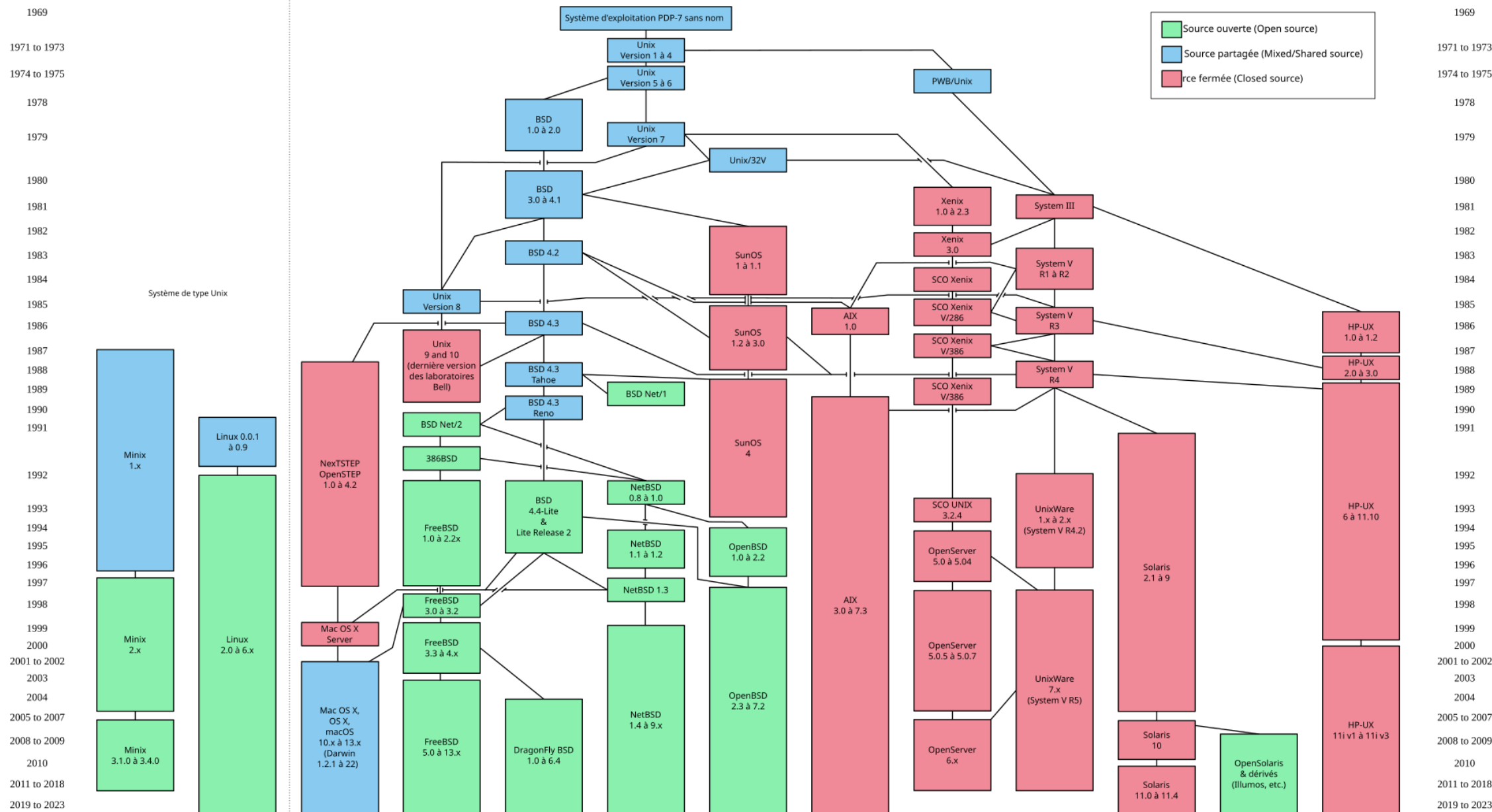
- fondé sur le noyau [Mach](#) et sur l'implémentation [BSD](#) d'[Unix](#)
- adapté à l'architecture matérielle de [Macintosh](#) ou de [Mac](#)

