Introduction à l'environnement UNIX

R&I 1

Antoine Rollet

Bordeaux INP - Enseirb-Matmeca

Unix/Bash

Introduction

Documentation

Lien vers le cours :

https://rollet.zzz.bordeaux-inp.fr/

Un peu de documentation :

https://rollet.zzz.bordeaux-inp.fr/IntroUnix/documentation/refcard-unix.pdf
https://rollet.zzz.bordeaux-inp.fr/IntroUnix/documentation/refcard-bash.pdf

Système d'exploitation (SE)

☞ Operating System (OS) en anglais

Programme complexe réalisant les fonctionnalités de base indispensables à tous les utilisateurs (interactions avec le système, les périphériques)
Ex. de SE : Windows, MacOS, Linux, DOS, etc. → d'autres ?

- SE mono-processus, mono-utilisateur : exécute un programme à la fois avec un seul utilisateur (DOS [1])
- SE multi-processus, mono-utilisateur : plusieurs tâches simultanées correspondant à des programmes différents. Répartition des ressources à tour de rôle entre les applications. (Windows 95)
- SE multi-processus, multi-utilisateurs. Plusieurs utilisateurs peuvent lancer en même temps leurs programmes sur la même machine. (exemple : Unix)

Les stations de travail à l'Enseirb-Matmeca sont multi-processus, multi-utilisateurs

Processus : toute occurrence en mémoire d'un programme en cours d'exécution

1. Disk Operating System

Systèmes de fichiers

Système de fichiers = structure logique permettant au système d'exploitation d'accéder aux fichiers d'un système de stockage → organise les données des fichiers dans le disque

Réalisé lors d'un **formatage**.

Chaque SE gère différents systèmes de fichiers, et en utilise un par défaut. Les plus connus sont :

- NTFS (remplace FAT32) : système de fichiers Microsoft pour Windows
- FAT32 (remplace FAT16) : (ancien) système de fichiers Microsoft pour Windows

  Limitations :
  - → un fichier ne peut pas dépasser 4 Go
  - → Partition doit être inférieure à 8 To
- ext4 (remplace ext3) : système de fichiers par défaut de Linux
- APFS (remplace HFS+) : système de fichiers par défaut de MacOSX

Compatibilités :

- Windows ne sait pas gérer ext4 ou APFS
- Linux gère pas mal de systèmes de fichiers, dont NTFS et FAT32

Systèmes de type Unix

- Plusieurs distributions (versions) par ex : Knoppix, Ubuntu, Linux Mint, Fedora, Gentoo, Debian, Suse, Redhat, Slackware, Alma . . .
- Modèle client/serveur
- Plusieurs terminaux/utilisateurs se connectent à une machine, locale ou distante grâce au réseau

Quelques caractéristiques d'UNIX :

- système à temps partagé : exécution concurrente de processus, ordonnancement
- système multi-utilisateur : compte, connexion (login/password), environnement personnalisé
- partage des ressources : arborescence globale des fichiers, mécanismes de protection
- gestion des communications : entre processus, entre utilisateurs
- simple, portable, extensible

⚠ ne pas confondre Unix et Linux :

Systèmes de type Unix

- Plusieurs distributions (versions) par ex : Knoppix, Ubuntu, Linux Mint, Fedora, Gentoo, Debian, Suse, Redhat, Slackware, Alma . . .
- Modèle client/serveur
- Plusieurs terminaux/utilisateurs se connectent à une machine, locale ou distante grâce au réseau

Quelques caractéristiques d'UNIX :

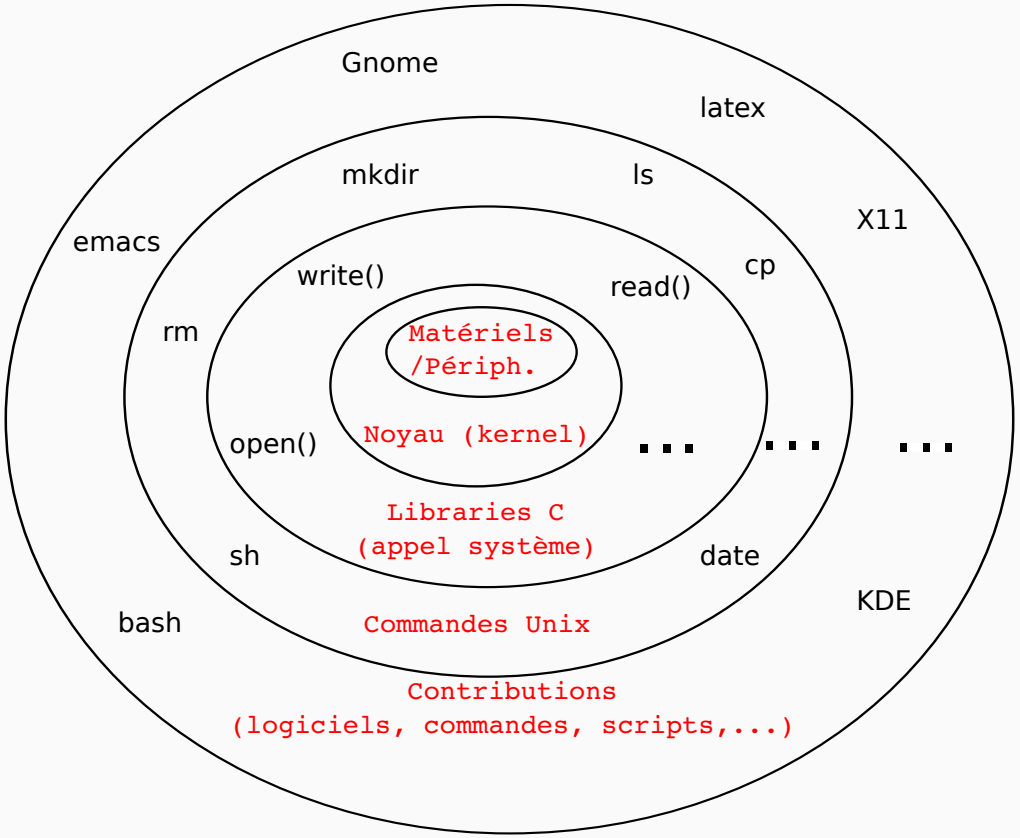
- système à temps partagé : exécution concurrente de processus, ordonnancement
- système multi-utilisateur : compte, connexion (login/password), environnement personnalisé
- partage des ressources : arborescence globale des fichiers, mécanismes de protection
- gestion des communications : entre processus, entre utilisateurs
- simple, portable, extensible

⚠ ne pas confondre Unix et Linux :

- Unix = système d'exploitation (avec les caractéristiques ci-dessus) développé par AT&T dans les années 60. Beaucoup de dérivés → systèmes de type Unix (famille Unix)
- Linux = système d'exploitation (en fait à l'origine c'est le noyau du système d'exploitation GNU) open-source, généralement gratuit reprenant le principe d'Unix.

Organisation générale

Un noyau central qui gère : système de fichiers, processus, mémoire, communications, etc. Autour un ensemble de commande et logiciels.



Système de fenêtrage

Pas de fusion entre SE et système d'interface graphique (différent de Windows)

- Système de fenêtrage indépendant
- Possibilité de le modifier (KDE, Xfce, Gnome, Unity, ...)
- Possibilité de lancer des fenêtres à distance via la commande `ssh -X`
- Possibilité d'utiliser l'ordinateur même sans interface graphique

Compte utilisateur et protection

- connexion à un système multi-utilisateur : nécessite un compte
- le système connaît la liste des comptes/utilisateurs autorisés
- à chaque compte est associé : un nom (`login`) un mot de passe (`password`) un espace privé (`home directory, ~`)

Rappel sur la protection des comptes/mots de passe

- **Vous ne devez pas** prêter votre compte
- **Vous ne devez pas** prêter votre mot de passe (même pour échanger copier des fichiers)
- **Vous êtes responsable** de ce qui se trouve dans votre compte
- **Vous êtes responsable** de ce que vous faites sur le réseau de l'école (logs analysés)
- **Votre mot de passe** doit être complexe (majuscule, minuscule, alpha numérique, etc)
- **Votre mot de passe** doit être résistant à des attaques type brute force ou dictionnaire → pas de noms, pas de mots usuels, . . .
- ne pas utiliser le même mot de passe pour des usages/sites différents
- ne pas utiliser de base commune pour plusieurs mots de passe

☞ si multitude de mots de passe → logiciel de type `keypass` !

Première manipulation

Première connexion

- Démarrer la machine et choisir Alma, le noyau est chargé en mémoire, c'est la phase de boot
- Un fois terminé la phase de boot : le système d'exploitation démarre et vous obtenez une invite de commande : connectez vous (login/password)
- gestion du compte (par exemple changement de mot de passe) `http://ccc.ipb.fr`
- si ok : le window manager se lance (Gnome), bienvenue sous Linux, vous êtes dans votre *home directory*
- chercher dans *Applications/Utilitaires* et lancer l'application *Terminal* (interpréteur de commande/console) (Alt-F2 + `gnome-terminal`)
- vérifier que vous êtes bien dans votre *home directory* en tapant la commande `pwd` (= *print name of current/working directory*) (vous devez voir quelque chose de la forme `/truc/machin/votrelogin`)
- lancer dans la console le navigateur internet `firefox`
- vérifier que votre connexion à internet fonctionne : `https://perdu.com/`
- Bougez des fenêtres et dockez les. Jouez avec des bureaux (= espaces de travail).
- Vérifiez que le mail fonctionne. Créer des filtres mail.

