Introduction à l'environnement UNIX

R&I 1

Antoine Rollet

Bordeaux INP - Enseirb-Matmeca

Avant de commencer

Première connexion

- nécessite d'être inscrit
- nécessite d'avoir un certificat de scolarité
- nécessite d'avoir validé son compte sur le site de Bordeaux INP
  https://ccc.bordeaux-inp.fr
  (en utilisant l'identifiant sur le certificat)

Il est OBLIGATOIRE d'avoir validé son compte avant les TDs
A faire en sortant si ce n'est pas déjà fait ! ! !

Déroulement du module / évaluation

- 11 séances de 2h de cours / TP sur machine
- 1 épreuve sur machine, mode examen (sans Internet)

Supports de cours :
https://rollet.zzz.bordeaux-inp.fr/

Objectifs de cet enseignement

- appréhender les outils informatiques fondamentaux et nécessaires au cursus d'informaticien
- notion de programmation, de programme, exécution, systèmes d'exploitation
- acquérir de bons automatismes
- devenir efficace et productif devant une machine
- arrêter d'utiliser la souris → pourquoi ?
- Privilégier les outils libres

Choix pour cet enseignement

- initiation à un système d'exploitation : GNU/Linux
- édition de code : emacs
- programmation shell
- initiation à un langage interprété : bash
- initiation à un langage compilé : LaTeX

Plan

Ressources informatiques à l'Enseirb-Matmeca

Unix/Bash

- Introduction
- Shell - Interpréteur de commandes
- Arborescence et système de fichiers
- Substitutions et variables
- Mécanismes d'entrées/sorties
- Commandes avancées

Emacs

- Edition de texte
- Exercices

Unix/Bash

- Scripts - Scripts avancés

LaTeX

- Introduction
- AMS-LaTeX
- Afficher du code

Et pour pratiquer / réviser

- Pour pratiquer / s'informer

Utilisation des logiciels - Payants

- utiliser un logiciel payant sans l'avoir payé est : illégal
- utiliser un logiciel sous licence sans licence légale est : illégal
- utiliser un logiciel piraté est : illégal
- utiliser un logiciel sans avoir été autorisé est : illégal
- vous êtes responsable des logiciels que vous utilisez
- surtout ceux qui sont installés sur votre compte Enseirb-Matmeca

Utilisation des logiciels - Gratuits

- ne pas confondre “gratuit” et “libre”
- gratuit = libre d’utilisation
- gratuit (et pas libre) $\neq$ accès au code source
- gratuit $\approx$
  - publicité ?
  - présence de virus ?
  - intégrité des données utilisateurs ?
  - sûreté de fonctionnement ?
  - possible aussi avec les logiciels payants

Souvent, si c’est gratuit, c’est l’utilisateur le produit !

Logiciels libres (*open source*)

- ne pas confondre “gratuit” et “libre”
- libre = utiliser, étudier, modifier, redistribuer
  → il existe toutes sortes de licences libres (GPL, MIT, BSD, etc...,) [1]
- libre = accès au code source (correction, sûreté, etc)
- enjeux sociétaux : intérêt pédagogique, intérêt scientifique, diffusion des connaissances, partage des savoirs, égalité devant le logiciel.
- ouverture des formats, échange de données
- alternative aux géants de l'édition logiciel

A l'Enseirb-Matmeca

- tous les outils nécessaires à votre formation sont (a priori) libres
- tous les outils étudiés dans ce module sont libres

⚠Toujours vérifier la licence d'un logiciel utilisé (libre ou pas) et la respecter !

1. cf `https://www.gnu.org/licenses/license-list.fr.html`

Ressources informatiques à l'Enseirb-Matmeca

Une fois le compte validé

- Gestion du compte sur : https://ccc.bordeaux-inp.fr
- Une addresse mail login@enseirb-matmeca.fr.
- Un compte utilisateur ~login avec un espace disque limité
- Deux sites Internet personnels.

⚠ Attention

- choisir un très bon mot de passe
  - https://www.cybermalveillance.gouv.fr/tous-nos-contenus/bonnes-pratiques/mots-de-passe
  - https://www.francenum.gouv.fr/magazine-du-numerique/combien-de-temps-un-pirate-met-il-pour-trouver-votre-mot-de-passe-co
- toujours penser à se déconnecter → pourquoi ?

Le bon usage de l'Internet :
https://fr.wikipedia.org/wiki/Netiquette

Quelques conseils sur la sécurité numérique :
https://moodle.bordeaux-inp.fr/course/view.php?id=3839

Le mail à l'Enseirb-Matmeca

- Plusieurs alias mail disponibles :
  - `login@enseirb-matmeca.fr`, `prenom.nom@enseirb-matmeca.fr`,
  - `login@bordeaux-inp.fr`, `prenom.nom@bordeaux-inp.fr`,
- Rq : `bordeaux-inp.fr` sont redirigées sur `enseirb-matmeca.fr` par défaut. **A ne pas changer**
- Utilisez là comme votre **adresse professionnelle** et non personnelle (idem avec le mail de l'entreprise)
- Savoir utiliser les différents champs `to:`, `cc:`, `cci:` (`bcc:`)
  → on ne rend pas visible l'adresse email d'une personne sans son autorisation
- Un mail possède toujours un **sujet**
- Un mail commence toujours par : *une formule de politesse* et se termine toujours par *une formule de politesse* et une *signature*

☞ Conseils :

- (sauf bonne raison) on ne transfère pas le contenu d'un mail d'une autre personne sans son autorisation
- on ne répond pas à un mail sous l'effet de la colère, on attend un peu...

Le mail à l'Enseirb-Matmeca

- Ne pas faire de `mail-all`
- Ne pas répondre aux `mail-all` inutiles par un `reply-all`
- Webmail : `https://partage.bordeaux-inp.fr`
- Utiliser de préférence un client mail (e.g. *thunderbird*)
- Pièces jointes : **éviter** les gros fichiers, et/ou fichiers non inter-opérables (utiliser par exemple `https://filesender.renater.fr/`)
- Savoir purger son courrier, l'effacer, le ranger
- Faire des filtres

⚠ Attention aussi à votre attitude sur les réseaux sociaux !

Le compte à l'Enseirb-Matmeca

- Accessible via `~login`
- Vous êtes (légalement) responsable de votre compte
- Votre compte ne doit pas être prêté (il existe des solutions, voir après)
- L'espace disque est limité : attention au images, vidéos, logiciels installés, sauvegarde des mails
- Attention au contenu : problème de licence logiciel, peer-to-peer,
- Espace disque : sous linux commande `du -sh ~`
- Compte accessible via l'extérieur par `ssh` sur la machine `ssh.enseirb-matmeca.fr`
  - Travailler sur n'importe quelle machine dont vous avez retenu le nom
  - Ne pas travailler sur la frontale `ssh.enseirb-matmeca.fr`

Le site personnel à l'Enseirb-Matmeca

Possibilité d'avoir deux sites internet :

un site visible de l'extérieur

- https://login.zzz.enseirb-matmeca.fr
- Seulement en html statique.

un site visible seulement sur le réseau local

- http://login.zzzpedago.enseirb-matmeca.fr
- Accès à php et à une base de données mysql
- Utilisation pédagogique

Transfert de fichiers par webdav (cf documentation dans l'ENT)

⚠ espace professionnel et non personnel !

Comment travailler depuis chez vous ?

- Accès SSH aux machines de l'Enseirb-Matmeca(Putty / client ssh). Cf plus loin.
- Installation de linux sur votre machine :
  - Comme seul système d'exploitation (tout simple)
  - En dual boot (Windows et linux côte à côte)
    → un peu technique (à cause de l'UEFI), mais possible
  - Sur une clé USB bootable
  - En machine virtuelle, une image pour virtualbox est mise à votre disposition pour débuter plus facilement

    https://thor.enseirb-matmeca.fr/ruby/docs/teaching/vmlinux

Et l'IA ?

- Le terme "IA" est très flou : symbolique ? générative ? recherche opérationnelle ? programme informatique ? Généralement associé à IA générative.
- Exemples d'IA génératives de type LLM (Large Language Model) : chatGPT, Le Chat de Mistral, Claude, Gemini, Grok, Llama, Qwen, Deepseek, ...
- De formidables outils, mais attention !...
  - Attention au risque de plagiat
  - Attention à ne pas transmettre des informations personnelles/confidentielles
  - Attention à respecter le règlement intérieur de l'école ainsi que les consignes de l'enseignant(e)
  - Aucune certitude que le résultat obtenu soit correct
  - Souvent trouver les erreurs nécessite une réelle expertise
  - "Questions" sur la méthodologie d'apprentissage
  - Vous êtes (légalement) responsable du texte/code utilisé (*c'est pas moi, c'est l'IA...*)
- ⚠ depuis l'été 2026,il est obligatoire de signaler l'usage de l'IA par un logo :



(https://digital-strategy.ec.europa.eu/fr/policies/eu-icons-labelling-ai-generated-content)

En cas de souci d'ordre informatique à l'Enseirb-Matmeca

Demande d'assistance :

https://assistance.bordeaux-inp.fr/

Documentation :

https://ent.bordeaux-inp.fr/

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 [2])
- 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

2. 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 :