L'HISTORIQUE D'UNIX



Android

- fondé sur le noyau **Linux**
- système d'exploitation pour les **appareils mobiles**



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 **SH**ell (**sh** : shell historique de **Bourne**)

CARACTÉRISTIQUES DU SHELL

- **Shell** est l'interface utilisateur du système d'exploitation.
 - **bash** : Bourne Again **SH**ell (**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 réceptacles contenant des données
 - les périphériques apparaissent également comme des fichiers

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

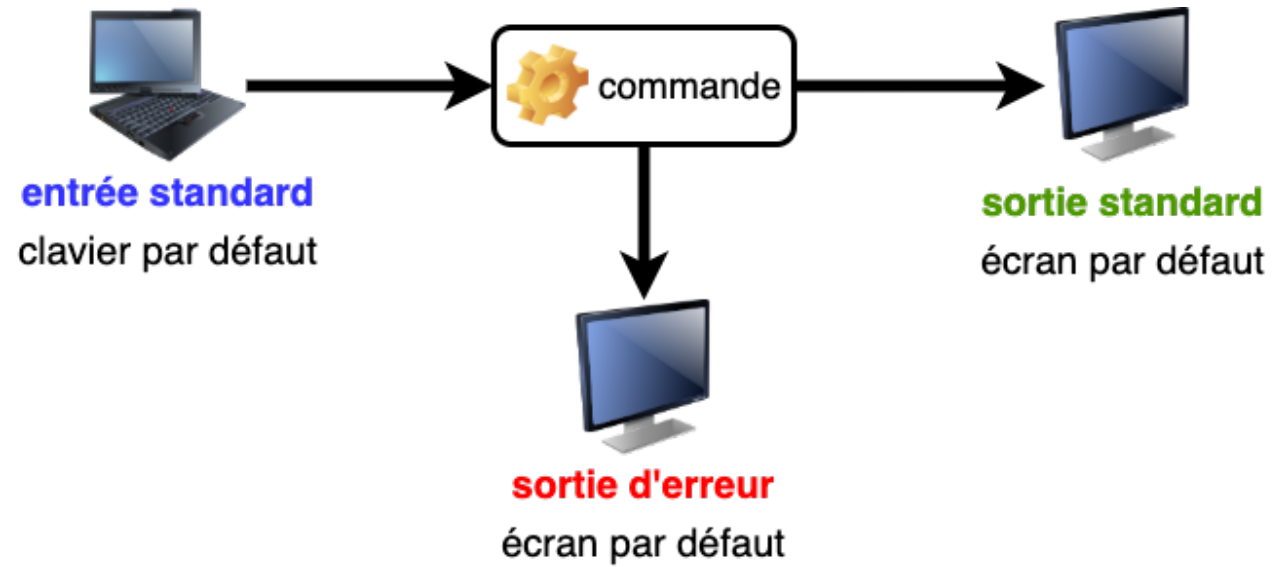
PLAN

- Le système Unix
- Redirection
- Tubes ou pipes
- Expressions régulières ou rationnelles
- Gestion des processus
- Synthèse