Le système de fenêtrage

Quelques commandes **clavier** de base pour bien utiliser son environnement (à remplir)

Raccourci	*Fonctionnalité*
	Lancer une application
	Maximiser/minimiser la fenêtre active
	Changer de fenêtre active
	Fermer la fenêtre active
	Se déplacer sur le bureau en dessous
	Déplacer la fenêtre active sur le bureau en dessous
	Déplacer la fenêtre active sur le bureau

Le système de fenêtrage

Quelques commandes **clavier** de base pour bien utiliser son environnement (à remplir)

Raccourci	*Fonctionnalité*
	Lancer une application
	Maximiser/minimiser la fenêtre active
	Changer de fenêtre active
	Fermer la fenêtre active
	Se déplacer sur le bureau en dessous
	Déplacer la fenêtre active sur le bureau en dessous
	Déplacer la fenêtre active sur le bureau

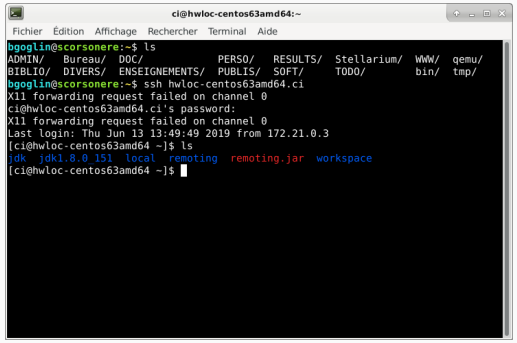
https://doc.ubuntu-fr.org/raccourcis_clavier

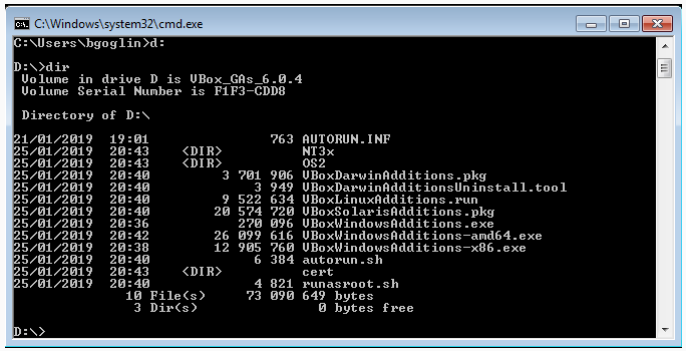
Unix/Bash

Shell - Interpréteur de commandes

Le shell

Le shell est un programme permettant d'interagir avec les services fournis par le système d'exploitation.





- On va utiliser des programmes dédiés à certaines opérations de base *Déplacer des fichiers, renommer un fichier, changer de dossier, changer des droits, etc.*
- Pas très pratique au premier abord
- Extrêmement puissant avec un peu de pratique
- Quel outil utilise t'on autrement ?

Quelques définitions

- shell : c'est un interpréteur de commandes pour interagir avec le noyau. (shell : coquille qui entoure le noyau)
- interpréteur de commande : programme (processus) qui fournit une interface texte entre l'utilisateur et le noyau
  rôle : lire la ligne de commande et exécuter les actions associées
- bash : Bourne-Again shell
  C'est le programme shell par défaut dans la plupart des environnements.

Remarque

Dans la suite du cours, nous mélangerons souvent les termes `shell` et `bash` pour désigner la même chose. Mais il existe plusieurs implémentations de shell différentes comme : `sh`, `csh`, `zsh`, `ksh`, `cmd.exe` ...

Le shell - Fonctionnement

Fonctionnement général - le shell exécute la boucle ∞ suivante :

```
POUR toujours FAIRE :
  | Affichage du prompt (prêt à lire) et attente d'une commande
  | Vérification de la syntaxe
  | SI la syntaxe est correcte ALORS :
  |    exécuter la commande
  |    attente de la fin de la commande en cours
  | SINON
  |    afficher un message d'erreur
```

Votre prompt est la chaîne de caractères, appelée également **invite de commande** qui s'affiche dans votre terminal avant que vous ne tapiez une commande.

Traditionnellement, il se termine par un `$` pour un utilisateur normal, et par un `#` quand on est connecté en tant que superutilisateur.

(Rem : il est possible de modifier le prompt).

Le superutilisateur (appelé aussi "`root`") dans GNU/Linux est l'utilisateur qui a les droits d'accès administratifs au système ($\rightarrow$ commande `sudo`)

Qu'est ce qu'une commande shell ?

- C'est une suite de mot-clés séparés par le caractère de séparation : **l'espace**
  Eviter les espaces dans les noms de fichiers
- Toute commande Unix a la même syntaxe : (séparateur : l'espace)
  `cmd options arguments`
  `ls -l Documents`

- La touche `enter` lance l'interprétation de la commande
- Le résultat de la commande s'affiche dans le terminal
- `options` est une suite de mot clés de la forme :
  `-o` ou `-o valeur` pour les options *courtes*
  `-option` ou `-option=valeur` pour les options *longues*
  → Il est parfois possible de "contracter" les options courtes : `-o -a` devient `-oa`
- Remarques :
  - Lors de la description d'une commande, tout paramètre optionnel sera affiché entre crochets `[-v]`, et tout paramètre obligatoire est affiché sans crochets.
  - Souvent, seul le nom de commande est obligatoire.
  - Souvent, il existe une option `-h` ou `-help` pour obtenir l'aide

Quelques commandes de base

Entrer dans le sous-dossier `divers` du sous-dossier `autres` :

```
$ cd autres/divers
```

Savoir dans quel dossier on se trouve :

```
$ pwd
/home/login/autres
```

Lister le contenu du dossier courant :

```
$ ls
dossier1 dossier2 fichier.txt
```

Lister le dossier courant avec les détails :

```
$ ls -la
```

Lister le dossier courant par ordre chronologique de modification :

```
$ ls -c
```

Exercice : premier contact

- ouvrez un terminal
- tapez `echo hello world !`, validez avec `enter`
- que s'est-il passé ?
- exécutez maintenant `ls`
- que s'est-il affiché ?
- exécutez `cd <nom>` où vous remplacez <nom> par un des répertoires disponibles
- `ls`
- à quoi sert `cd` ?
- `cd ..`
- `ls`
- que s'est-il passé ?
- `ls --reverse`
- quelle est la différence ?
- `ls -l`
- quelle est la différence ?
- `ls -l <nom>`

Pour supprimer les "beeps intempestifs" du terminal : `setx -b`

Obtenir de l'aide

La commande `man` :

- affiche et formate l'aide associée à la commande, ou fonction, passée en paramètre
- `man <cmd>` : affiche le manuel de `<cmd>`
- `man -h` : affiche une courte aide sur `man`
- `man <n> <com>` : affiche une page spécifique à une section (homonyme)
  → 1 : Programmes exécutables ou commandes de l'interpréteur de commandes
  → 2 : Appels systèmes
  → 3 : Appels de bibliothèque (dans les programmes)
  → ...
  Exemple : `man sleep` et `man 3 sleep`
  Par défaut, recherche dans toutes les sections (par ordre "prédéfini", affiche le premier élément qui matche)

- `q` pour quitter
  `espace` pour faire défiler d'un écran
  `up/down` pour faire défiler par ligne
  `/<pattern>` recherche de *pattern*
  `n/N` pour le résultat suivant/précédent

Pour aller plus loin :
`http://www.explainshell.com/` `http://www.grymoire.com/Unix/index.html`
Pour s'entraîner : `https://github.com/phyver/GameShell`

Exercice : manuel

Familiarisons-nous avec `man` :

- `man man`
- Quelles sont les sections des manuels ?
- `man ls`
- quittez le manuel
- `man ls`
- quel est le nom complet de l'option `-a` ?
- quelle est l'option à une lettre équivalente à `--reverse` ?
- quel est le nom complet de l'option `-l` ?
- que fait la commande `ls -cr`
- Lancez la commande `oneko` avec un chat rouge
- Utiliser `pdftk` pour créer un nouveau fichier pdf de ce cours :
  - qui retire la page 13 du contenu
  - pivoté à 180 degrés

Pour les curieux :
ouvrez le man de `gcc`. Avec la touche espace, essayez d'atteindre le bas du texte.
Bonne chance !