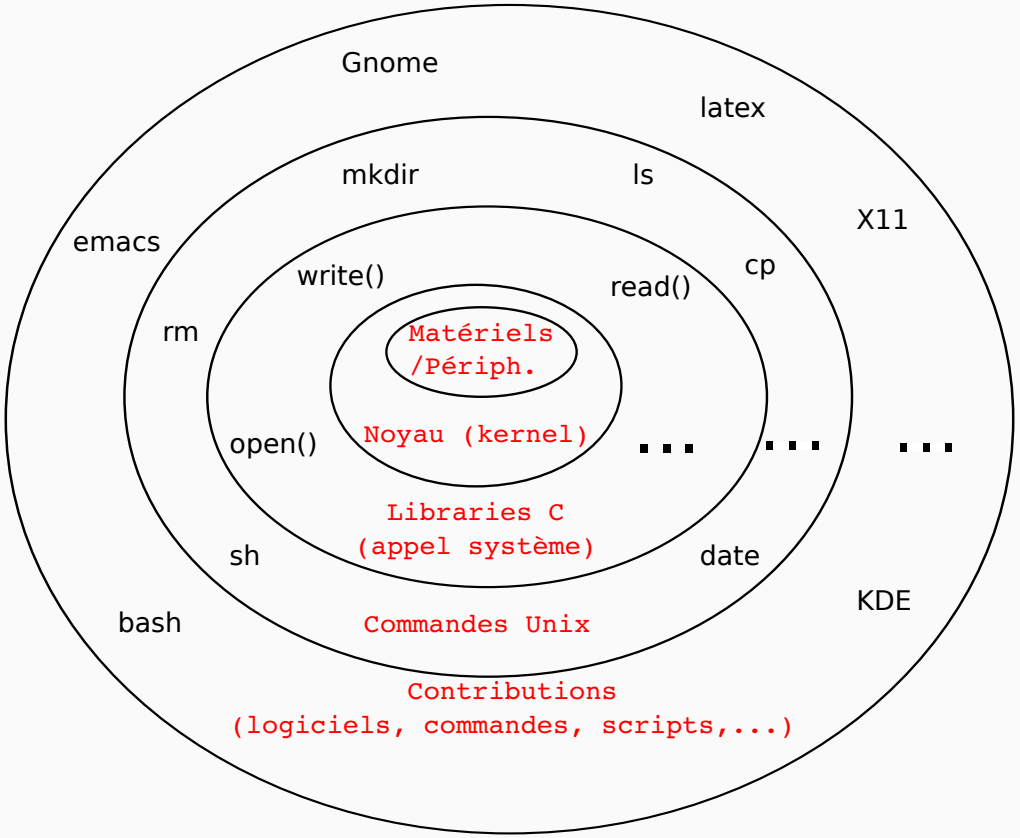
- système à temps partagé : exécution concurrente de processus, ordonnancement
- système multi-utilisateur : compte, connexion (login/password), environnement personnalisé
- partage des ressources : arborescence globale des fichiers, mécanismes de protection
- gestion des communications : entre processus,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

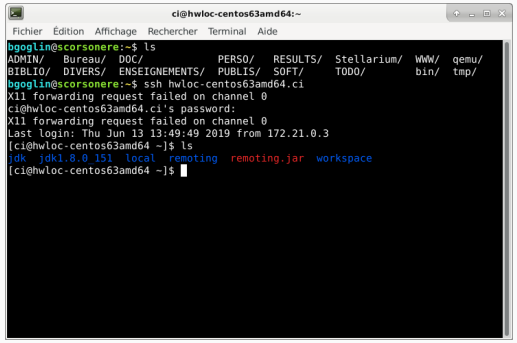
https://doc.ubuntu-fr.org/raccourcis_clavier

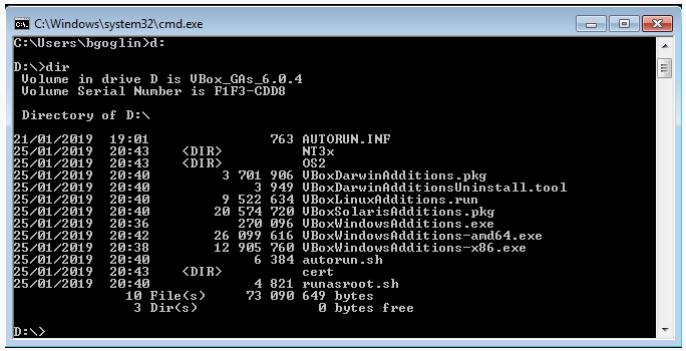
Unix/Bash

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

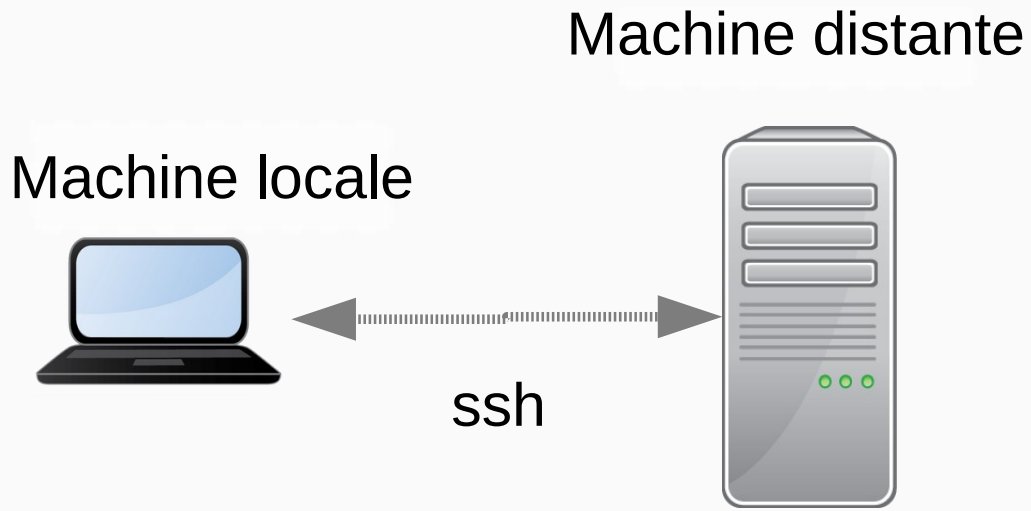
Sur Unix, chaque utilisateur est membre d'un ou plusieurs groupes (qui peuvent être composés d'une seul utilisateur)

- 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

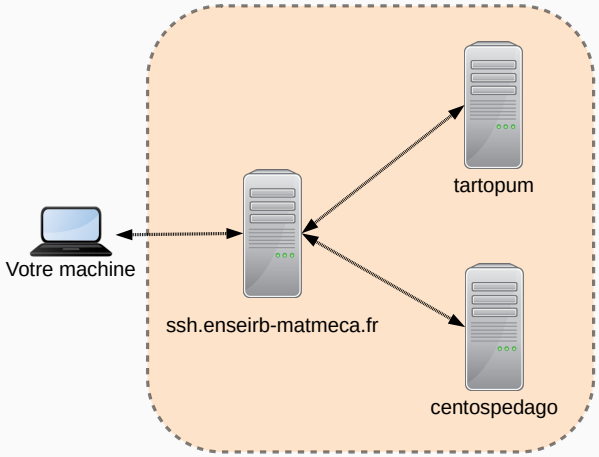
Travail à distance / SSH



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 31.

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

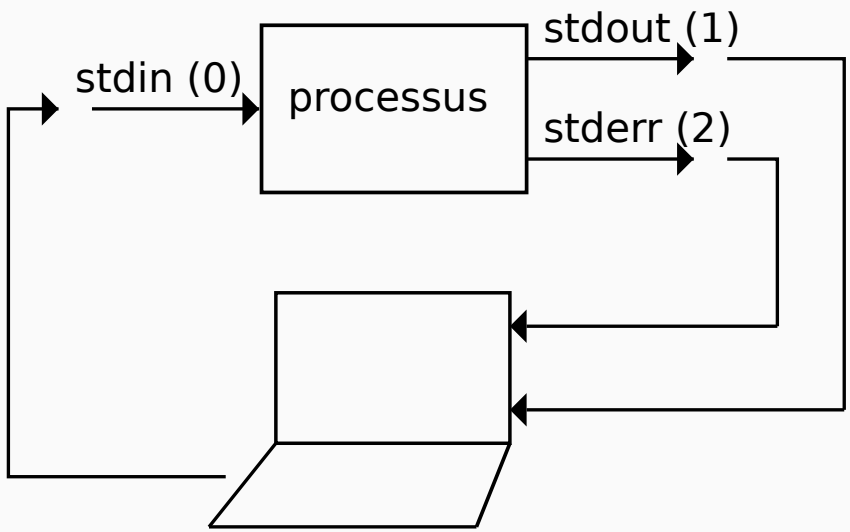
Mécanismes d'entrées/sorties

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)