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

LE FLUX STANDARD

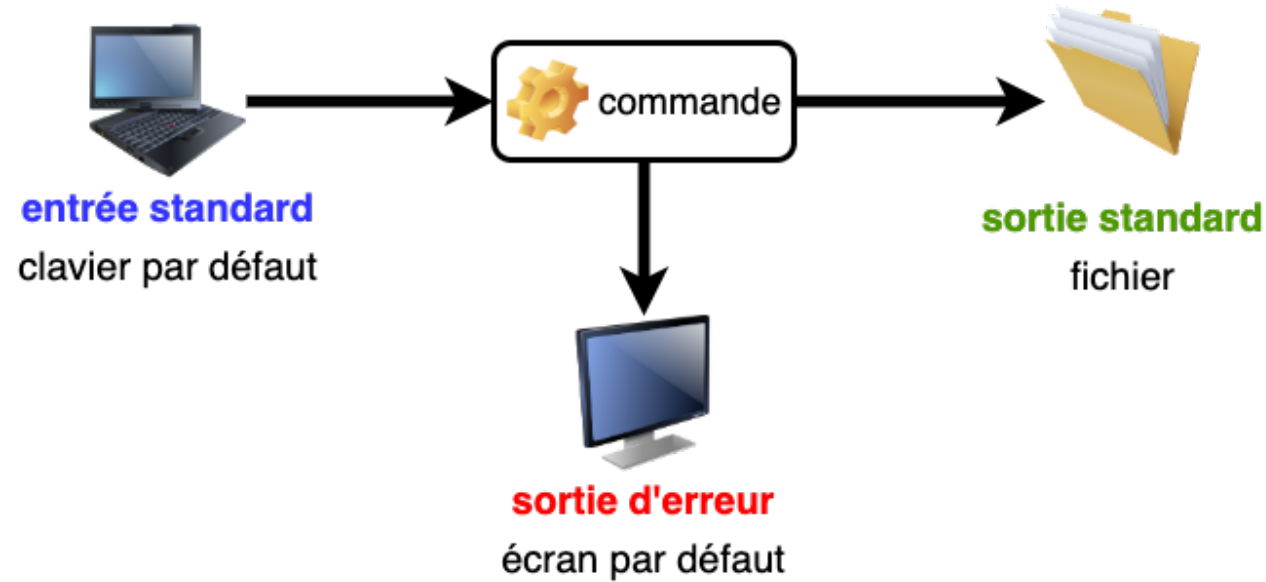
Commande Unix → trois flux standards de données :