Gestion des utilisateurs

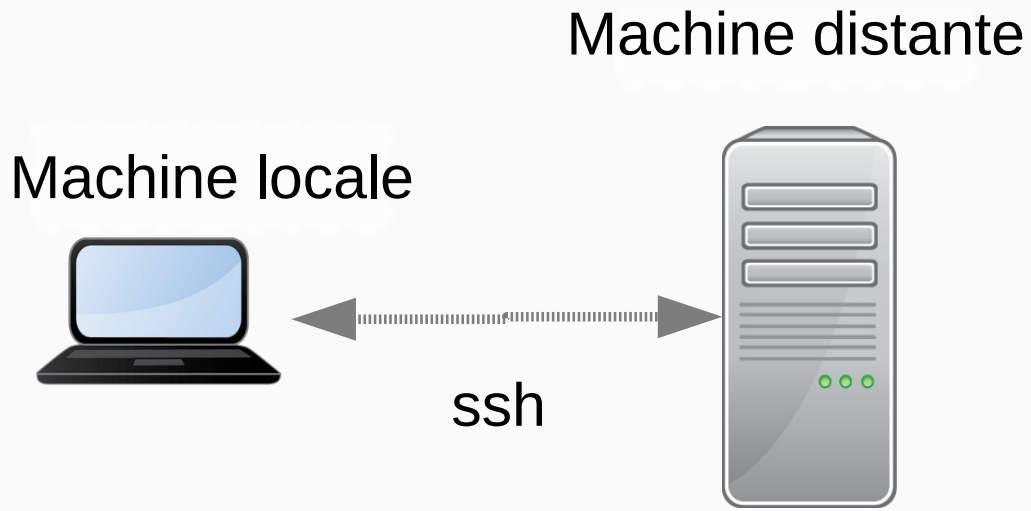
Sur Unix, chaque utilisateur est membre d'un ou plusieurs groupes (qui peuvent être composés d'une seul utilisateur)

- Pour créer/supprimer/modifier un utilisateur : `useradd`/`userdel`/`usermod` (en tant que superuser) (ou `adduser` pour une version interactive)
- Pour changer de mot de passe : `passwd`
- Pour créer/supprimer/modifier un groupe : `groupadd`/`groupdel`/`groupmod` (an tant que superuser) (ou `addgroup` pour une version interactive)
- Pour connaître les groupes auquel un utilisateur appartient : `groups`

⚠ Ces commandes ne sont pas (toutes) utilisables sur votre compte Enseirb

- Le changement de mot de passe se fait via `http://ccc.ipb.fr`
- Vous n'êtes pas superuser

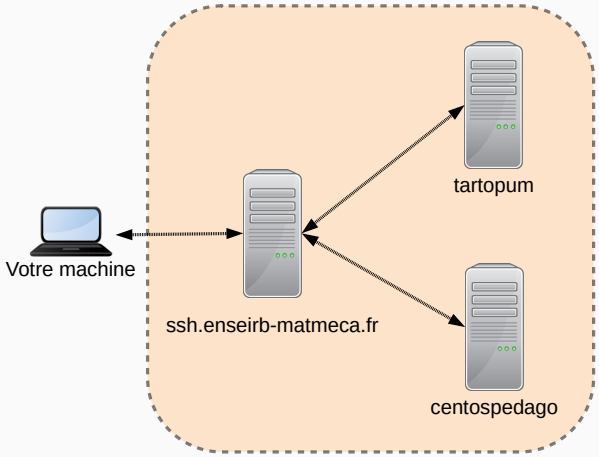
Travail à distance / SSH



Connexion sécurisée à une machine distante

- `ssh [-X] login@hostname` (secure shell)
- `-X` permet d'accéder aux applications graphiques ("Enables X11 forwarding")
- `login` est votre login sur la machine **distante**
- `hostname` est le nom de la machine sur laquelle se connecter

Travail à distance / SSH à l'Enseirb-Matmeca



Connexion sécurisée à une machine de l'Enseirb-Matmeca

- Nécessite de passer par une *frontale/passerelle* : `ssh.enseirb-matmeca.fr`
- ⚠ Ne rien faire tourner sur la frontale
- Se connecter sur une machine de l'école :
  `almanew`, `...`, ou le nom de machine suggéré lors de la connexion
- Les noms sont affichés sur les machines. Il est a priori possible de se connecter sur n'importe laquelle, ... à condition qu'elle soit allumée

Essayer de vous connecter (via la passerelle) à la machine de votre voisin, puis taper la commande `who` pour vérifier que vous êtes tous les deux sur la même machine.

A partir de maintenant, vous pouvez travailler intégralement à distance, ou localement.

Installation de programmes

Pour installer un programme sous Linux, il est possible :

- de télécharger le code source et de le compiler
- de télécharger le paquet (dépend de la distribution, et de l'architecture de la machine) sous forme d'archive contenant tous les fichiers nécessaires et compilés.
  - La plupart des distributions Linux mettent à disposition un dépôt contenant les programmes standards
  - L'installation d'un programme provoque souvent un problème de *dépendance* avec d'autres

  → utilisation d'un gestionnaire de paquets (`apt, dnf, yum, gdebi,`...).
- Principales commandes pour installer un programme avec `apt` (à utiliser en tant que `sudo user` uniquement) :
  - `apt update` → récupère les dernières informations sur le dépôt
  - `apt search expr` → recherche une expression dans la liste de paquets
  - `apt install prog_name` → installe un programme et ses dépendances
  - `apt remove prog_name` → supprime un programme

Historique et raccourcis utiles

Le shell intègre des fonctions de gestion de l'historique des commandes exécutées

- `C-r` (reverse search) : recherche avec motif
  - `C-r` plusieurs fois pour aller plus loin dans l'historique
  - `enter` pour lancer la commande
  - `Flèche droite` pour la modifier
- `C-p` (previous) : commande précédente (up)
- `C-n` (next) : commande suivante (down)
- `history` : affiche l'historique des commandes
- `!num` : relance la commande de numéro donné dans l'historique

Quelques raccourcis utiles pour le shell

- `C-k` (kill) : coupe et mémorise le texte depuis la position du curseur jusqu'à la fin de la ligne
- `C-y` (yank) : insère le texte précédemment supprimé
- `C-a` : début de ligne
- `C-e` : fin de ligne
- `Tab` : **active la complétion (à utiliser sans modération ! !)**

Exercice : quelques commandes en vrac

- Essayer les commandes suivantes et interpréter les résultats :
  `dite`
  `date`
  `whoami`
  `pws`
  `pwd`
  `cal`
  `cal 2012`
  `cal 10 2012`
  `who`
  `uname -a`
  `uptime`
  `top` (comment quitter cette commande ?)
  `time sleep 5`
  `history`
- Grâce à la complétion rechercher une commande qui nettoie l'écran (`cl...`). Vérifier à l'aide du manuel
- Grâce à la complétion rechercher une commande qui permet de quitter le shell (`ex...`). Vérifier à l'aide du manuel

Enchaînement et combinaison de commandes shell

- L'enchaînement de plusieurs commandes se fait avec `;`
  `cmd1; cmd2; cmd3`
  → `cmd2` se lance quand `cmd1` est terminée, etc.
- On peut combiner les commandes en faisant des tests sur les codes de retour :
  `cmd1 || cmd2`
  `cmd1 && cmd2`
- **A ne pas confondre avec :**
  `cmd1 & cmd2` (voir après)
  `cmd1 | cmd2` (voir après)

Exercice : Enchaînement et combinaison de commandes shell

Observez et commentez :

- `cd .. ; ls ; cd; ls`
- Expliquez les comportement des 5 commandes suivantes :
  - `ls . || ls ..`
  - `ls /toto || ls ..`
  - `ls . && ls ..`
  - `ls /toto && ls ..`
  - `ls /toto; ls ..`

  → proposer une application possible du &&
- lancez la commande `oneko`
- est-ce que vous pouvez exécuter une nouvelle commande dans le shell ?
- fermez `oneko` (`C-c`)
- lancez la commande `oneko &`
  Quelle différence ?

Gestion des processus

processus : toute occurrence en mémoire d'un programme en cours d'exécution. Il est identifié par un numéro appelé le **PID**

- Lancer une commande en tâche de fond : `cmd &` (& = "esperluette")
- `C-z` **: endort le processus en cours**
- Réveiller un processus endormi en tant que tâche de fond : `bg [n]` (background)
- Reprendre la main sur processus endormi ou en tâche de fond : `fg [n]` (foreground)
- ☞ ⚠ le numero de job donné à fg/bg pour manipuler un processus spécifique n'est pas le *pid*. Il est obtenu avec `jobs`
- Obtenir le *pid* d'un processus :
  `ps`, ou `echo $!` pour la dernière commande lancée.
- `C-c` **: termine le processus en cours**
- Terminer un processus spécifique
  `kill -KILL pid`
  A quoi sert la commande `kill` ?

Exercice : gestion des tâches 1

- Lancer `oneko` (on pourra utiliser `gnome-calculator` au besoin, ou `ssh -X`)
- Terminer le programme par un raccourci clavier et reprendre la main
- Relancer la commande ; endormir le processus avec un raccourci clavier pour récupérer la main
- Vérifier qu'il est bien endormi (i.e. bloqué ! !)
- Revenez sur le terminal, et repasser `oneko` au premier plan
- Endormez de nouveau `oneko`, puis faites le passer en tâche de fond
- Essayez à nouveau d'interagir avec `oneko`
- Revenez sur le terminal et essayez de tuer le processus de `oneko` depuis la console.

Exercice : gestion des tâches 2

- Lancez la commande `ls -l&`
- Que se passe-t-il ? Pouvez vous l'expliquer ?
- Tapez la commande `xclock & oneko &`
  Trouver une solution dans tous les tests pour que `xclock` "bouge" un peu plus.
  L'utiliser dans la suite.
  (on pourra utiliser `gnome-calculator` et `gnome-font-viewer` au besoin, ou `ssh -X`)
- Reprenez la main sur `xclock` en premier plan
- Terminez `xclock` avec un raccourci clavier
- Terminez `oneko` avec la méthode de votre choix
- Lancez `xclock`, puis endormez le
- Lancez `oneko`, puis endormez le
- Faites exécuter `xclock` en arrière plan, sans modifier l'état de `oneko`.