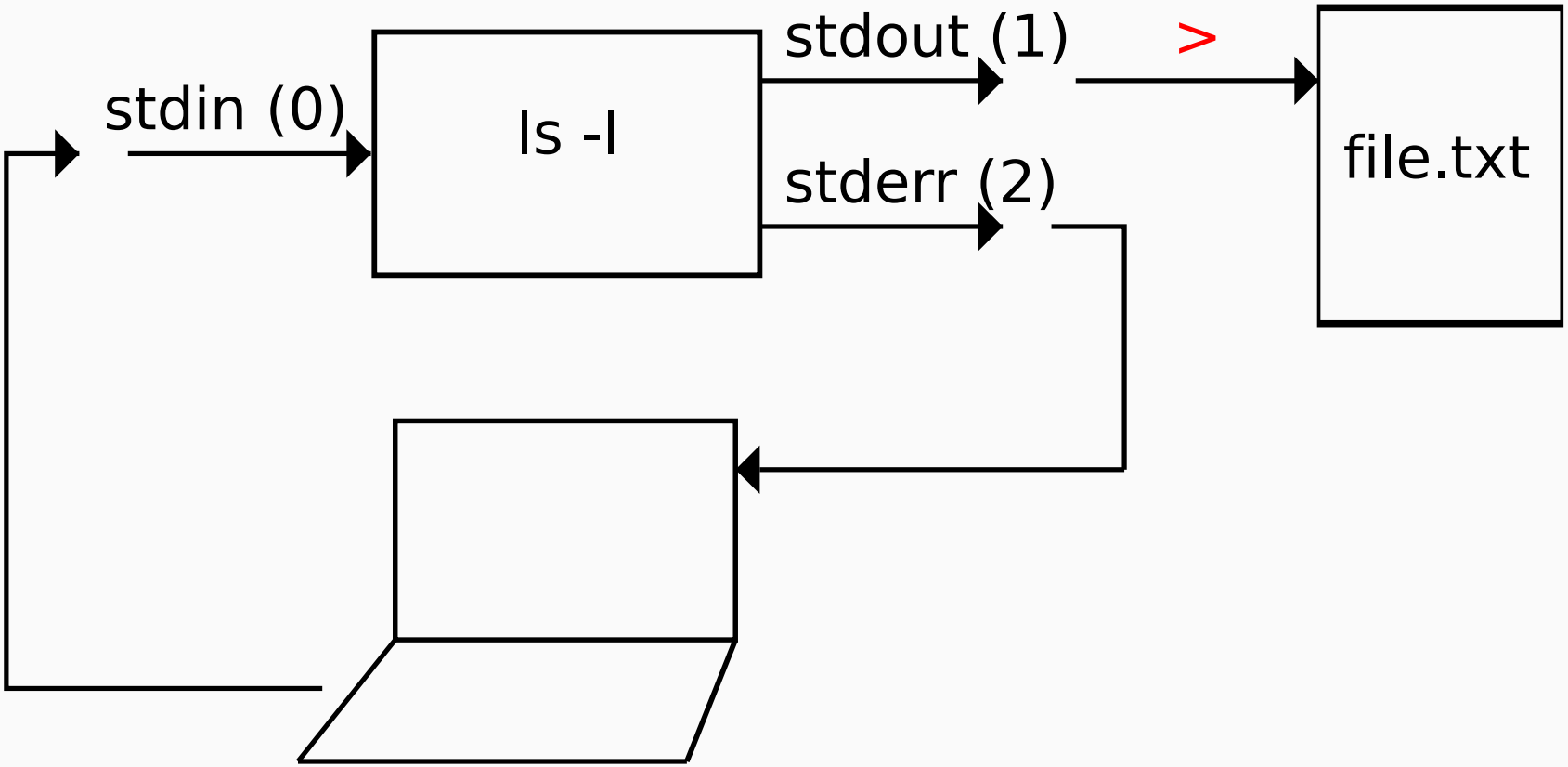
Il est possible de donner plusieurs variables en argument :

```
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)

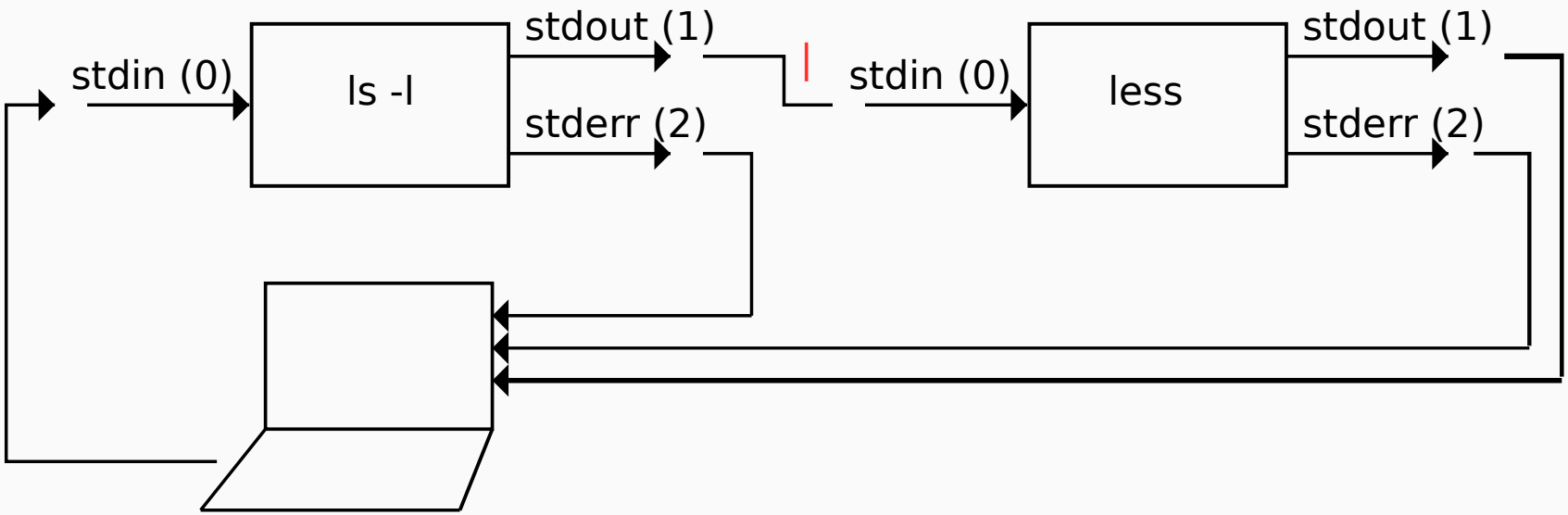
Rediriger un processus (programme) vers un fichier via '>'.
Ex : `ls -l > file.txt`



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`

Emacs

Edition de texte

Documentation

Lien vers le cours :

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

Un peu de lecture :

```
https://rollet.zzz.bordeaux-inp.fr/IntroUnix/documentation/refcard-emacs.pdf
```

Introduction

`emacs` :

- est un logiciel permettant de saisir et de modifier du texte
- n'est pas un traitement de texte
- permet de manipuler des fichiers contenant du texte au sens large
- possède les fonctions classiques : déplacement dans les fichiers, recherche, positionnement, copier/couper/coller, ...
- s'interface avec le système : parcours de répertoires, lecture de courrier, compilation, etc
- possède de nombreuse bibliothèques et modes spécialisés
- utilise un mode d'édition adapté au type de fichier ouvert (automatique ou manuel)
- environnement extensible et configurable ; programmation en `lisp`
- est un logiciel libre
- peut s'utiliser sans fenêtre (mode `no-window`), sans souris

Alternatives (?) : `nano, pico, vi, vim, VSCode, gedit, ...`

Principales façons de lancer emacs en ligne de commande

- `emacs &` → lance emacs seul en tâche de fond
- `emacs file &` → emacs ouvre le fichier (si le fichier n'existe pas il le crée)
- `emacs dir/ &` → emacs ouvre le répertoire et montre son contenu.

Lexique (1)

- **buffer** (tampon) : zone mémoire dans laquelle emacs copie le fichier
- **fenêtre** : écran dans lequel emacs affiche le fichier. Emacs est capable de diviser l'écran en plusieurs fenêtres
- chaque fenêtre est munie d'une **ligne de mode** : position du curseur, nom du buffer, buffer modifié, ...
- en bas de l'écran, se trouve le **mini-buffer** utilisé pour effectuer des opérations d'entrée/sortie, affiche des messages, ...
- le cycle d'édition classique : ouverture d'un fichier dans un buffer, édition (modification) du fichier, sauvegarde (il faut sauvegarder le plus souvent possible)

Dans la suite :

- `C` : Contrôle, touche 'CTRL'
- `M` : Méta, touche 'ESC' ou ALT
- `C-x` : Appuyer sur 'CTRL' et 'x' simultanément
- `M-x` : Appuyer sur 'ALT' et 'x' simultanément, ou 'ESC' puis 'x'

Lexique (2)

- **clé** : suite de caractères interprétée comme un tout. Certains caractères sont des **préfixes** et ne constituent pas des clés : `C-c, C-x, C-h, ESC`, i.e., ils attendent une suite dans la séquence de caractères.
- **commande** : toute action effectuée dans emacs est le résultat d'une commande (même l'insertion de caractère). De nombreuses commandes sont liées à des clés.

  Ex. :
  - `forward-char C-f` ;
  - `backward-char C-b` ;
  - `save-buffers-kill-emacs C-x C-c`
- l'exécution d'une commande se fait via :
  - la clé associée (ex. : `C-f`),
  - le `mini-buffer` en tapant le nom de la commande (ex. `M-x forward-char`)
- chaque commande est définie par une fonction en `lisp`

Remarque

- La complétion est souvent très utile dans le `mini-buffer`
- ⚠ Éviter d'ouvrir deux instances d'`emacs` sur un même fichier !

Exercice : lancer la commande `snake` (et quitter avec le raccourci `C-x k`)

Quelques commandes/clés de base (1)

Commandes sur les fichiers/buffers

- ouvrir un fichier : `C-x C-f`
- enregistrer un fichier : `C-x C-s`
- enregistrer sous : `C-x C-w`

Recherche/remplacement de texte

- recherche incrémentale en avant `C-s` et en arrière `C-r`
- remplacer un texte interactif `M-x query-replace`, `M-%`
- remplacer toute les occurrences `M-x replace-string`
- complétion d'un mot existant dans les buffers ayant le même préfixe `M-/`

Exercice : Chargement d'un fichier

Tous les exercices suivants sont à réaliser sans la souris

- exécuter la commande `find-file` à l'aide de sa clé. Emacs affiche dans le mini-buffer le message : `Find file : ~/`
- essayer `esc esc esc`. Commenter
- relancer `find-file`, mais cette fois sans utiliser le raccourci
- essayer `tab tab`. Commenter.
- Récupérer le fichier `https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/gnu.txt` (en dehors d'emacs), l'enregistrer dans votre répertoire `~/IntroUnix`.
- à l'aide de la complétion, allez chercher le fichier `gnu.txt` et l'ouvrir dans emacs. Quitter emacs, et ouvrir le même fichier avec la commande `emacs ~/IntroUnix/gnu.txt`
- écrire un peu de texte à la suite, et enregistrer le contenu.
- écrire encore un peu de texte, et enregistrer le contenu du buffer à l'adresse `~/IntroUnix/gnu2.txt`
- rechercher toutes les occurrences du mot `GNU`, remplacer toutes les occurrences du mot `public` par le mot `private`.
- à la fin du fichier, taper `Fou` puis `M-/`. Que s'est-il passé ?
- fermer le buffer. Quelle est la commande à exécuter ? Quelle est sa clé ?

Quelques commandes/clés de base (2)

Correction orthographique

- `M-x ispell-change-dictionary` pour modifier le dictionnaire
- `M-x flyspell-mode` analyse à la volée (`M-Tab` pour corriger automatiquement)
- `M-x ispell-buffer` analyse le buffer (correction auto possible via un menu idoine)

Mise en forme

- `M-x auto-fill-mode` ajouter automatiquement un saut de ligne en fin de ligne

Exercice

- Ouvrir à nouveau le fichier `gnu.txt`.
- Le rendre un peu plus "lisible" grâce à la commande `auto-fill-mode`
- Repérer, et si possible corriger tous les fautes d'orthographe, sauvegarder.
- Passer en mode *Analyse à la volée ("flyspell")*
- Rajouter un court paragraphe en anglais avec quelques fautes.

Quelques commandes/clés de base (3) : édition

Quelques commandes d'édition

- sélectionner tout le buffer : `C-x h`
- copier/couper/coller/couper une sélection :
  - `M-w` : copier (vers le kill ring)
  - `C-w` : couper (et placer la sélection dans le kill ring)
  - `C-y` : coller (depuis le kill ring)
  - `C-k` : couper (et placer la sélection dans le kill ring) la (fin de la) ligne
- coller depuis le kill ring : `C-y`, puis `M-y` autant de fois que désiré pour parcourir l'anneau de ce qui a été coupé
- annuler l'action précédente : `C-_`
- `C-u n char` répète n fois char (n=4 par défaut, char peut être une clé)

Exercice (toujours sans la souris)

- Reprendre le fichier `gnu.txt`. Copier le 1er paragraphe à la fin.
- Couper/coller le 2è pargraphe à la fin.
- Coller à nouveau le 1er paragraphe à la fin (sans retourner au début).
- Annuler la dernière action.

Quelques commandes/clés de base (4) : manipulation de fenêtres

Commande de manipulation des fenêtres

- effacer cette fenêtre : `C-x 0`
- effacer toutes les autres fenêtres : `C-x 1`
- split fenêtre horizontal : `C-x 2`
- split fenêtre vertical : `C-x 3`
- déplacer le curseur vers une autre fenêtre : `C-x o`
- changer de buffer : `C-x b`

Exercice (toujours sans la souris)

- Reprendre le fichier `gnu.txt`. Ouvrir une autre fenêtre (split vertical) contenant le fichier `/etc/passwd`. Copier la première ligne du ficher `/etc/passwd` à la fin du fichier `gnu.txt`.
- Revenir à une seule fenêtre contenant le fichier `gnu.txt`. Ouvrir une 2è fenêtre (split horizontal) sur ce même fichier. Se placer à la fin. Ajouter une ligne et sauvegarder. Cette modification a t'elle été prise en compte dans l'autre fenêtre ?

Lexique (3) : liaisons

liaison : on peut modifier dynamiquement une liaison entre une clé et une commande (valables pour tous les buffers) via `global-set-key`.
→ cette modification est temporaire.

- `global-unset-key` pour annuler,
- `C-h k: describe-key` pour vérifier l'existence,
- `C-h b: describe-bindings` pour lister l'ensemble des clés.
- `describe-function` pour obtenir de l'aide sur une commande (notamment sa clé)

Exercice

- Faire en sorte que la commande `C-l` lance automatiquement la commande `goto-line`. Essayer.
- D'après la documentation, à quoi sert la commande `compile`. A partir de quelle version a t'elle été introduite ?
- Associer à la touche `f4` la commande `compile`. Essayer. S'assurer que la clé existe bien (`describe-key`).
- Annuler la dernière association
- Quelle commande permet de passer un paragraphe en majuscule ? Vérifier. Si vous utilisez la clé associée, `emacs` vous demander confirmation. Pourquoi ?

Lexique (4) : macros

macro : suite de commandes clavier définie par l'utilisateur.

- `C-x (` / `C-x )` : début/fin de définition.
  ⚠ attention, il s'agit de la touche "parenthèse" du clavier ici !
- `C-x e` : exécute la dernière macro définie.
- `M-x name-last-kbd-macro` : permet de nommer la macro (définie en dernier).
- `M-x` *unemacro* pour exécuter la macro nommée *unemacro*.

Macros : exercices

- Définir une macro permettant de placer les guillemets anglais (") autour du mot sur lequel est positionné le curseur (utiliser les touches `M-b` et `M-f` pour se déplacer en début et fin de mot)
- Nommer la macro en `insert-quote`. La tester sur le fichier `gnu.txt` avec `M-x insert-quote`.
- Vérifier que la clé `M-"` est libre et la lier globalement à la macro `insert-quote`. Tester.
- Ouvrir un nouveau fichier (`C-x C-f` puis saisir l'adresse d'un fichier inexistant)
- Saisir deux lignes de texte en minuscules, se placer au début du fichier
- Ecrire une macro qui transforme complètement en majuscules le 1er mot de la ligne, et ajoute une majuscule au 3è, puis replace le curseur en début de ligne. Pour cela, on pourra utiliser les commandes `forward-word`, `upcase-word` et `capitalize-word`
- Appliquer cette macro sur quelques lignes du fichier `gnu.txt`

Personnalisation `.emacs`

- si un fichier de configuration `~/.emacs` se trouve à la racine du `home` alors à chaque lancement, le fichier de configuration est chargé et permet de configurer emacs.
- ce fichier contient une liste de commandes écrites en `lisp`.
- ⚠ toujours en faire une sauvegarde avant de le modifier

Quelques exemples :