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
```

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

Attention : le **shell** interprète très tôt les redirections

➡ ne pas rediriger la sortie vers le fichier d'entrée

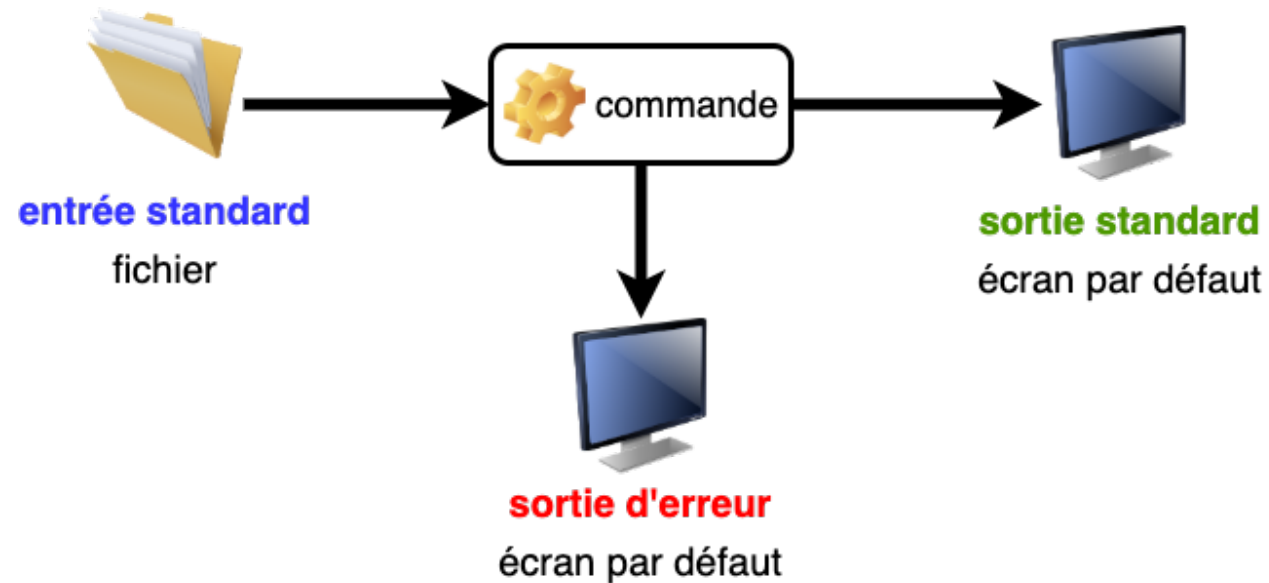
```
$ cat -n fichier.txt > fichier.txt
```

Solution :

➡ utiliser un fichier tampon

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

ENTRÉE DEPUIS UN FICHIER



Le fichier doit exister au préalable

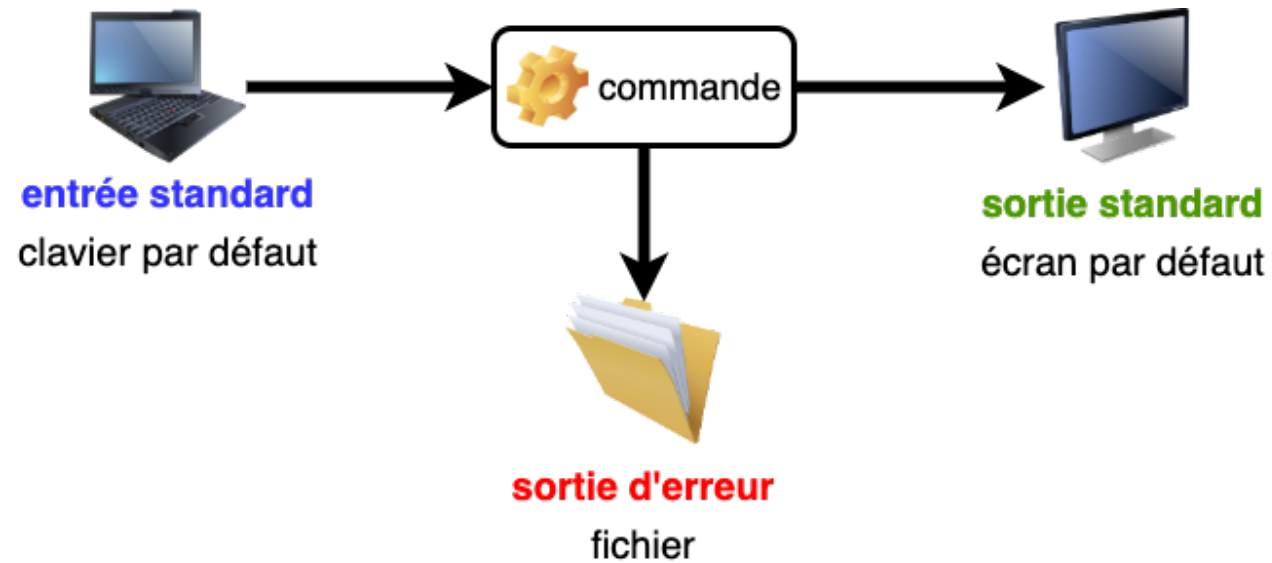
```
$ commande < fichier
```

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