Unix/Bash

Arborescence et système de fichiers

Système de fichiers

Les fichiers sont structurés sous la forme d'un arbre dont la racine est `/` :

- les noeuds sont des répertoires
- les feuilles sont des fichiers *(au sens large)*

Petit lexique :

- fichier : file
- répertoire : directory
- chemin : path
- répertoire de connexion : home directory (`~`, *tilde*)
- répertoire courant : working directory (`pwd`)

Particularité : sous Unix tout peut être vu comme un fichier

- fichier régulier : fichier contenant du texte, du code source, des commandes exécutables, ...
- répertoire : un fichier contenant d'autres fichiers
- fichiers spéciaux : liens vers des périphériques, canaux de communication entre processus

Dans la suite, si on ne précise pas fichier *régulier*, c'est donc qu'il s'agit de fichier au sens générique désignant n'importe quel noeud de l'arbre (noeud ou feuille).

Notion de chemin

Un chemin identifie un fichier dans le système.

- Il correspond à un déplacement dans l'arborescence des fichiers.
- Il s'écrit comme une suite de noms entrecoupés de `/`
  `/usr/share/fonts`
- On distingue deux parties dans un chemin :
  - le `dirname`, ou préfixe, qui précède le dernier `/` du chemin
    Le préfixe est **toujours** le chemin d'un répertoire.
    Dans l'exemple ici : → `/usr/share`
  - le `basename` qui suit le dernier `/` du chemin. Dans l'exemple ici : → `fonts`

  Remarque : `basename` et `dirname` sont aussi des commandes shell.
- Un chemin est valide si son préfixe est le chemin d'un répertoire existant et si le basename est le nom d'un fichier existant

Deux noms sont réservés dans l'arborescence de fichier et sont contenus dans tout répertoire R sous Unix :

- un élément `'.'` : référence le répertoire R
- un élément `'..'` : référence le répertoire père de R

D'où le `cd ..` dans l'exercice slide 15.

Notion de chemin

Un chemin comme son nom l'indique correspond à un déplacement d'un point A à un point B. Il est donc nécessaire de déterminer le point de départ.

Chemins absolus

Le point de départ **ne dépend pas de la position courante** (working directory) dans le système de fichier. Il est défini quelque soit l'endroit où on se trouve dans l'arborescence. Il commence donc par :

- `/` : La racine du système de fichier
- `~` : Alias/Raccourci vers le répertoire home de l'utilisateur courant
- `~toto` : Alias/Raccourci vers le répertoire home de l'utilisateur `toto`

Chemins relatifs

Le point de départ **dépend de la position courante** (working directory) dans le système de fichier. Il n'est défini et valide que pour la position courante dans l'arborescence. Il commence par :

- `./` : Le répertoire courant
- `../` : Le répertoire parent
- `subdir1` : un fichier contenu dans le répertoire courant (équivalent à `./subdir`)

Notion de chemin

Soit l'arborescence suivante :

```
/usr/
|--local/
|   |--bin/
|   |   |--emacs
|   |--lib/
|   |   |--libm.so
```

Si le répertoire courant est `/usr/local/` :

- On peut faire référence au sous-répertoire `bin` avec :
  `../local/bin`, `bin`, `./bin`, ...
  `/usr/local/bin`, `/usr/local/../local/bin`, ...
- On peut faire référence au fichier `emacs` avec :
  `bin/emacs`, `../../usr/../usr/local/bin/emacs`, ...
  `/usr/local/bin/emacs`, `/usr/local/./bin/emacs`, ...

(Est-ce que ces références sont toutes correctes ?)

Remarque

Il est possible d'obtenir un affichage d'un répertoire sous forme d'arbre avec la commande `tree`.

Arborescence standard

Dans la majorité des cas (variantes possibles selon les versions)

- les répertoires de fichiers exécutables : `/bin`, `/usr/bin`, `/usr/local/bin`
  → on pourra se créer un répertoire `~/bin`.
- répertoires de bibliothèques : `/lib`, `/usr/lib`, `/usr/local/lib`
- pages de manuel : `/usr/man`
- répertoire relatif à l'administration `/etc`
  exemple : `/etc/passwd` (contenant des infos sur les utilisateurs)
- divers : `/tmp`, `/dev` (fichiers spéciaux : drivers)

Chaque processus lancé possède un répertoire courant : sa position dans l'arborescence

- Ex. lors la connexion, le répertoire courant du shell est le répertoire de l'utilisateur (home directory ou `~`)
- Pour modifier cette position courante on utilise la commande : `cd` (change directory)

Quelques règles pour le nommage des fichiers

- Eviter les espaces : utiliser "`_`" (underscore) ou "`-`" (minus)
  espace est le caractère de séparation des mot-clés des commandes bash
- pas de caractères spéciaux : risque d'être interprété par le shell (`$, ~, ...`)
- ajouter un suffixe pour désigner les types de fichiers : .txt, .html, .sh, .tgz, ...
  → c'est une convention
- ⚠ **L'extension ne définit pas le type de fichier ! ! !**
  le nom d'un pdf peut très bien se terminer par «`.txt`», mais ça n'est pas une raison pour faire n'importe quoi.
- Utiliser la commande `file` pour connaître le type d'un fichier

Commandes de manipulation de fichiers

- `cd` (change the working directory)
- `pwd` (print name of current/working directory)
- `ls, -l, -la, -G` (list directory contents)
- `tree` (display the content of a directory in a tree-like form)
- `cp, -r` (copy files and directories)
- `mkdir` (make directories)
- `mv` (move (rename) files)
- `rm, -r` (remove files or directories)
  ⚠ avoid the commands `rm *` ou `rm *.*`, even worse with the option `-f`
- `touch` (change file timestamps) : if the file does not exist, create an empty file
- `file` (determine file type)
- `cat` (concatenate files and print on the standard output)
- `less` display the content of a file with the possibility to navigate inside
- `more` idem (but older)
- `wc, -l, -w`, (print newline, word, and byte counts for each file)
- `chmod` (change file mode bits) change the users permissions. Important command, that will be seen further.

Exercice : échauffement des doigts

Essayer la séquence de commandes suivantes et donner/interpréter les résultats :

- `cd`
- `pwd`
- `ls -la`
- `cd .`
- `pwd`
- `cd ..`
- `pwd`
- `ls -la`
- `cd ..`
- `pwd`

- `cd /etc`
- `tree`
- `ls -la`
- `cat passwd`
- `cd -`
- `cd ..`
- `pwd`
- `cd ..`
- `pwd`
- `cd`
- `pwd`

Liens (symboliques)

`ln [-s] TARGET LINK_NAME` création d'un lien (physique sans `-s`, sinon symbolique) vers `TARGET` avec le nom `LINK_NAME`.

- lien physique (sans -s) : les deux fichiers pointent directement vers les données sur le disque
  → si on efface un des deux, l'autre existe toujours
- un lien symbolique est un fichier qui pointe vers un autre fichier ; si on supprime le fichier cible, le lien symbolique sera toujours présent, mais pointera vers un fichier inexistant

Illustration :

```
cp /etc/passwd .
ln passwd link
cat link
rm passwd
cat link
```

Effacer `link` et recommencer avec `ln -s`

Exercice : manipulation des fichiers

Après chacune des commandes suivantes, vérifier que la commande a été validée grâce à la commande `ls`

- Déplacez-vous dans votre home directory
- Créer un répertoire nommé `IntroUnix` (si ça n'est pas déjà fait)
- Dans le répertoire `IntroUnix`, créer un sous-répertoire nommé `subdir`
- Copier le fichier `/etc/passwd` dans `subdir`
- Copier le fichier `/etc/passwd` dans `subdir` en le nommant `mypasswd`
- Allez dans le répertoire `subdir`
- Vérifier la présence des fichiers `passwd` et `mypasswd`
- Renommer le fichier `passwd` en `passwd.bak`
- Afficher le contenu de ce `passwd.bak`
- Remonter au répertoire père de `subdir`
- Dans ce répertoire, créer un lien symbolique `mots_de_passe` vers le fichier `mypasswd` évoqué ci-dessus
- Copier `subdir` en `subdir.bak`
- Supprimer `subdir` (qu'en est-il de notre lien symbolique ?)
- Proposer une manière plus "directe" de réaliser les deux étapes précédentes
- Créer le fichier `a_test` à l'intérieur de `subdir.bak`
- Gardez votre dossier pour l'instant, nous nous en servirons encore une fois.

La commande `find`

Recherche **récursive** (i.e. dans le répertoire courant et les sous-répertoires) de fichiers → très utilisé (et très puissant) !

```
find path options
```

Quelques options (à connaître) :

- `-name, -iname` : recherche par nom de fichier
- `-type` : recherche par type de fichier
- `-size, -empty` : recherche par taille de fichier
- `-ctime, -mtime, -atime` : recherche par date de création, date de dernière modification ou date de dernier accès
- `-user, -group` : recherche par propriétaire ou par groupe (cf après)
- `-perm` : recherche par autorisations d'accès (cf après)
- `-mindepth / -maxdepth` : profondeur de récursivité min / max
- `-exec` exécute une commande sur tous les fichiers listés :
  `find path options -exec commande {} \;` où `{}` est remplacé par le nom de chaque fichier
  ⚠ C'est une commande dangereuse !... utiliser `-ok` (à la place de `-exec`) pour demander confirmation

La commande `find` : exercices

- Lister récursivement les fichiers nommés `.bashrc` dans votre home directory.
- Lister récursivement les fichiers nommés `bin` (avec ou sans majuscule) dans votre home directory.
- Lister récursivement les répertoires nommés `bin` dans votre home directory.
- Lister récursivement tous les fichiers du répertoire courant qui font moins de 10 Ko
- Lister récursivement tous les fichiers du répertoire courant qui sont vides (en créer un ou deux avant pour vérifier)
- Lister récursivement tous les fichiers du répertoire courant qui ont été modifiés ces 10 dernières minutes
- Lister récursivement tous les fichiers du répertoire courant qui ont été modifiés entre ces 2 et 10 dernières minutes.
- Lister récursivement tous les fichiers du répertoire `~/IntroUnix` qui ont été modifiés la dernière minute et les renommer (en demandant confirmation) en leur ajoutant un .old à la fin. Tester en créant quelques fichiers vides.