```
;; a comment
(set-background-color "gray")
(global-set-key (kbd "C-l") 'goto-line) ;; goto line
(global-set-key (kbd "C-c c") 'comment-region) ;; comment region
(global-set-key (kbd "C-c u") 'uncomment-region) ;; uncomment
(global-set-key (kbd "C-c i") 'indent-region) ;; indent
(show-paren-mode t) ;; show parenthesis mode
(setq inhibit-startup-message t) ;; no startup screen
```

Exercice

Pensez à sauvegarder le fichier `.emacs` avant toute modification. Fixer la couleur de fond que vous souhaitez, associer à la clé `C-l` le lancement de la commande `goto-line`. Vérifier que les changements ont bien été pris en compte. (Vous pourrez configurer le reste plus tard).

Gestionnaire de paquetages

emacs est équipé d'un gestionnaire de paquetages très complet

- Pour installer un nouveau paquetage[3] :
  - `M-x package-refresh-contents` ← mettre à jour la base de données des paquetages
  - `M-x package-install packageName` ← installer la paquetage
  - (`M-x package-delete packageName` ← désinstaller un paquetage)

  Exercices :

  - installer le paquetage `2048-game`, le lancer, quitter (`C-x k`)
  - installer le paquetage `doom-themes`, installer le thème de votre choix (commande `load-theme`)
  - installer le paquetage `which-key`, l'activer avec la commande `which-key-mode` et l'essayer (`c-x` puis attendre un peu)

⚠ les paquetages ne seront (a priori) pas disponibles en examen sur machine

3. la marche à suivre précise est en commentaires dans votre fichier `.emacs`

Note : gestionnaire de paquetage

Contenu du fichier `.emacs` pour installer des paquets :

```
;; Configuration des plugins de Melpa
(require 'package)
(add-to-list 'package-archives '("melpa" .
"https://melpa.org/packages/"))
(package-initialize)
;; Les mises à jour de la base de paquetages se font avec la commande
;; M-x package-refresh-contents
;; ;; Si les mises à jours des paquetages ne fonctionnent pas à cause
;; d'un 'Failed to verify signature archive-contents.sig', il suffit de
:
;;
;; 1) positionner la variable suivante a nil
;; M-x set-variable package-check-signature nil
;; 2) rafraichir la base de paquetages
;; M-x package-refresh-contents
;; 3) installer le paquetage de la cle GPG necessaire
;; M-x package-install gnu-elpa-keyring-update
```

Exemple de thème : `load-theme RET doom-nord RET`

Navigation dans `which-key` : `c-h n/p`, refaire le `c-h` à chaque fois (pas très pratique)

Emacs

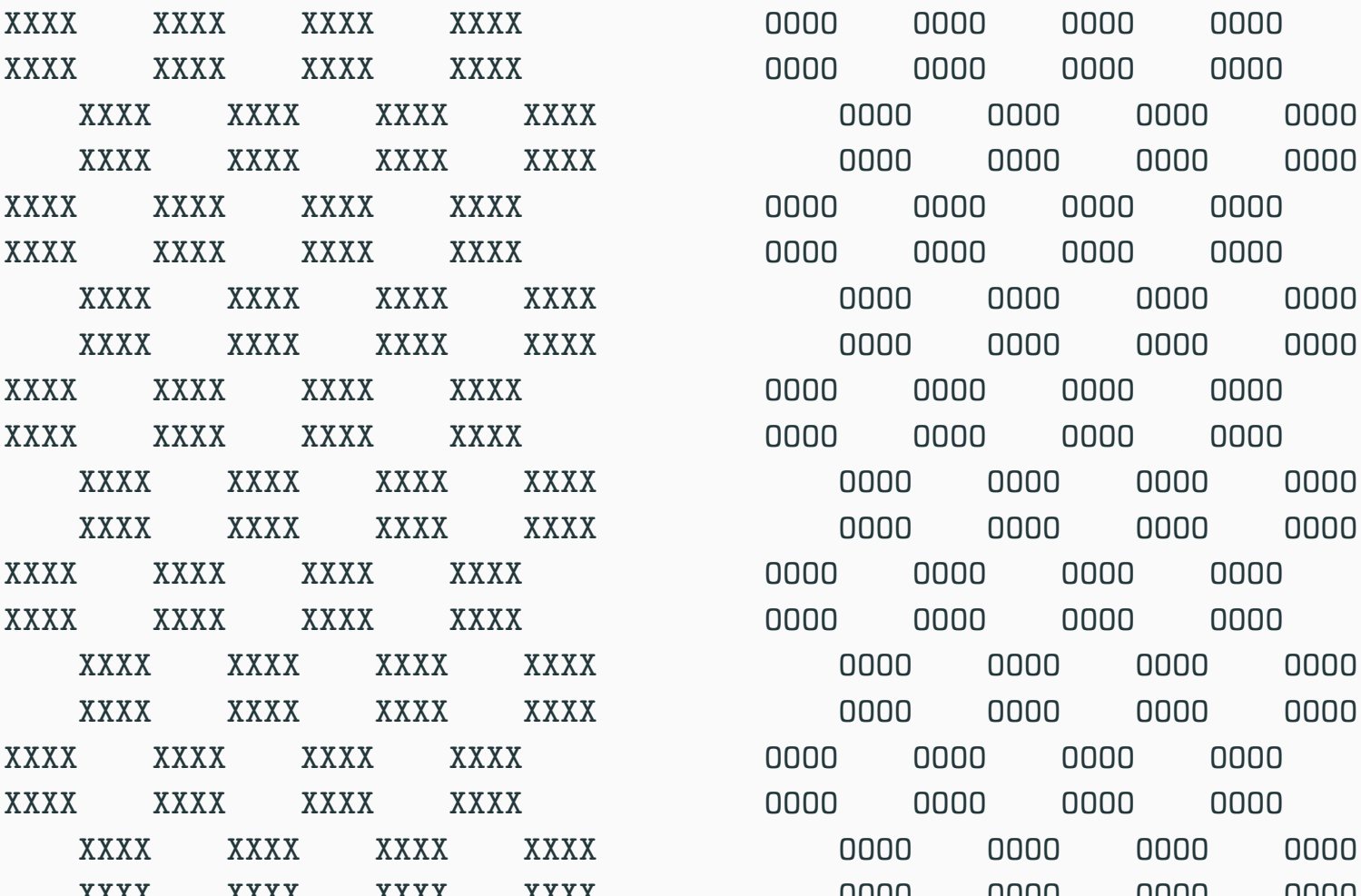
Exercices

Tous les exercices sont à faire **SANS** la souris

Exercice : Damier

- éditer un nouveau fichier
- en un minimum de manipulation réaliser le damier de gauche puis celui de droite (en partant de celui de gauche)

  Question optionnelle : est-ce qu'on pourrait passer du damier de gauche à celui de droite en ligne de commande ?



Exemple : programmation en C

- Récupérer le fichier `https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/tictactoeBugged.c` et l'ouvrir avec `emacs`
- Utiliser la touche `TAB` sur les 6 premières lignes. Que remarquez-vous ?
- Indenter complètement le code (aide : la commande `C-x h` permet de sélectionner tout le buffer)
- Compiler avec la commande `compile`. Il y a des erreurs. Expliquer dans quelle mesure les couleurs et l'indentation permettent d'aider à trouver ces erreurs
- Corriger le code et le compiler. Lancer le programme dans emacs, à l'aide d'un `shell`.
- Ajouter un compteur pour limiter le nombre de parties.

Pour réviser un peu

Quelques exercices pour vous entraîner / réviser :
`https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/exos_emacs-sujet.pdf`

Unix/Bash

Scripts - Scripts avancés

Avant propos

Lien vers le cours :

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

La programmation en `shell`, comme tout langage de programmation, nécessite des structures de contrôle avec des syntaxes particulières, i.e., la façon d'écrire ces structures.

Attention : la programmation en shell nécessite beaucoup de rigueur du programmeur au niveau de la syntaxe. Le bash est très sensible aux espaces (dont il se sert comme délimiteur) et aux sauts de ligne

Scripts `Shell`

- Programme écrit dans un langage interprétable par un interpréteur de commande (pas de phase de compilation)
- `bash` est à la fois le nom de l'interpréteur de commande et le langage de programmation interprété
- Un script `shell` simple peut être une suite de commandes écrite dans un fichier qui est exécutable par le système
- Un script `shell` permet de créer des programmes exécutant une série de commandes qui pourraient être pénibles et répétitives à saisir au clavier

Exécution

- Une manière d'exécuter un programme `shell` est de le lancer en ligne de commande et de spécifier quel interpréteur est capable de lire ce langage de programmation (on parle d'interprétation).
- Script : `myscript.sh` (convention `.sh`) alors
```
echo "echo Bonjour" > myscript.sh
bash myscript.sh
Bonjour
```
- Pour éviter d'indiquer explicitement l'interpréteur, le script `shell` peut commencer par une ligne spéciale qui va indiquer au shell quel interpréteur de commande utiliser.
- Cette ligne est le *shebang* (contraction de `shell` et de `bang` : !)
- Un script commence toujours par `#!cheminverslinterpreteur`.
- Rendre le script exécutable, i.e., lui donner le droit d'exécution (`x`).

Exécution (Exemple)

- Ecrire dans un fichier `myscript.sh` le code suivant (utiliser par exemple l'éditeur de texte `nano`) :