```
$ cat fichier_existant fichier_inexistant
```

Affiche **le contenu de** **fichier_existant** et un **message d'erreur**

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

Affiche un **message d'erreur**

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

N'affiche rien

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

- Le système Unix
- Redirection
- Tubes ou pipes
- Expressions régulières ou rationnelles
- Gestion des processus
- Synthèse

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

TUBES OU PIPES

Deux méthode pour **synchroniser deux processus** :

1. **Méthode séquentielle** avec fichier intermédiaire

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

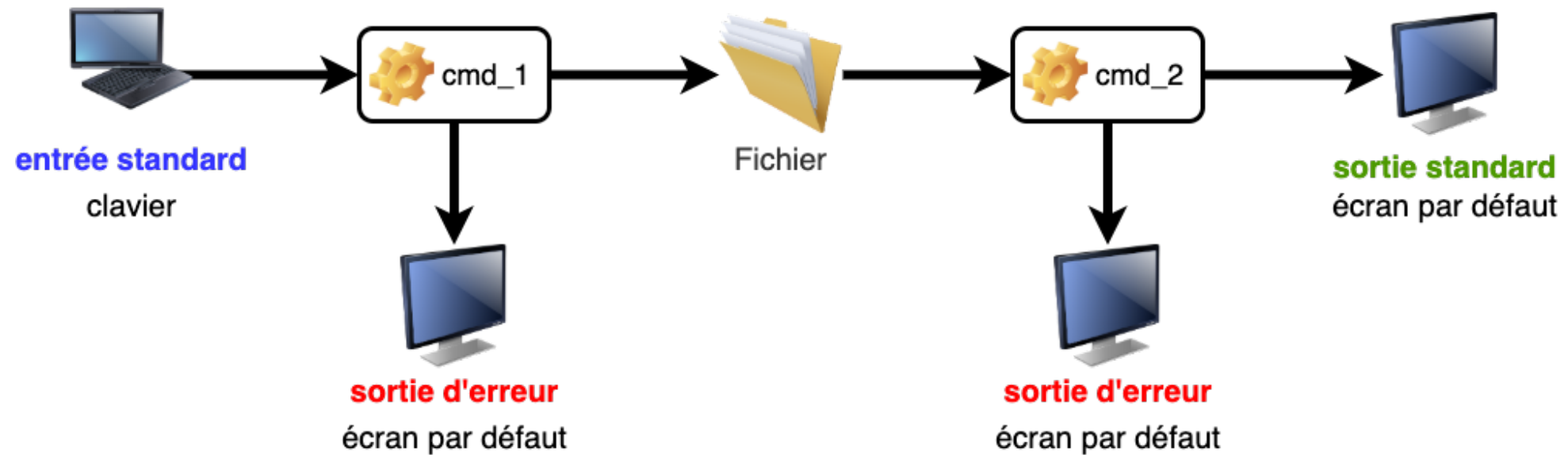
2. **Traitement à la chaîne** en connectant les deux processus par un **Pipe**

```
$ cmd_1 | cmd_2
```

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

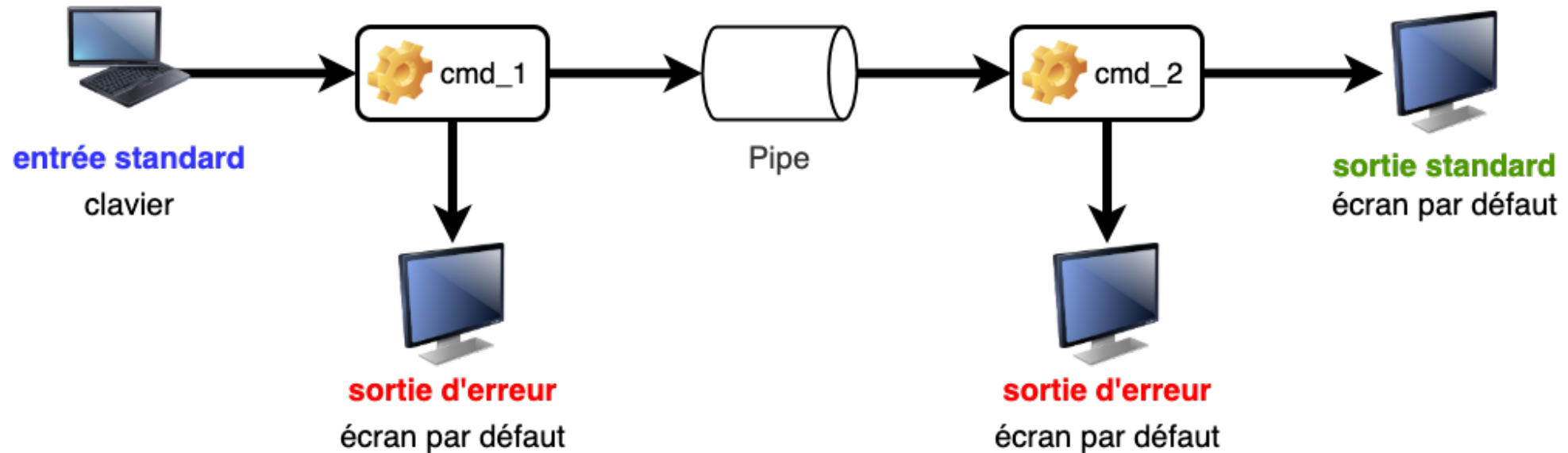
MÉTHODE SÉQUENTIELLE

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



CHAÎNAGE AVEC TUBE

```
$ cmd_1 | cmd_2
```



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
```