Droits des fichiers

Toute opération sur un fichier est soumise à un ensemble de droits d'accès qui sont de 3 types.

r : droit de lecture

- Si répertoire, consultation de ses entrées (c.-à-.d, `ls` autorisé, et complétion)
- Sinon, consultation du contenu du fichier

w : droit d'écriture

- Si répertoire, droit de création, de renommage et de suppression d'une entrée dans le répertoire
- Sinon, droit de modification du contenu du fichier

x : droit d'exécution (/ traverser)

- si répertoire, droit de traverser (c.-à-.d., `cd` autorisé)
- sinon, droit d'exécution

Droits des fichiers

- Chaque fichier a un propriétaire et un groupe
- 3 catégories d'utilisateurs :
  - Propriétaire (`u`)
  - Groupe propriétaire (`g`)
  - Tous les autres (`o`)
- Chaque catégorie possède ses types d'accès `r w x`

Droits des fichiers - consultation

- `ls -ld` → donne les informations sur les droits des fichiers d'un répertoire sans montrer son contenu
- Format de sortie de `ls -l`
- `- --- --- ---`
- `-` → Type du fichier
  - `-` : fichier régulier
  - `d` : répertoire
  - `l` : lien symbolique
  - `s` : socket
  - ...
- `---` → Droits du propriétaire (`rwx`)
- `---` → Droits du groupe (`rwx`)
- `---` → Droits des autres (`rwx`)

Droits des fichiers - modification

- Modification des droits sur un fichier/répertoire existant
  - `chmod droit fichier`
- Droits à appliquer :
  - Catégorie : u (=user), g (=group), o (=other), ou a (=all)
  - Opérations : + (ajout), - (retrait), = (affectation)
  - Exemple : u+rx,o=r
- Alternativement, on peut aussi les donner en octal : 4 (lecture), 2 (écriture) et 1 (exécution)
  Exemple : `641` pour `rw-r----x`, `752` pour `rwxr-x-w-`
- Il est possible de modifier des droits en profondeur (i.e. récursivement) avec l'option `-R`

Exercices

1. En se connectant à une machine de l'Enseirb-Matmeca(localement ou via ssh)
   - Expliquer `-rw-------`
   - Expliquer `drwx--x--x` (cf `ls -ld ~rollet`)
   - Essayer d'afficher le contenu du fichier `~rollet/IntroSyst/file.txt` en utilisant `TAB` pour compléter son nom. Expliquez.
2.
   - A qui appartient le fichier `/etc/passwd` ? Et à quel groupe ?
   - Pourquoi n'avez vous pas le droit de le modifier ?
   - Quels sont vos droits sur ce fichier
3. Donner deux façons possible d'obtenir les droits suivants (en symbolique et en octal)
   - `-rw-r--r--`
   - `-rwxr-xr-x`
   - `-rwxr-xr--`
   - `-r--r--r--`
   - `-r----x---`

Exercices

4. Soit le répertoire
   `drwxr-xr-x 2 user staff 68B Mar 23 12:01 tmp`
   Donner et expliquer le résultat des commandes suivantes sur `tmp` en supposant que `tmp` ce trouve dans le homedir de `user`, et que le terminal est aussi positionné dans le homedir de `user`.
   - `chmod 355 tmp; ls tmp`
   - `cd tmp; pwd`
   - `cp /etc/passwd .; ls`
   - `cd ..; chmod 755 tmp; ls tmp`
   - `chmod 655 tmp; cd tmp`
   - `chmod 555 tmp; rm -f tmp/passwd`

   Faites les modifications nécessaires jusqu'à suppression de ce répertoire.

Exercices

1. Revenez à votre `subdir.bak` (`cd ~/IntroUnix/subdir.bak`).
2. Enlever le bit de lecture sur le fichier `passwd.bak` pour tous les utilisateurs et vérifier à nouveau les droits
3. Afficher le contenu de ce `passwd.bak`
4. Revenez dans le répertoire `~/IntroUnix` et enlever le bit autorisant le parcours sur le répertoire `subdir.bak`
5. Vérifier
6. Supprimer le répertoire `subdir.bak`
7. Lister récursivement tous les répertoires de votre compte dont vous êtes propriétaire, idem avec les fichiers ayant les droits 777

Copie distante (scp)

Copier un fichier vers une machine distante (secure copy)

- `scp [-r] src1 [src2 ... srcN] login@hostname:dst_path`
- `dst_path` est un chemin absolu, ou un chemin relatif à partir de `~login`
- Suit les mêmes règles que `cp`

Exemple : `scp -r ~/IntroUnix rollet@ssh.enseirb-matmeca.fr:~/`

Copier depuis une machine distante vers la machine locale

- `scp login@hostname:src_path dst_path`
- Suit les mêmes règles que `cp`
- 1 seul fichier (au sens large) peut être la source

Exemple : `scp -r rollet@ssh.enseirb-matmeca.fr:~/IntroUnix ~/RI1`

Unix/Bash

Substitutions et variables

Substitutions et expansions (1/3)

Le shell analyse la ligne de commande avant de l'exécuter pour faire des substitutions/expansions. L'ordre des substitutions est toujours le même, il y a entre autres :

1. **Expansion des accolades** :
   `a{b,c,d}e` = `abe ace ade`
2. **Expansion des tildes** `~`
   `~` → chemin vers le homedir de l'utilisateur courant
   `~toto` → chemin vers le homedir de `toto`
3. **Expansion des variables, variables d'environnement** :
   - syntaxe d'une variable : `$VAR`, `${VAR}`, `${VAR:-defaut}` `(remplacé par defaut si non initialisée)`
     Exemple : `toto=ta ; echo ${toto}ta` → `tata`
   - affectation d'une variable :
     `VAR=value` → variable disponible **uniquement** dans l'environnement du processus courant ; ⚠ PAS d'espace entre le nom de var et le =
     `export VAR=value` → variable disponible aussi dans l'environnement des processus fils (→ variable d'environnement)
   - Commande `env` (sans argument) → lister les vars d'environnement

   Remarque : pour un export "permanent", il faut intégrer la commande dans le fichier `~.bashrc` ou `~.profile`

Substitutions et expansions (2/3)

4. **Substitution de commandes** :
   `$(cmd)` exécute `cmd` dans un sous-shell et remplace `$(cmd)` par le résultat de l'exécution de la commande.
   Exemple : `echo $(ls)` (essayer `echo ls` pour comparer)
   Peut parfois s'utiliser avec backquote : `echo ‘ls‘` (à éviter)

5. Expansion arithmétique (expressions de type `$(( expr ))`)

6. **Coupures des mots** :

   - Chaque séquence de blancs (par défaut : espaces, tabulations, ou retours de ligne) délimite chaque mot-clés (variables, commande, paramètre, option) de la ligne de commande
   - `'une chaîne de caractère'` : → Le contenu n'est pas interprété
   - `"une chaîne de caractère"` : → Le contenu est interprété

   → *le délimiteur peut être modifié grâce à la variable `IFS` (Internal Field Separator)*. Exemple :
   `IFS=$' \t\n'` (qui est la valeur par défaut) ; remarquer la notation `$'...'` pour gérer les `\` (on abordera à nouveau l'IFS plus loin dans ce cours)

Substitutions et expansions (3/3)

7. **Expansion des noms de fichiers** : on parle de *wildcards* ou *jokers*. Résumé :

Fonction	Motif
Représente le caractère 'a'	`a`
Représente n'importe quel caractère	`?`
Représente n'importe quelle chaîne de caractères (y compris la chaîne vide)	`*`
Représente le caractère 'a' ou 'b'	`[ab]`
Un caractère parmi tous les caractères entre 'n' et 'z'	`[n-z]`
Un caractère parmi tous les caractères sauf ceux entre 'n' et 'z'	`[^n-z]`
Représente un caractère décimal	`[0-9]`
Représente tout sauf un caractère décimal	`[^0-9]`
Représente un caractère alphabétique (majuscule ou minuscule)	`[A-Za-z]`
Echappe le caractère 'a' (utilisé pour les caractères spéciaux)	`\a`

⚠ ne pas confondre avec les expressions régulières :

- les wildcards concernent les commandes : `ls`, `mv`, `cp`, `find` (avec des guillemets et sans l'option `-regex`), et généralement les commandes manipulant des fichiers (exemple `evince *.pdf`)
- les expressions régulières concernent les commandes : `sed`, `awk`, `grep`, et seront vues plus tard

→ L'objectif est de réaliser du Pattern Matching

Exemple : `[04a-dG-L6]` matche avec les caractères : $0, 4, a, b, c, d, G, H, I, J, K, L, 6$