```
#!/bin/bash
echo "Bonjour"
```

- Ajouter les droits d'exécution : `chmod +x myscript.sh`
- Exécuter le script : `./myscript.sh`
  → Affiche `Bonjour`

Arguments

- Un script `shell` peut prendre en paramètre des arguments.
- Lors de l'évaluation de la commande par le `shell`, ce dernier coupe les chaînes de caractères selon différents séparateurs, le plus courant étant l'espace
- Il stocke ensuite les différents token dans un vecteur. Le programmeur peut accéder aux éléments de ce vecteur.

Les motifs d'accès sont les suivants

- `$#` : le nombre d'éléments courant dans le vecteur
- `$1 ... $9` : accès au premier, deuxième, . . .neuvième
- `$*` : liste de tous les éléments

La commande `shift` permet de décaler la liste des arguments d'une position vers la gauche.

Il existe d'autres motifs associés à `$`

- `$0` : le nom de la commande courante
- `$?` : valeur de retour de la dernière commande
- `$$` : pid de la commande courante

Exercice : Ecrire un script `commands.sh` qui affiche toutes ces valeurs. Le tester avec `commands.sh Hello my friends` puis `commands.sh Hello "my friends"`. Expliquer.

Commande `test`

La commande `test`, possède deux syntaxes :
`test arg_1 arg_2 ...` et `[ arg_1 arg_2 ... ]` (⚠ne pas oublier les espaces)

Elle permet de tester :

1. l'existence et la nature d'un fichier, par exemple
   - `-e chemin` : si chemin est un fichier existant
   - `-f chemin` : si chemin est un fichier existant et est un fichier ordinaire
   - `-d chemin` si chemin est un fichier existant et est un répertoire
   - `-r chemin` : si chemin est un fichier existant et est accessible en lecture
   - `-w chemin` : si chemin est un fichier existant et est accessible en écriture
2. test d'égalité de deux chaînes de caractères
   - `chn_1 = chn_2`
   - `chn_1 != chn_2`
   - Chaîne vide : `-z chn`
3. comparaison de nombres
   - `chn_1 -eq chn_2` : $=$
   - `chn_1 -ne chn_2` : $\neq$
   - `chn_1 -gt chn_2` : $>$
   - `chn_1 -ge chn_2` : $\geq$
   - `chn_1 -lt chn_2` : $<$
   - `chn_1 -le chn_2` : $\leq$

Négation : `! EXPRESSION` → `EXPRESSION` est fausse
Pour plus de détails cf le man : `man test`

Conditions (if)

Structures de contrôle de type conditionnelle

- **Si** quelque-chose-de-vrai **alors :** execute-bloc-1 **sinon :** execute-bloc-2
- et du type aiguillage par rapport à un motif reconnu

([ ] = optionnel, `<liste>` liste de commande, pas forcément un test)

Syntaxe

```
if <liste>
then
  <liste>
[ elif <liste>
then
  <liste>
        ]
[ else
    <liste> ]
fi
```

Exemple

```
if [ $USER = "$1" ]
then
  echo $USER c\'est moi
else
  echo c\'est pas moi
fi
```

Conditions (if) - Remarques

Il est possible d'écrire tout ou partie du test sur une seule ligne

Syntaxe

```
if <liste>; then <liste>; [ elif <liste>; then <liste>; ]
[ else <liste>; ] fi
```

Exemple

```
if [ $USER = "$1" ]; then echo $USER c'est moi;
else echo c\'est pas moi; fi
```

Notez aussi les guillemets autour de `$1`. A quoi servent-ils ?
Indice : Essayer l'exemple précédent avec "$1" ou $1, avec les paramètres :

- tic tac
- "tic tac"

Conditions (case)

Structure conditionnelle test de type case à la syntaxe suivante (aiguillage de motif)

Syntaxe

```
case <mot> in
  <motif> [ | <motif> ...] )
    <liste>
    ...
    ;;
  ...
esac
```

Exemple

```
case $N in
  mbox )
    echo $N: boxes
    ;;
  * )
    echo $N: unknown type
    ;;
esac
```

Itérations (1)

Structures de contrôle permettant d'appliquer une série de commandes identiques à une liste de variables données en argument.

1. de type `for`, i.e. : pour chaque élément de la liste : appliquer un traitement à cet élément.

Syntaxe

```
for <nom>  in <mot> ...
do
  <liste>
  ...
done
```

⚠En cas d'utilisation combinée de `for` avec `cat`, l'espace fait partie des séparateurs. Cf après.

Exemples

```
for A in 1 2 3 4 5
do
  echo $A
done

for B in *
do
  cp $B $B.old
done

LISTE="1 2 3 4"
for C in $( echo $LISTE ); do ...; done

for D in $( cat inputFile ); do ...; done
```

Itérations (2)

Structures de contrôle permettant d'appliquer une série de commandes identiques à une liste de variables données en argument.

2. de type `while`, i.e. : tant qu'il y a un élément ou tant que la condition est vrai : appliquer un traitement

Syntaxe

```
while <liste_1>
do
  <liste_2>
  ...
done
```

Dans l'exemple, la commande `shift` permet de décaler la liste des arguments vers la gauche, le premier étant perdu, la boucle itère (est répétée) tant que le nombre de paramètres est positif. Ici, on affiche au final l'argument qui suit immédiatement `-o` (s'il existe).

Exemple

```
while [ $# -gt 0 ]
do
  case $1 in
    -o )
      shift
      echo $1
      ;;
  esac
  shift
done
```

Itérations (3)

Autre type de boucle : la boucle `select` → menus numérotés

Syntaxe

```
select var in str1 str2 ... strN
do
   Statement(s) to be executed
done
```

Le script affiche un menu numéroté et attend une saisie clavier stockée dans la variable `var`. L'exemple de droite affiche :

```
1) quit
2) bonjour
3) bonsoir
#?
```

Example

```
select CHOIX in "quit" "bonjour" "bonsoir"
do
   case $CHOIX in
      "quit")
         break
      ;;
      "bonjour"|"bonsoir")
         echo "Hello!"
      ;;
      *)
         echo "Bad choice"
      ;;
   esac
done
```

Trois façons usuelles de sortir d'une boucle `select` :

- `continue` : passer à l'itération suivante
- `break` : sortir de l'itération
- `exit [n]` : sortir du programme avec la valeur `n` (*convention* : 0 si correct)

Déclaration de fonctions

Un programme peut être décomposé en fonctions ; afin de

- modulariser le code
- factoriser le code (éviter de réécrire du code inutile)
- faciliter la maintenance du code

Syntaxe

```
my_function()
{
  #corps de la fonctions
}

my_function #ici on l'appelle
my_function valeur #autre appel
                   #avec un arg
```

Exemple

```
usage()
{
    echo "Erreur de type" $1
    echo "Usage: $(basename $0) [-ab] file"
}

if [ $# -eq 0 ]
then
  usage "Parameter required"
  exit -1
fi
```

- les variables `$1, $2,`... contiennent les valeurs passées en paramètre à la fonction
- ⚠Il faut déclarer la fonction **avant** de l'appeler

Retour sur l'IFS

- En bash, les mots sont découpés selon un délimiteur défini dans la variable `IFS` (Internal Field Separator).
- Par défaut, il contient espace, tabulation et saut de ligne : `IFS=$' \t\n'` (remarquer la notation `$'...'` pour gérer les `\`)
- Il est possible de le modifier → exemple : `IFS=$'\n'` (classique pour lire des fichiers) ou `IFS=";"` (plus rare). Dans ce cas, il est recommandé de sauvergarder sa valeur et de la remettre ensuite : `oldIFS="$IFS" ... IFS="$oldIFS"`

scriptIFS.sh

```
oldIFS="$IFS"
IFS=(v1 ou v2, cf ci-dessous)
for i in $(cat data.txt)
do
    echo [$i]
done
IFS="$oldIFS"
```

data.txt