EXEMPLE II

Affichage de la 12^e 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
```

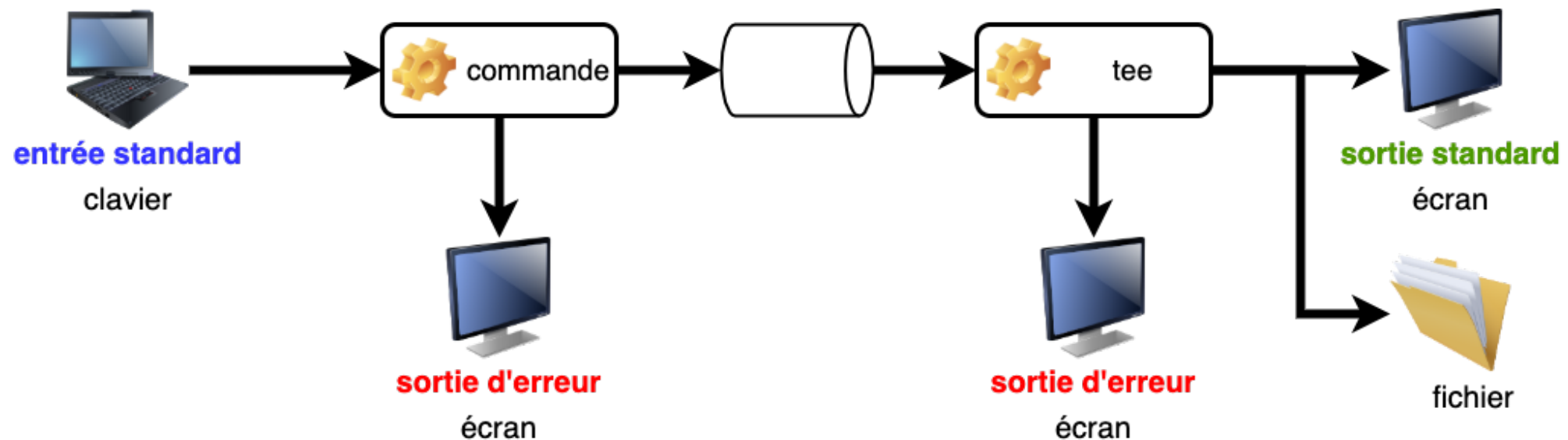
Avec **deux commandes** et **un tube**, ne pas détourner le flux

```
$ 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

- Le système Unix
- Redirection
- Tubes ou pipes
- Expressions régulières ou rationnelles
- Gestion des processus
- 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**.

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 caractère quelconque

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

LES ANCRES

Les ancres (**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

EXEMPLES

$\wedge a$ une ligne commençant par un a

$\wedge a.*b$ une ligne commençant par un mot commençant par un a
et finissant par b

$\wedge \$$ une ligne vide

$\wedge .* \$$ une ligne quelconque, y compris vide

$\wedge ..* \$$ 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:]`

EXEMPLES

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

LE FILTRE **grep**

- **grep** : **g**lobal **r**egular **e**xpression **p**rint
- 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

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 dont le premier caractère non blanc est **{**

```
$ grep '^ *{' application.java
```

affiche les lignes qui ne sont pas des commentaires

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

EXEMPLES

affiche les lignes qui ne comportent pas que des blancs

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

affiche la liste des sous-répertoires du répertoire courant

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

PLAN

- Le système Unix
- Redirection
- Tubes ou pipes
- Expressions régulières ou rationnelles
- Gestion des processus
- Synthèse

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

GÉNÉRALITÉS

- **Processus** → **tâche élémentaire** identifiée par un **numéro unique** (**pid** - **p**rocess **i**dentifier)
- **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

GÉNÉRALITÉS

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
--------	------	----------------------------------

CONTRÔLE ET SIGNAUX

- La commande `stty` gère l'affectation des **caractères de contrôle** à certaines fonctions

```
$ 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

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 \neq 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

- Le système Unix
- Redirection
- Tubes ou pipes
- Expressions régulières ou rationnelles
- Gestion des processus
- 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 basée sur 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](#)