Exercices

Anticiper les résultats des commandes suivantes

- Quelle est la différence entre `evince *.pdf` et `evince "*.pdf"`
- Que fait `echo projet/{src/{module1,module2},lib,bin}` ?
- Que produit la séquence suivante ?

```
echo ${BROL:-Je ne le connais pas}
export BROL="Monsieur Brol"
echo ${BROL:-Je ne le connais pas}
```

- Que produit `echo $PATH` ? Que représente cette variable d'environnement ?
- Que font les séquences suivantes :

```
export $VAR=$(date); echo VAR
export VAR=$(date); echo VAR
export VAR=$(date); echo $VAR
```

- Que fait la séquence suivante :

```
export BROL="Monsieur Brol"
echo "$BROL et trol"
echo '$BROL et trol'
```

Exercices

- Anticiper (et confirmer) le résultat des commandes suivantes :
  ```
  echo hostname
  echo 'hostname'
  echo `hostname`
  echo $(hostname)
  echo $hostname
  ```
- Que fait `ls -l /usr/lib/lib[a-jt-z]?e*`
- Que fait la séquence suivante :
  ```
  RAC=/bin
  echo $RAC
  echo $RAC/pwd
  $RAC/pwd
  ```
- Que fait la séquence suivante :
  ```
  echo '/bin/[a-c]?'
  echo "/bin/[a-c]?"
  echo /bin/[a-c]?
  ```
- Écrire une commande permettant de renommer récursivement tous les fichiers `.txt` commençant par un `B` majuscule ou minuscule du répertoire `toto` en leur ajoutant une extension `.save` (Exemple : le fichier `Bim.txt` devient `Bim.txt.save`). Tester en créant des fichiers avec la commande `touch`.
  ☞ Pour comprendre ce qui se passe, rajouter un `echo` devant.
- Compléter la séquence suivante pour calculer et afficher la somme `c` des carrés de `a` et `b` :
  ```
  a=5779 && b=1274 && c= ...
  ```

Unix/Bash

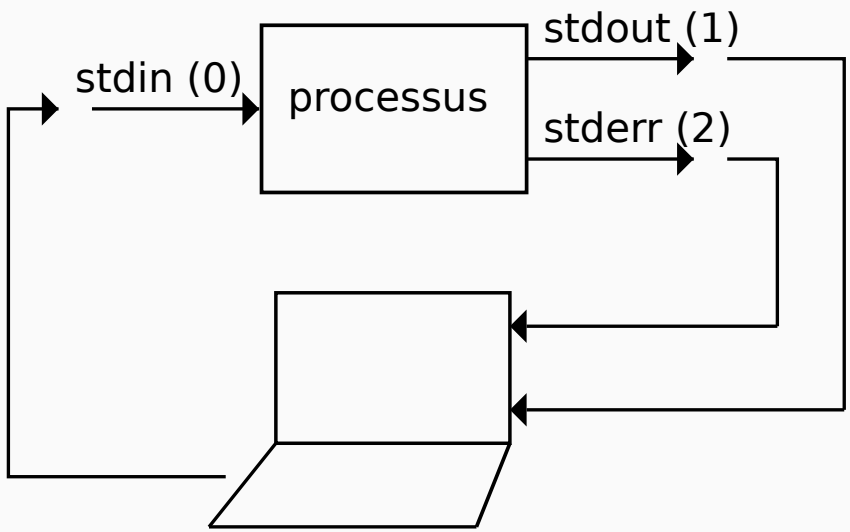
Mécanismes d'entrées/sorties

Mécanismes d'entrées/sorties

Généralement une opération d'E/S :

- transfert de données entre une zone mémoire et un fichier ou un périphérique
- Ex. un processus effectue des E/S : depuis le clavier vers l'écran

A sa création chaque processus possède 3 flots E/S standards



- `stdin (n° 0)` : entrée standard opérations lecture, par défaut le clavier
- `stdout (n° 1)` : sortie standard opérations d'écriture, par défaut l'écran
- `stderr (n° 2)` sortie d'erreur opérations d'affichage des messages d'erreurs, par défaut l'écran

Unix, permet à l'utilisateur de modifier et de reconnecter les flots d'E/S
c'est le mécanisme de redirection

La commande `read` (1/2)

La commande `read` permet de lire sur l'entrée standard et affecte les valeurs saisies dans la ou les variables passées en argument. Exemples :

```
myPrompt$ read toto1
Hello
myPrompt$ echo $toto1
Hello
```

Tous les mots saisis sont stockés dans la variable passée en argument :

```
myPrompt$ read toto1
Hello tout le monde
myPrompt$ echo $toto1
Hello tout le monde
```

La commande `read` (2/2)

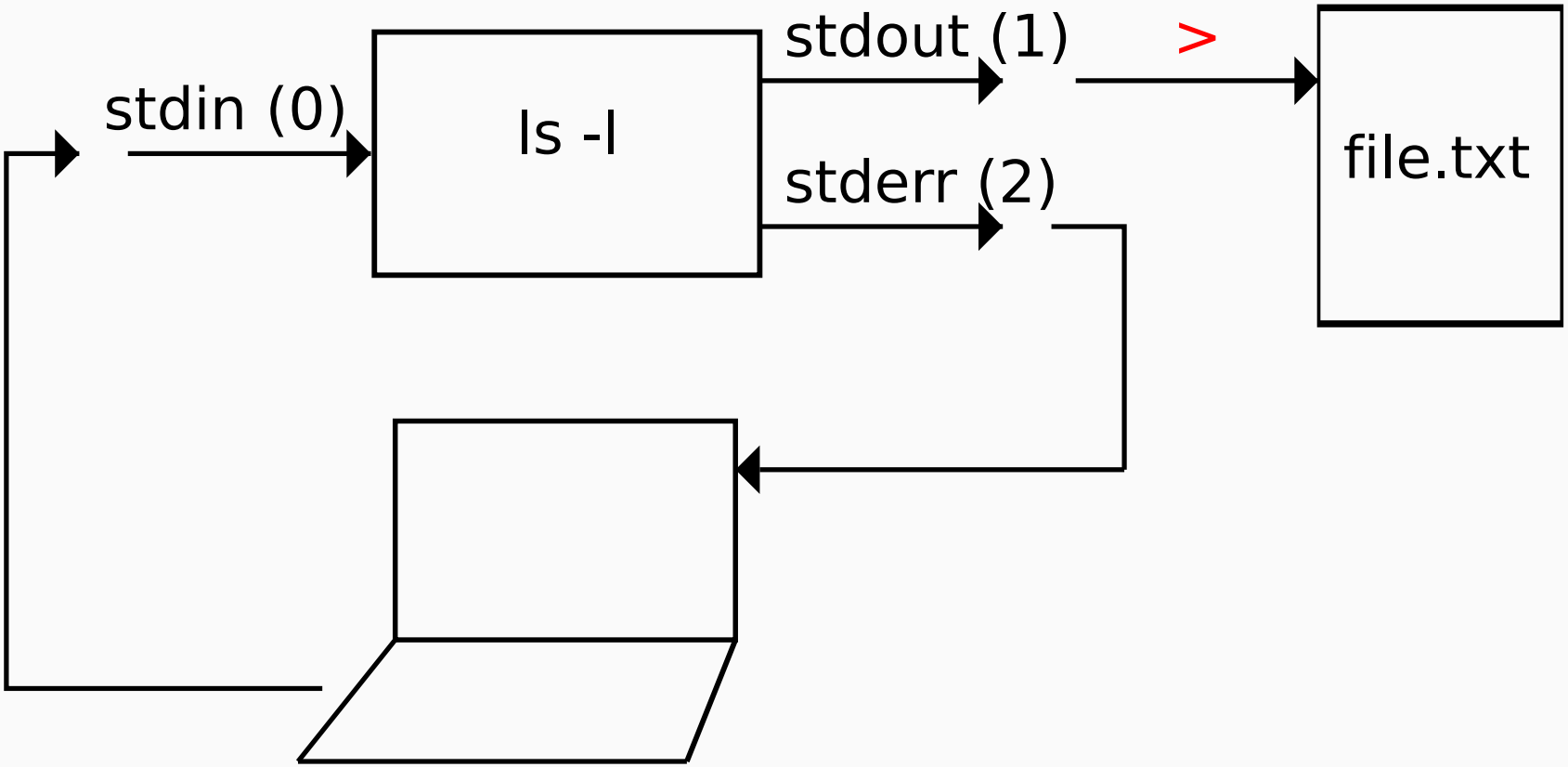
Il est possible de donner plusieurs variables en argument :

```
myPrompt$ read toto1 toto2 toto3
Hello tout le monde
myPrompt$ echo $toto1
Hello
myPrompt$ echo $toto2
tout
myPrompt$ echo $toto3
le monde
```

Exercice : dans la console, demandez à l'utilisateur ses jour, mois et année de naissance, et donner son âge approximatif (i.e. sans tenir compte du mois).