```
1 2

3 4
```

- v1 : `IFS=' '` affiche :
  ```
  [1]
  [2
  3]
  [4]
  ```
- v2 : `IFS=$'\n'` affiche :
  ```
  [1 2]
  [3 4]
  ```

Lire un fichier ligne par ligne

- Il est fréquent de prendre un fichier en entrée, et d'appliquer un traitement à chaque ligne
- Possibilité : `for i in $(cat input.file); do ... done`
  → attention à la gestion du séparateur (espace vs ligne, ...)
  → exemple : `IFS=$'\n'` (classique) ou `IFS=";"` (plus rare)
  cf slide précédent.
- Autre méthode : `while read line; do COMMAND; done < input.file`
  ⚠ avec `read`, la dernière ligne du fichier `input.file` doit terminer par un saut de ligne pour être prise en compte (il existe des moyens de le prendre en compte, non abordé ici). ⚠ C'est subtil... la plupart des éditeurs de texte rajoutent un `\n` en fin de ligne en sauvegardant au format `.txt` (ou assimilé).
- `-r` n'autorise pas les échappements de caractères (backslashs), par exemple une ligne avec `\n` au milieu sera considérée comme une seule ligne par `read`
- `IFS=` pour effacer le séparateur de champs (seule la fin de la ligne sera prise en compte par le `read`)
- Ce qui donne :
  `while IFS= read -r line; do COMMAND_on $line; done < input.file`
- Adaptable pour lire prendre en entrée une commande dont le résultat contient plusieurs lignes à traiter (utilisation d'un *"here string"*) :
  `while IFS= read -r line; do COMMAND_on $line; done <<< $(COMMAND)`

Exercices : `displayArgs.sh`

- Ecrire un script bash qui affiche tous les arguments passés en paramètre sauf le premier
- Idem, mais qui affiche tous les arguments passés en paramètre sauf le dernier

Exercices : `decompte.sh`

- Ecrire un script bash prenant deux paramètres (*délai* et *début*) qui réalise un décompte jusque à 0 toutes les *délai* secondes, en partant de *début*.
- Ecrire une version avec une boucle `for` et une autre avec boucle `while`.

Exercices : `check_pdf.sh`

- Ecrire un script bash prenant un répertoire en paramètre, et qui pour tous les fichiers terminant par `.pdf` de ce répertoire (recherche non récursive), s'assure qu'il s'agit bien de fichiers pdf. Le script affiche une alerte uniquement en cas de détection de fichier corrompu.
- Ecrire une version avec une boucle `for`.
- Contrainte : ne pas utiliser la commande `find`
- On pourra supposer que les noms de fichiers ne contiennent pas d'espaces (sinon, penser à utiliser des guillements + gestion de l'`IFS`)

Exercices : `notes.sh`

- Ecrire un script qui saisisse en boucle au clavier un nom et une note (format `Nom:note`) sous forme de nombre entier, qui affiche à chaque fois à l'écran la mention correspondante (Ajourné, Passable, Assez Bien, Bien, Très Bien), et sauvegarde l'ensemble dans un fichier `my_notes.csv` (à la suite si le fichier existe déjà). Taper la touche `q` pour quitter.

Exercices : montage audio

- Ecrire un script bash permettant de découper un fichier audio en plusieurs sous-parties. Il prend 3 paramètres :
  - un fichier audio : le fichier à découper
  - un fichier au format csv, comprenant des lignes de la forme `beginning;end;title_of_the_subpart`, les timings sont exprimés sous la forme `HH:MM:SS`
  - une date au format `YYMMDD`
- Pour chaque ligne du fichier csv, il génère un fichier audio au format `.wav`, démarrant et terminant aux instants spécifiés, ayant le nom `YYMMDD_title_of_subpart.wav`
- Indice : utiliser la commande `ffmpeg`
- Contrainte : utiliser une boucle `for` [4]
- Vous pouvez tester votre script avec une interview de Richard Stallman [5] disponible à https://jayandjef.github.io/programming/featured/2025/02/15/an-interview-with-richard-stallman/, et vous pouvez par exemple utiliser le fichier audio à https://jayandjef.github.io/assets/stallman/chocolate.wav avec le fichier `chocolate.csv` fourni ici : https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/chocolate.csv
- Ajouter des contrôle de validité du fichier csv, et du nombre de fichiers créés

4. pour des raisons techniques, la boucle `while` est plus difficile à écrire
5. https://www.stallman.org/

Exercices : `random_files.sh`

- Ecrire un script bash qui prend un répertoire en paramètre, choisit aléatoirement (récursivement) un fichier `.txt` dans ce répertoire, et propose à l'utilisateur de l'ouvrir automatiquement avec un éditeur de texte, de recommencer, ou de quitter.
- Indice : utiliser l'opérateur `select`
- Remarque : on pourrait envisager ce genre d'application avec des fichiers audios, ou vidéos.

Exercices : `mykill.sh`

Il est souvent fastidieux de trouver le pid d'un processus à tuer

- Écrire un script bash qui recherche un processus par son nom
- Récupère le (ou les) pid correspondant
- Envoie le signal sigkill a l'ensemble de ces processus

Exemple d'exécution :

```
~$ ./mykill.sh oneko
```

Exercices : `liste.sh`

- Écrire un script bash qui prenne en entrée un fichier contenant une liste de logins et crée en sortie un fichier avec les noms et prénoms correspondants
- Exemple d'exécution :

  ```
  ~$ ./liste.sh logins.csv resultat.txt
  ```

- Indice : utiliser la commande `finger`. Essayer le script sur la machine `ssh.enseirb-matmeca.fr` (`finger` n'est pas installé sur les machines de la salle).
- Adapter pour écrire une nouvelle version qui fasse la même chose avec les utilisateurs connectés (command `who`)

Exercices : `valid_url.sh`

- Ecrire un script bash qui prend en paramètre un fichier contenant des sites web (des `url` de noms de domaine, sans le `http` devant, ou des adresses IP), et affiche à l'écran s'ils/elles ne sont pas atteignables
- Indice : utiliser la commande `ping` avec l'option `-c1`
- Indice : penser au code de retour de la fonction `ping`

Exercices : `backup.sh`

Écrire un script bash qui

- affiche les n fichiers les plus récemment modifiés dans le répertoire courant (non récursivement)
- demande à l'utilisateur s'il veut créer une archive à partir de ces fichiers (y/n).
- en cas de réponse positif : demande le nom de l'archive,
- en cas de réponse négative : quit
- tout autres réponse : message d'erreur et arrête le programme.

Exemple d'exécution :

```
~$ ./backup.sh 4
backup.sh sum.sh templates.sh myscript.sh
Do you want to archive and compress these files (y/n)?
n
~$ ./backup.sh 4
backup.sh sum.sh templates.sh myscript.sh
Do you want to archive and compress these files (y/n)?
u
unkwon answer, quit
~$ ./backup.sh 4
Do you want to archive and compress these files (y/n)?
y
Enter the archive name:
arxiv
Save in arxiv.tgz
~$
```

Exercices : `printer_ready.sh`

- Ecrire un script bash qui prend en paramètre le nom d'une imprimante de l'Enseirb-Matmeca, considérant par défaut l'imprimante `pontos`, et vous alerte quand cette dernière est prête à imprimer. Pour cela, le script interrogera l'imprimante toutes les 10 secondes.
- Indice : utiliser la commande `lpq`
- Contrainte : ne pas utiliser l'option `+intervalle` de la commande `lpq`
- Pour simplifier, on supposera que les messages sont `"... is ready"` ou `"... is printing"`

Exercices

- Quelques exercices pour vous entraîner / réviser : rendez-vous sur la page `https://cours-mf.gitlabpages.inria.fr/if104/exercices` pour faire les exercices associés au shell

LaTeX

Introduction

Documentation

Lien vers le cours :

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

Un peu de lecture :

`https://rollet.zzz.bordeaux-inp.fr/IntroUnix/documentation/refcard-latex.pdf`

`https://rollet.zzz.bordeaux-inp.fr/IntroUnix/documentation/Apprends_LaTeX.pdf`

Historique

- TEX (Donald E. Knuth) : programme pour des documents (scientifiques)
- TEX grande stabilité, portabilité, absence quasi totale de bogues, respect des usages des typographes et composeurs de textes.
- TEX se prononce "tech" (comme dans tech-nologie), X → χ (chi).
- LATEX (Leslie Lamport) sur-couche basée sur TEX comme système de mise en page avec de nombreux modules

Introduction

LaTeX vs. WYSIWYG

- LaTeX : outil très puissant pour rédiger tous types de documents, notamment scientifiques
- L'auteur → structure logique du document (section, sous-section, etc)
  LaTeX → calcul la mise en page optimale avec les bonnes règles de typographie.
- Information de structure insérée dans le texte avec des **commandes** : distinction entre le fond du document et sa forme.
- Démarche opposée aux traitements de texte WYSIWYG (What You See Is What You Get) : mélange de structure du document et de mise en forme. L'auteur voit à l'écran le résultat final du document.
- LaTeX → pas de résultat final immédiat, nécessite une phase de compilation. **compilateur : programme qui permet de transformer un langage vers un autre langage**. 'code source latex' → LaTeX→ (dvi, ps, pdf, html, etc)

☞ Remarque : ces slides sont rédigés en LaTeX !
☞ Extrait des retours d'année R&I1 précédents : *"LaTeX est vraiment utile..."*

Caractéristiques de LaTeX

Arguments en faveur de LaTeX par rapport aux logiciels WISIWYG

- mise en page professionnelle / respect des règles typographiques
- facilité de la composition des formules mathématiques
- on ne se préoccupe pas de la mise en page, on peut donc se concentrer sur le fond
- gestion automatique des structures complexes telles que : tables des matières, notes de bas de pages, liste des figures, références croisés, bibliographies, index, et bien d'autres.
- possibilité de programmer/très extensible
- multi-plateforme / libre
- extensions conventionnelles : `.tex` (LaTeX), `.bib` (fichier de bibliographie), `.aux` (fichiers auxiliaires), `.sty` (fichiers de style), etc

Remarque

Création de bibliographie : voir le système `bibtex` (fichier `.bib`)

☞ Remarques :

- les autres éditeurs de documents connus (e.g. libreoffice) ont aussi des qualités. A l'avenir, ça sera à vous de faire votre choix en fonction du contexte.
- il est possible d'utiliser d'autres éditeurs de fichiers qu'`emacs` pour rédiger des documents en LaTeX.

Un mot sur Markdown

Une autre alternative à LaTeX, très à la mode

`markdown` = langage de balise, syntaxe simple, facile à lire et à écrire

Un document `markdown` peut être converti en divers formats (e.g. sous linux, on peut utiliser la commande `pandoc`), notamment pdf ou html.
→ `pandoc file.md -o file.pdf` ou `pandoc file.md -o file.html`

Utile pour du protoypage rapide ou des mises en page simples, beaucoup moins de possibilités que LaTeX

Plus d'informations

`https://www.markdownguide.org/getting-started/`
Et un peu de pratique :
`https://www.markdowntutorial.com`

Structure d'un fichier source

LaTeX Langage balisé à base de commande

- **Commande**
  - commence par `\` suivi du nom de la commande et est sensible à la casse
  - syntaxe : `\commandname[options]{parameters}`
  - parfois : `\commandname*[options]{parameters}` action particulière (ex. `\section{}, \section*{}`)
- **Squelette d'un fichier**