Mécanismes de redirection (1)

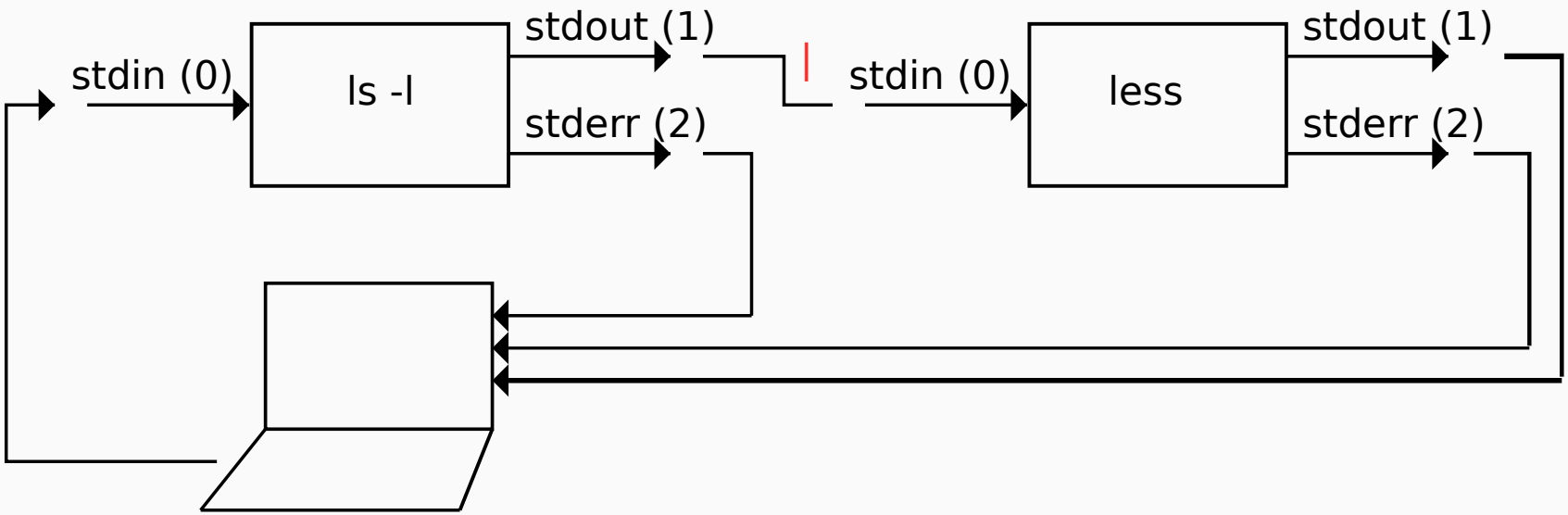
Rediriger un processus (programme) vers un fichier via '>'.
Ex : `ls -l > file.txt`



Si le fichier existe il est écrasé

Mécanismes de redirection (2)

Rediriger un processus (programme) vers un autre processus (programme) via le `pipe` (tube) : `'|'`
Ex. `ls -l | less`



Possibilité de créer des pipelines complexes en une seule ligne de commande

Mécanismes de redirection (3)

D'autres mots clés existent : les plus communs

- `< file` : redirection de l'entrée standard dans un fichier
- `> file` : redirection de la sortie standard dans un fichier (déjà vu)
- `>>` : idem, avec positionnement à la fin du fichier s'il existe
- `cmd | cmd` : le pipe (déjà vu)
- `2> file` : redirige la sortie d'erreur vers `file`
- `2>&1` : fusionne la sortie d'erreur sur le sortie standard

Question : que fait la commande

```
find / -iname "bin" 2>/dev/null
```

Editer des fichiers sans interface graphique

Très fréquent, notamment en cas de connexion sur une machine distante, ou simplement en l'absence d'interface graphique.
Plusieurs solutions, notamment :

- Utiliser des redirections. Par exemple :
  ```
  echo "yes" > yes_or_no.cfg
  echo "une ligne de plus" >> mon_fichier.txt
  ```
- Utiliser des éditeurs de fichier sans interface graphique (`nano`, `pico`, `vi` ...)
- `emacs` en mode "no window" `emacs -nw` (cf après)

Exercice : essayer d'éditer un petit fichier `.txt` sans interface graphique.

Exercices

1. Que fait la ligne de commande
   `date > tmp.txt; find ~ -name ".o*" | wc -l >> tmp.txt`
   Que contient finalement `tmp.txt`
2. Que fait la ligne de commande
   `echo $(date) : $(who | wc -l) util. sur $(hostname)`
3. Rediriger la sortie de `ls -la` vers un fichier (et afficher immédiatement le contenu de ce fichier)
4. Placez vous dans votre `homedir`
   - Déplacez vous dans le répertoire `dir` (qui n'existe pas), un message d'erreur est-il apparu ? si oui où ?
   - Recommencer mais en redirigeant la sortie d'erreur vers le fichier `err`.
   - Un message est-il apparu ? le fichier `err` a-t-il été créé ? Que contient le fichier `err` ?
   - Recommencer mais cette fois en positionnant le curseur à la fin (`>>`). Que contient le fichier `err` ?

Unix/Bash

Commandes avancées

Fichier de configuration `.bashrc`

- fichier `.bashrc` : fichier de configuration chargé à chaque fois qu'un **bash** est lancé
- configuration, personnalisation de l'interpréteur de commandes
- par ex. : créer des *alias*, modifier l'affichage du terminal, définir des fonctions `shell`, exporter des variables d'environnement, ...

Fichier de configuration `.bashrc`

Quelques exemples (de base) de lignes qu'il est possible de trouver dans le fichier `.bashrc`. Voir la documentation pour toutes les autres possibilités :

```
#alias
alias ll='ls -la'
alias rm='rm -i'
alias mv='mv -i'
alias cl='clear'
```

1. Vérifier la présence (et au besoin le créer) du fichier `~/.bashrc`, le sauvegarder (par exemple `~/.bashrc.save`)
2. Tester les différentes options ci-dessus
3. Pour recharger le fichier de configuration, lancer un nouveau `shell`
4. Ajouter dans le `.bashrc` deux alias pour se connecter "plus facilement" en ssh à l'Enseirb-Matmeca, avec et sans option "-X".

Fichier de configuration `.bashrc` et `PATH`

- Rappeler à quoi sert la variable `PATH`
- Ajouter dans votre fichier `.bashrc` la ligne :
  `export PATH=$PATH:~/myscripts/ #par exemple`
  Expliquer.
- Créer un répertoire `~/bin` dans votre home, et l'ajouter de manière permanente dans votre `PATH`
- ⚠ Que fait la commande `PATH=$PATH:.` ?
  Pourquoi faut-il éviter ça ?

Archivage - Compression

Archivage et compression avec `tar`

- `-v` : mode verbose
  `tar -cvf <archive> <dir>` : Archive dans le fichier `<archive>` le répertoire `dir`
  `tar -tvf <archive>` : liste les fichiers dans l'archive
  `tar -xvf <archive>` : extraction de l'archive
  `tar -cvzf <archive> <dir>` : archive et compresse avec `gzip` (extension `.tgz`)
  `tar -xvzf <archive>` : extrait une archive compressée avec `gzip`
  `tar -cvjf <archive> <dir>` : archive et compresse avec `bzip2` (extension `.tbz`)
  `tar -xvjf <archive>` : extrait une archive compressée avec `bzip2`
- **Rq :** possibilité de compresser un fichier seul avec `gzip/gunzip` ou `bzip2/bunzip2`.
- **Rq :** les versions récentes de `tar` sont capables de reconnaître automatiquement le format de compression d'un fichier

Espace disque

Mesure de l'espace disque avec `du (disk used)` et `df (disk free)`

- `-h` (human readable) : affiche sous la forme (Kio, Mio) valable pour les deux commandes
- `df .` : espace disque de la partition courante + diverses infos

```
Filesystem     Size   Used  Avail Capacity  Mounted on
/dev/disk0s2  465Gi  338Gi  127Gi    73%    /
```

- `du <dirs>` : espace occupé par la liste de répertoires dirs

  `du -c` : affiche le total de chaque répertoire

  `du -s` : affiche seulement le total de tous les répertoires

Exercices

Archivage et compression avec `tar`

1. Depuis une machine de l'Enseirb-Matmeca, ou via `ssh`, copier le répertoire `~rollet/public_html/IntroUnix` dans un dossier `tmp` sur votre compte
2. Déterminer la place occupée par le répertoire `~/tmp/IntroUnix`
3. Créer une archive compressée de ce répertoire nommée `introUnix-arch.tgz`
4. Lister le contenu de l'archive
5. Extraire l'archive dans un dossier `tmp` et vérifier la présence des fichiers
6. Supprimer l'archive ainsi que le dossier `tmp`
7. Extraire désormais l'archive `~rollet/public_html/IntroUnix/testtar.txt` dans un dossier `tmp`, jusqu'à obtention du fichier `README.txt` (au fait, est-ce bien une archive ? Comment savoir ?)

Combinaisons de commandes

Quelques combinaisons de commandes (rappels + éléments nouveaux)

- `cmd1 ; cmd2` : **enchaînement séquentiel** ; `cmd2` est exécutée lorsque `cmd1` est terminée
- `cmd1 & cmd2` : **enchaînement parallèle** ; `cmd1` et `cmd2` sont lancées en parallèle
- `cmd1 && cmd2` : **et** ; `cmd2` est exécutée si `cmd1` retourne *vrai*
- `cmd1 || cmd2` : **ou** ; `cmd2` est exécutée si `cmd1` retourne *faux*

Filtres, traitement de chaînes

- `grep` : recherche de motif dans des fichiers (cf après)
- `sort` : trie des lignes dans l'ordre alphabétique
- `uniq` : trie les doublons
- `head` : extraction des premières lignes
- `tail` : extraction des dernières lignes
- `tr` : remplacements de caractères
- `sed` : remplacements de motifs (cf après)
- `cut` : sélection par colonne
- `awk` : découpe une chaîne de caractères en fonction de délimiteurs.