```
\documentclass[options]{parameters}
\usepackage[package-options]{package-name} %optionnel
% >>préambule<<
\begin{document}
  le document commence ici
\end{document}
```

- `\documentclass[options]{paramaters}`
  `parameters` : article, report, book, etc ; `options` : 10pt, twocolumn, etc
- `\usepackage[package-options]{package-name}`
  `package-name` : graphicx, amsmath, etc ; `package-options` (personnalisation)
- ligne commençant par `%` = commentaires
  ☞ *pour le utilisateurs d'*emacs*, essayer la commande* `comment-region`
- **Gros documents, plusieurs fichiers**
  `\input{file.tex}` ou `\include{file}` (idem avec saut de page, ⚠sans le `.tex`)

Compilation / visualisation

LaTeX : Langage compilé qui nécessite un compilateur (programme informatique)
Code source LaTeX→ compilateur (`pdflatex`) → fichier visualisable, imprimable

- **Compilation standard avec le programme `pdflatex`**
  - comprend les commandes et instructions du fichier source.
  - phase de compilation crée un ensemble de fichiers : `.pdf` (mise en forme), `.aux` (numéro de pages, figures, références, etc), `.log` (messages d'informations ou d'erreur), `.toc` (table des matières), ...
  - il est souvent nécessaire de compiler le fichier `tex` deux fois pour que les références s'affichent correctement
  - visualisation : `evince` (a priori tout lecteur de pdf peut convenir)
    ☞ `evince` se met automatiquement à jour en cas de changement dans le fichier pdf

  Généralement, on édite le code LaTeX dans son éditeur de texte favori, on le compile dans un terminal (ou avec l'éditeur s'il le permet), tout en maintenant une fenêtre ouverte qui visualise le fichier pdf.

Compilation / visualisation

- **Prise en charge des figures/images**
  - `pdflatex` : `pdf`, `png`, `jpeg`
  - Privilégier les figures/images vectorielles par rapport aux images bitmap
- Edition partagée sur `http://www.overleaf.com`
- **Outil de manipulation des pdf :** `pdftk` / `pdfjam`

  ex : concaténer des pdf
  - avec pdftk : `pdftk a.pdf b.pdf cat output c.pdf`
  - avec pdfjam : `pdfjam a.pdf b.pdf -o c.pdf`

    (`pdfjam` est accessible sur les machines de l'école ici : `/opt/pdfjam/bin/pdfjam`)

Exercices

Récupérer le fichier
`https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/first-doc.tex`
(avec `wget`) et le mettre dans une arborescence locale du type
`~/IntroUnix/latex/compil/first-doc.tex`

1. Compiler le fichier, il y a des erreurs de compilation, interpréter, corriger les erreurs jusqu'à compilation **totale**
2. Vérifier dans le répertoire courant la présence des fichiers auxiliaires. De quels types sont-ils ?
3. Visualiser le document produit. Modifier le fichier source et vérifier la modification dans le document final.
4. Produire un fichier `pdf`. Le visualiser avec `evince`.
5. Effacer tous les fichiers sauf le fichier source. Modifier le fichier source. Compiler avec `pdflatex`. Vérifier la génération des fichiers auxiliaires et du `pdf`. Visualiser le `pdf`. Modifier le fichier source. Recompiler. Visualiser la modification.
6. Créer deux petits fichiers `.tex` contenant chacun une ligne de texte. Utiliser la commande `\input` pour intégrer le 1er, et la commande `\include` pour intégrer le 2è.

first-doc.tex

```
\documentclass{article}
% My first document using a professionnal editor!

\begin{document
This is my first \LaTeX{}} document with a spaces
test and.



A new    \ \ paragraph.

Today is \today.



end{document}
```

Quelques commandes

- Sauts de ligne, de page et césure : `\\`, `\newline`, `\newpage`, `ce\-su\-re`
- Chaînes toutes prêtes : `\today`, `\date`, `\TeX`, `\LaTeX`
- Caractères spéciaux :
  `\c{c}`, `\oe`, `\^o`, `\"i`, `\dots`, `\slash`, `\textbackslash`...
- Espaces : LaTeX calcul l'espacement optimal que les mots doivent avoir.
  - espace insécable, espacement : `~`,`\␣`, `\hspace{m}`, `\vspace{m}`,
  - mesure automatique : `\textwidth`, `\textheight`
- Polices : `\emph{}`, `\textbf{}`, `\texttt{}`, `\textit{}`, `\textsc{}`
- Tailles :
  `\tiny`, `\scriptsize`, `\footnotesize`, `\small`, `\normalsize` (default),
  `\large`, `\Large`, `\LARGE`, `\huge`, `\Huge`
  → Utilisation de blocs, par ex : `{\small petit} normal`

Exercice : Ajouter le texte dans votre fichier qui se rapproche le plus possible du rendu suivant :
The small text is written in big and conversely ;
italic is written in *bold* and conversely.

Quelques commandes

- Structure logique d'un document (toutes ont une version avec `*`)
  - `\part{title}` (`book`)
  - `\chapter{title}` (`book` et `report`)
  - `\section{title}`
  - `\subsection{title}`
  - `\subsubsection{title}`
- Un paragraphe : unité typographique commençant par un alinéa (**obtenu par une ligne vide dans le code source**) = pensée cohérente ou développement d'une idée. Lorsque la réflexion se poursuit, un saut de ligne `\\` suffit (pas d'alinéa).
- Références croisées : `\label{nom}`, `\ref{nom}`, `\pageref{nom}`, `\eqref{nom}`
  ⚠ Ne jamais écrire une référence en toutes lettres !
  Ecrire : `Section~\ref{labelCorrespondant}` et non `Section 1` (pourquoi ?)
- Listes automatiques : `\tableofcontents`, `\listoffigures`, `\listoftables`

→ Créer deux sections dans votre fichier précédent, et faites référence à la première dans la deuxième.

Environnements

Permet de composer du texte dans un contexte particulier
syntaxe générale `\begin{env} contenu \end{env}`

- `center` pour centrer le texte
- Listes : `enumerate, itemize`, utilisation avec `\item`
- Impression verbatim : `verbatim` (on peut aussi utiliser `\verb|text|` où `|` peut être remplacé par n'importe quel caractère)
- Tableau : `\begin{tabular}{description}`
  - `description` : droite, gauche, centré, trait vertical : `r, l, c, |`
  - séparateur de colonne, saut de ligne : `&`, `\\`
  - trait horizontal, trait horizontal de la colonne `i` à `j` : `\hline`, `\cline{i-j}`
  - fusion de colonnes : `\multicolumn{n}{description}{texte}`
    → `n` : nombre de colonnes fusionnées, `description` : idem ci-dessus

Dans votre fichier précédent, ajouter une énumération de trois items, puis donner une grille centrée du jeu de *morpion* de votre choix, par exemple celle-ci dessous à gauche, et le modifier ensuite pour obtenir la version de droite :

X	O	X
O	X	O
O	X	O

	OXO	
X	O	X
O	X	O
O	X	O

Objets flottants

Objets ne pouvant être coupés par un saut de page (figure, tableau, algorithme, etc). LaTeX place automatiquement les objets flottants, là où il y a de la place dans la page.

- **Squelette standard**

```
\begin{...} %figure ou table
  % commandes pour une figure, le tableau ou autre chose
  \caption{legende}
  \label{labelName} % \label se trouve apres \caption
\end{...} % figure ou table
```

- **Important**, un flottant (une figure, un tableau, etc), doit toujours
  - comporter une légende explicative ;
  - être cité, référencé dans le texte, par ex.
    `The figure~\ref{fig:nomdulabel} shows`... ou encore
    `The results are summarized by Table~\ref{tab:nomdulabel}`....
- Par ex : figure, tableau : `\begin{figure}[pref-placement]`, `\begin{table}[pref-placement]`
- Ordre de préférence de placement : `h t b p` (here, top, bottom, page)

Inclusion de figures

- dans le préambule : `\usepackage{graphicx}`, attention au `x`, `graphics` existe aussi
- commande : `\includegraphics[options]{filename}`
  - `options` : permet de régler la taille de la figure
  - préférer les tailles relatives, préservation de l'aspect/ratio
  - jouer sur la largeur par rapport à la largeur du texte : `\includegraphics[width=0.5\textwidth]{filename}`
  - `filename` (avec ou sans l'extension)

Exercice :

→ Dans votre fichier précédent, intégrez le logo de l'Enseirb-Matmeca(disponible à `https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/logo_em.pdf`)

→ L'intégrer dans un environnement `figure`, et y faire référence

Et le français dans tout ça ?

Prise en charge des caractères français, il y a plusieurs façons de faire :

- utiliser les caractères spéciaux `\c{c}`, `\oe`, `\^o`, `\"i`, ...
  (mais un peu pénible à la longue...)
- utiliser `\usepackage[utf8]{inputenc}` en encodant le fichier en `utf8`.
  → indique à LaTeX comment les caractères du fichier .tex sont codés
  → considéré par défaut si non précisé normalement

☞ Pour modifier l'encodage d'un fichier sous `emacs` :

- commande `set-buffer-file-coding-system`

Et le français dans tout ça ?

Prise en charge des caractères français, il y a plusieurs façons de faire :

- utiliser les caractères spéciaux `\c{c}, \oe, \^o, \"i, ...`
  (mais un peu pénible à la longue...)
- utiliser `\usepackage[utf8]{inputenc}` en encodant le fichier en `utf8`.
  → indique à LaTeX comment les caractères du fichier .tex sont codés
  → considéré par défaut si non précisé normalement

☞ Pour modifier l'encodage d'un fichier sous `emacs` :

- commande `set-buffer-file-coding-system`

Le package `fontenc` définit l'encodage de la police :

- `\usepackage[T1]{fontenc}` → T1 est le plus adapté pour les caractères français

Et le français dans tout ça ?

Prise en charge des caractères français, il y a plusieurs façons de faire :

- utiliser les caractères spéciaux `\c{c}, \oe, \^o, \"i, ...`
  (mais un peu pénible à la longue...)
- utiliser `\usepackage[utf8]{inputenc}` en encodant le fichier en `utf8`.
  → indique à LaTeX comment les caractères du fichier .tex sont codés
  → considéré par défaut si non précisé normalement

☞ Pour modifier l'encodage d'un fichier sous `emacs` :

- commande `set-buffer-file-coding-system`

Le package `fontenc` définit l'encodage de la police :

- `\usepackage[T1]{fontenc}` → T1 est le plus adapté pour les caractères français

Le package `babel` apporte un support multilingue (titres (e.g. chapter → chapitre)) et adapte les règles typographiques (césures, etc...) :

- `\usepackage[french]{babel}`

Exercice : reprendre votre fichier précédent, le traduire, et le mettre en forme en langue française. Remarquer que la mise en forme de la date a changé.





Documentation

Un peu de lecture :

`https://rollet.zzz.bordeaux-inp.fr/IntroUnix/documentation/refcard-amslatex.pdf`

- AMS-LaTeX : collection d'extensions et de classes pour saisir des formules mathématiques
- développé par l' *American Mathematical Society*.
- `\usepackage{amsmath, amssymb}`
- symboles mathématiques dans le texte `$...$`, par ex. $a+b=c$: `$a+b=c$`
- hors texte : environnement (flottant) `\begin{equation}...\end{equation}`, par ex : `\begin{equation} a+b=c\end{equation}`

$$a+b=c \tag{1}$$

- pour supprimer la numérotation : `\begin{equation*}...\end{equation*}`

$$a+b=c$$

  Equivalent à `\[ ... \]` ou `$$ ... $$`

- référencer une équation
  1. déposer une étiquette (label) `\label{eq:addition}`
  2. la référencer dans le texte `\eqref{eq:addition}`

  En résumé :

  1. `\begin{equation} \label{eq:addition}a+b=c\end{equation}`
  2. `L'équation \eqref{eq:addition} est une addition`
     → *l'équation* (1) *est une addition*

Commandes en vrac

- `^ _ ^{} _{}` : exposant, indice
- `\sum, \int, \lim, \prod` : somme, intégrale, limite, produit
- `\frac{}{}` : fraction
- les symboles grecs, syntaxe du nom, e.g., `\pi, \epsilon, \Gamma` , ...
- opérateurs, etc. : `=, \neq, \forall, \in, \notin,`...
- `\, \; \quad \qquad` gestion des espaces
- `\sqrt{}, \sqrt[]{}`, racine carré, racine n^{ieme}.
- `\cdots, \ldots, \vdots, \ddots`, différents points de suspensions
- `\text{...}` texte (droit) en mode math
- rq : variables en italique, les fonctions en police droite, e.g., `\sin, \cos, etc`.
- `\partial, \| \| , | |` : dérivée partielle, norme, valeur absolue
- etc.

Lien utile : `http://detexify.kirelabs.org`

Exercice : → écrire sous forme d'équation : $E = MC^2$ puis $a + \beta = 1 + \frac{\beta}{a}$ si $a \neq 0$, et enfin

$$\int_a^b f(x)dx = -\int_b^a f(x)dx$$

LaTeX

Afficher du code

Afficher du code : le package `listings`

Permet d'afficher du code de façon formatée

- `\usepackage{listings}` à ajouter en début de document
- Utilisation : `\begin{lstlisting} some code \end{lstlisting}`
- Nombreuses options possibles, notamment le langage affiché (C, Java, etc...) :
  `\begin{lstlisting}[language=Java]...`
- Possible de configurer une fois pour toutes :
  `\lstset{language=C#, ...}`
- Utilisable intégré dans du texte : `\lstinline|some small code|`
  (les "|" peuvent être remplacés par n'importe quel caractère)
- Intégrer un (morceau de) fichier de code :
  `\lstinputlisting[language=C, firstline=32, lastline=47]{source.c}`

Plus de détails :

`https://ctan.mines-albi.fr/macros/latex/contrib/listings/listings.pdf`

Exercice : afficher le code suivant à la suite de votre ficher LaTeX :

```
int factorielle (int x) {
  if (x == 1)
    return x;
  else
    return x*factorielle(x-1);
}
```

Exercices

1. Avant de commencer penser
   - à activer dans `emacs` le mode `flyspell-mode`
   - à activer dans `emacs` le mode `auto-fill-mode`
   - à utiliser plusieurs bureaux
   - à mettre votre souris de coté
2. Récupérer le fichier `https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/rapport.tex` (avec `wget`), et le compléter pour essayer de rédiger un document le plus proche possible de la mise en page suivante : `https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/rapport_enseirb_matmeca.pdf`. Terminer par la page de garde. Le logo est diponible ici : `https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/logo_em.pdf`.
   ☞ Ce rapport pourra vous servir comme base pour vos rapports futurs
3. Mettre sous forme d'article "papier" le contenu en ligne suivant : `https://interstices.info/la-machine-a-perspective-de-jan-van-eyck/` [6].
4. Rédigez votre CV en LaTeX

6. sous licence Creative Common

Et pour pratiquer / réviser

Pour pratiquer / s'informer

Informations à propos de l'évaluation

ENSEIRB-Matmeca 2025-2026
1ère année

Examen d'ERI5 -ITEC1 - filière R&I
Responsable : A. Rollet

Introduction Unix, environnement de travail

1h30, aucun document autorisé, toute forme de communication interdite, examen sur machine (session *"exameen"*)

Cette épreuve comporte 4 exercices indépendants.

- Les deux premiers sont composés de questions diverses dont les réponses peuvent nécessiter parfois un peu de texte en français. Il est demandé de répondre dans les fichiers texte `exo1.txt` et `exo2.txt`. Dans ces fichiers, il est demandé de répondre après la ligne `#### Reponse Q`, où `Q` correspond précisément à la question dans le sujet au format `pdf`.
  Exemple :
  ```
  #### Reponse 2.3
  Voici ma réponse à la question 2.3 (de l'exercice 2)
  ```
  Attention à ne pas tout décaler !
- Les deux autres exercices demandent d'écrire du code `bash` à l'intérieur des fichiers `exo3.sh` et `exo4.sh`.

Toutes les réponses contenant des commandes ou des scripts donnant un résultat différent de celui demandé, ou une erreur d'exécution, seront considérées comme fausses. Attention à bien respecter les noms et formats de fichiers, c'est une condition nécessaire pour passer les tests du script de correction.

Si le chemin d'un fichier n'est pas précisé, c'est qu'on considère qu'il est (ou doit être) dans le répertoire courant.

Exercices : script `csv2courrierPdf.sh`

Ecrire un script `csv2courrierPdf.sh` permettant de générer des courriers au format pdf pour des membres du personnel d'une grande école d'ingénieurs les informant de leur affectation de bureau pour la rentrée prochaine.

- Il prend comme paramètre un ficher de données au format csv, avec des lignes au format suivant : `Nom:Prénom:NuméroDeBureau`
- Pour chaque ligne du fichier, il génère un fichier NomPrénom.pdf informant la personne de son nouveau numéro de bureau

Tester votre script avec plusieurs personnes.

Indices :

- utiliser `pdflatex` pour générer le pdf
- utiliser `sed` pour remplacer les mots génériques dans le fichier `.tex` par les valeurs spécifiques à chaque membre

Ajouter un contrôle de validité du fichier csv, et du nombre de fichiers créés.

Exercices : range ton jukebox

`listerMP3.sh`, `rangerMP3.sh`, `corrigerMP3.sh`

Nous avons récupéré à l'adresse
https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/mp3Exo.tar.gz
l'arborescence d'un utilisateur mélomane, mais pas très organisé.

- Récupérer et décompresser l'archive
- Lister (récursivement) tous les fichiers MP3 présents dans le répertoire
- Certains de ces fichier MP3 contiennent des erreurs dans leurs méta-données. Les identifier.
- Créer un répertoire par décennie, et écrire un script qui déplace automatiquement les fichiers MP3 dans le bon répertoire
- (Bonus) Corriger (automatiquement) les fichiers erronés.

Indices :

- Quel outil pourrait récupérer rapidement autant d'informations ?
- Pour extraire les méta-données d'un fichier MP3, essayer :
  `ffprobe -show_entries 'stream_tags : format_tags' TakeDown.mp3 2>/dev/null`
- pour modifier les méta-données d'un ficher MP3 :
  on pourra utiliser `ffmpeg`. Un exemple :
  `ffmpeg -i TakeDown.mp3 -metadata title="Take Down" -metadata artist="HuntrX" -metadata date=2025 "tmp.mp3"`

⚠ Penser à supprimer tous les fichiers MP3 à la fin de l'exercice...

Exercices : aide à la correction d'examen machine

Dans le fichier `exam.tgz` fourni ici `https://rollet.zzz.bordeaux-inp.fr/IntroUnix/ressources/exam.tgz`, vous trouverez les éléments obtenus après un examen sur machine, à savoir : le sujet de l'examen ainsi que la copie de chaque étudiant (le nom du répertoire correspond au login de l'étudiant). Rédigez des scripts pour aider l'enseignant à automatiser la correction et la notation des travaux :

- Écrire un script permettant de générer automatiquement un fichier de réponses vide, comme `answers.txt`, avec un nombre arbitraire de questions.
- Écrire un script permettant d'évaluer les réponses de tous les étudiants pour une question spécifique. Ce script prend en argument le numéro de la question. L'enseignant doit inscrire les points en début de ligne sous forme d'entier avant les #### pour chaque question. Les notes sont soit 0, soit 1. Il est possible d'utiliser `nano` comme éditeur de texte.
- Écrire un script permettant d'obtenir un fichier CSV résumant les notes de chaque étudiant pour chaque question.
- Transformer le script précédent de façon à ajouter un champ dans le fichier CSV contenant la somme des points (fournissant ainsi une note sur 10).
- Écrire un script permettant d'évaluer automatiquement les étudiants
  - Indice 1 : pourquoi ne pas rediriger la sortie standard dans un fichier ?
  - Indice 2 : pourquoi ne pas concaténer la réponse de l'étudiant dans un fichier de test approprié ?

En complément

Les plus rapides / motivés pourront aller piocher d'autres exercices au lien suivant :

`https://cours-mf.gitlabpages.inria.fr/if104/`