  Les délimiteurs sont connus de bash grâce à la variable `IFS` (généralement, par défaut : espace, tabulation et saut de ligne)

Expressions régulières (*REGEXP*)

→ L'objectif est de réaliser du Pattern Matching
Expression régulières = motif représentant un ensemble de chaînes de caractères
BRE = Basic Regular Expressions (≠ ERE)
Principales expressions régulières (POSIX, non exhaustif) :

- `a` le caractère 'a'
- `.` n'importe quel caractère
- `^` début de ligne
- `$` fin de ligne
- `\a` échappe le caractère 'a' (caractères spéciaux)
- `?` répète l'expression précédente zéro ou une fois
- `*` répète l'expression précédente zéro fois ou plus
- `+` répète l'expression précédente une fois ou plus
- `{n}` répète l'expression précédente exactement `n` fois
- `{n,m}` répète l'expression précédente entre `n` et `m` fois
- `{n,}` répète l'expression précédente au moins `n` fois
- `{,m}` répète l'expression précédente au plus `m` fois
- `[ab]` Représente le caractère 'a' ou le caractère 'b'
- `[n-z]` un caractère parmi tous les caractères entre `n` et `z`.
- `[^n-z]` Un caractère parmi tous les caractères sauf ceux entre `n` et `z`
- `(expr1expr2)` Groupe un ensemble d'expressions
- `expr1|expr2` Ou logique sur les expressions `expr1` et `expr2`

Exercices sur les expressions régulières

- Que capturent les expressions régulières `abc+`, et `(abc)+` et `[abc]+` et `(chat|chien)s?`
- Ecrire une expression régulière qui matche (uniquement) avec tous les nombres binaires, mais évitant la chaîne vide
- Ecrire une expression régulière qui matche (uniquement) avec tous les nombres binaires qui finissent par `'00'`
- Ecrire une expression régulière qui matche (uniquement) avec tous les nombres binaires contenant au moins trois `'1'`
- Ecrire une expression régulière qui matche (uniquement) avec les adresses email de `gmail.com`
- Ecrire une expression régulière qui matche (uniquement) avec l'ensemble des mots alphabétiques de longueur paire
- Ecrire une expression régulière qui matche (uniquement) avec les numéros de portable en France (commençant par 06 ou 07)
- Ecrire une expression régulière qui reconnaît une adresse IPv4 sous forme décimale pointée (pour simplifier on considérera des nombres entre 0 et 999).
- Ecrire une expression régulière matche (uniquement) avec les URL (`http`, `https`, ou `ftp`) du domaine `.fr`.

(Zoom sur) la commande `grep`

Utilisation : `grep [options] pattern files`
→ recherche du motif `pattern` dans `files`
→ `files` peut être un ou des fichiers, ou plus généralement un flux de texte
⚠ Le pattern peut-être exprimé sous forme d'*expression régulières* à ne pas confondre avec les *wildcards* vus précédemment.

Quelques options utiles :

- `-i` : ignore la casse
- `-n` : précède chaque ligne par son numéro
- `-v` : affiche les lignes ne contenant pas le pattern
- `-R` : recherche récursive dans les répertoires
  Exemple : `grep -R "include" .`

⚠ les méta-caractères `?,+,{,|,(,)` doivent être échappés (avec \). Sinon, utiliser l'option `-E`.
⚠ Ne pas oublier les guillemets autour des pattern (risque d'expansion de certains caractères).

Exercices sur la commande `grep`

- Tester vos réponses précédentes avec la commande `grep -E`.
- Afficher dans le fichier `/proc/cpuinfo` toutes les lignes, avec leur numéro, commençant par `cpu`.
- Dans le répertoire `/etc`, extraire toutes les adresses IP contenues dans des fichiers (recherche récursive)
- Tapez la commande `man ls`. Utilisez la recherche par motif (touche `/`, puis `n` pour *next*) pour rechercher toutes les expressions commençant par “list”.

  ☞ Noter que la commande `man` utilise le programme `less` pour afficher, ce dernier permettant une recherche par motif sous forme de *REGEXP*.

(Zoom sur) la commande `sed`

- D'après le `man` :
  `sed` : `stream editor for filtering and transforming text.`
- La recherche de motif peut se faire sur un caractère, suite de caractères ou des expressions régulières.
- C'est une commande vaste → on se concentre sur un usage très fréquent : *la substitution* :

  ```
  sed 's/text_to_replace/new_text/' [file]
  ```

  `sed 's/guten/ta/'` → remplace la première occurrence de `guten` par `ta` dans le flux

  `sed 's/guten/ta/g'` → remplace toutes les occurrences de `guten` par `ta` dans le flux

  ☞ Les `/` sont des séparateurs et peuvent être remplacés par (presque) n'importe quel caractère. Exemple `sed 's!guten!ta!g'`
- Par défaut, le résultat est affiché sur la sortie standard. On peut utiliser l'option `-i` pour modifier directement le fichier d'origine

Exercices sur la commande `sed`

- Afficher le contenu du fichier `/etc/passwd` en remplaçant la lettre “x” par la chaîne “y”.
- Afficher le contenu du fichier `/etc/passwd` en remplaçant la lettre “x” par la chaîne “icse”.
- Afficher le contenu du fichier `/etc/passwd` en supprimant tous les “o”.
- Afficher le contenu du fichier `/etc/passwd` en remplaçant tous les chiffres par “n”.
- Afficher le contenu du fichier `/etc/passwd` en remplaçant toutes les voyelles minuscules par la chaîne “VOYELLE”.

☞ Essayer de faire la même chose avec le commande `tr`.
Question : différences entre les commandes `sed` et `tr` ?

(Zoom sur) la commande `awk`

- `awk` : découpe (notamment) une chaîne de caractères en fonction de délimiteurs
- commande très puissante, mais aussi très vaste → interprète des programmes en langage de script *Awk*
- Usage : `awk [options] 'selecting_pattern {action}' input` vers la sortie standard
- le pattern de sélection s'écrit sous forme de *REGEXP* au format `/expr/`.
- Options fréquentes :

  `-f file` : lit un fichier contenant un programme en langage *Awk*
  `-F fs` : pour définir le délimiteur de champs

  Quelques exemples :

  - `awk '{print}' input.csv` → affiche toutes les lignes de `input.csv`
  - `awk '/toto/ {print}' input.csv` → affiche toutes les lignes de `input.csv` contenant le pattern `toto`.
  - `awk '{print $1,$4}' input.csv` → affiche les champs 1 et 4 de chaque ligne du fichier `input.csv`
  - `awk -F; '{print "Valeur: "$1}' input.csv` → affiche le champ 1 de chaque ligne du fichier `input.csv`, avec " ;" comme délimiteur, précédé du texte `Valeur:`

Exercices sur la commande `awk`

- A partir du fichier `/etc/passwd`, pour chaque ligne, affichez :
  - Le `Username`
  - Le `Username` suivi par `:GroupID` (le sien)
  - Idem, mais avec un affiche style "Utilisateur `Username` a le groupe ID *GroupID*.
- Récupérer le fichier
  `https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/refuse.mail`
  (qui normalement se trouve à l'adresse `/etc/mail/refuse.mail` mais a été anonymisé ici). En extraire :
  - les noms des machines refusées
  - les adresses IP refusées, avec le texte "IP `@IP` refusée".
- Afficher le contenu du fichier `.bashrc` en retirant (uniquement) les commentaires qui commencent au début d'une ligne.

☞ Essayer de faire la même chose avec le commande `cut`.
Question : différences entre les commandes `awk` et `cut` ?

Exercices

- À partir du fichier `/etc/passwd` et des commandes de traitements de chaînes, trouver la ligne de commandes permettant de
  - Lister tous les utilisateurs
  - Idem, mais n'afficher que les deux premières lignes, idem avec les deux dernières
  - Lister tous les utilisateurs en affichant des `@` à la place des `a` et des `!` à la place des `i`
  - Trier les utilisateurs dans l'ordre alphabétique et afficher les deux premiers
  - Afficher `r00t` à la place de `root`
  - Rediriger le contenu du fichier vers le fichier `~/IntroUnix/tmp/passwd.txt` en remplaçant tous les `:` par `::`
  - Lister tous les `userID` du fichier `passwd.txt`
  - Afficher tous les utilisateurs ayant bash pour shell, l'affichage se fera sous la forme `<user> uses /bin/bash`
- Compter le nombres d'adresses IP et URL (terminant par `enserb.u-bordeaux.fr`) refusés (i.e. terminant par l'expression "refuse") dans le fichier `refuse.mail` précédent.

Exercices

- Ecrire un alias qui affiche la dernière commande lancée et efface la console au bout de 2 secondes. Le modifier pour que le numéro n'apparaisse pas.
- Le fichier `notes.txt` (disponible ici : `https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/notes.txt`) contient des notes au format : `login,note`. Écrire les commandes permettant d'afficher les logins des étudiants qui ont obtenu une note supérieure ou égale à 10, triés par ordre décroissant de note.
- Quelques exercices pour vous entraîner / réviser : `https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/exos_unix_shell_bash-sujet.pdf`