

Tools_Shell

Outils pour l'informatique : Système d'exploitation **Unix**

Utiliser sa puissance

Antoine Cornuéjols – Christine Martin

AgroParisTech – **INRAe** MIA Paris-Saclay

EKINOCS research group

Objectifs du cours

1. Savoir utiliser l'environnement Unix
2. Avoir un 1^{er} contact avec les expressions régulières
3. Notions de script `bash` et d'utilisation de `Python` pour le maniement des fichiers
4. Savoir installer un logiciel

Plan

1. L'environnement Unix
2. Travailler avec des motifs : les expressions régulières
3. Scripts Bash et programmes Python
4. L'installation de logiciels

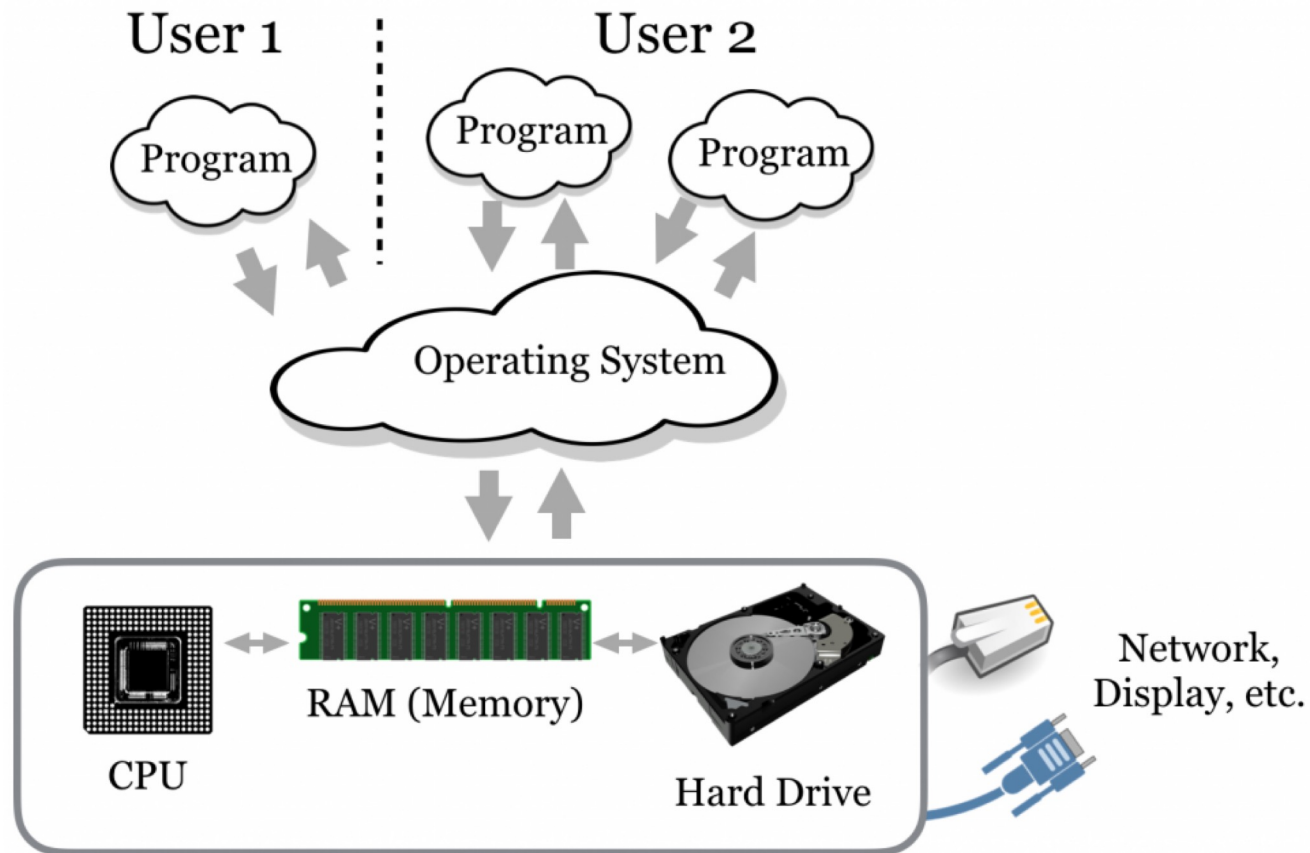
Rôles d'un système d'exploitation

- Un **ordinateur** est constitué de composants électroniques
 - Disque
 - Mémoire RAM
 - Processeur(s)
- Le **système d'exploitation** (*Operating System* ou *OS*)

Rôles d'un système d'exploitation

- Un **ordinateur** est constitué de composants électroniques
 - Disque
 - Mémoire RAM
 - Processeur(s)
- Le **système d'exploitation** (*Operating System* ou *OS*)
 - Organise le **disque** en une arborescence de fichiers
 - **Lance** un ou plusieurs **programmes** / applications pouvant marcher simultanément sur le(s) processeur(s)
 - Fournit de l'**espace mémoire** à ces programmes
 - Fait **communiquer** le(s) processeur(s) et les périphériques : clavier, touchpad, souris, écran, et.

Système d'exploitation



From [Shawn T. O'Neil (2019) « A primer for Computational Biology », Oregon State University]

Système d'exploitation

- Qu'est-ce qu'un **système d'exploitation** (Operating System (OS))?
 - Tout le fonctionnement d'un ordinateur est **contrôlé** par un système d'exploitation (*Operating System* ou OS)
 - **Exemple** : lorsque plusieurs programmes tournent ensemble, c'est l'OS qui alloue les ressources entre eux et les interrompt si nécessaire.
 - **Unix** est un exemple d'OS, très répandu (dont toutes les machines de calcul scientifique)
 - Développé dans les années 70 (après Multics)
 - **Mac OS X** et **Ubuntu** (utilisant Linux), par exemple, sont basés sur Unix (mais Windows ne l'est pas)

Exemples de systèmes d'exploitation

- Linux
 - Ubuntu
 - Redhat
 - Suze
 - Debian
 - ...
- Unix
 - Dont MacOS
- Windows
- Smartphones
 - iOS
 - androïd

Système d'exploitation

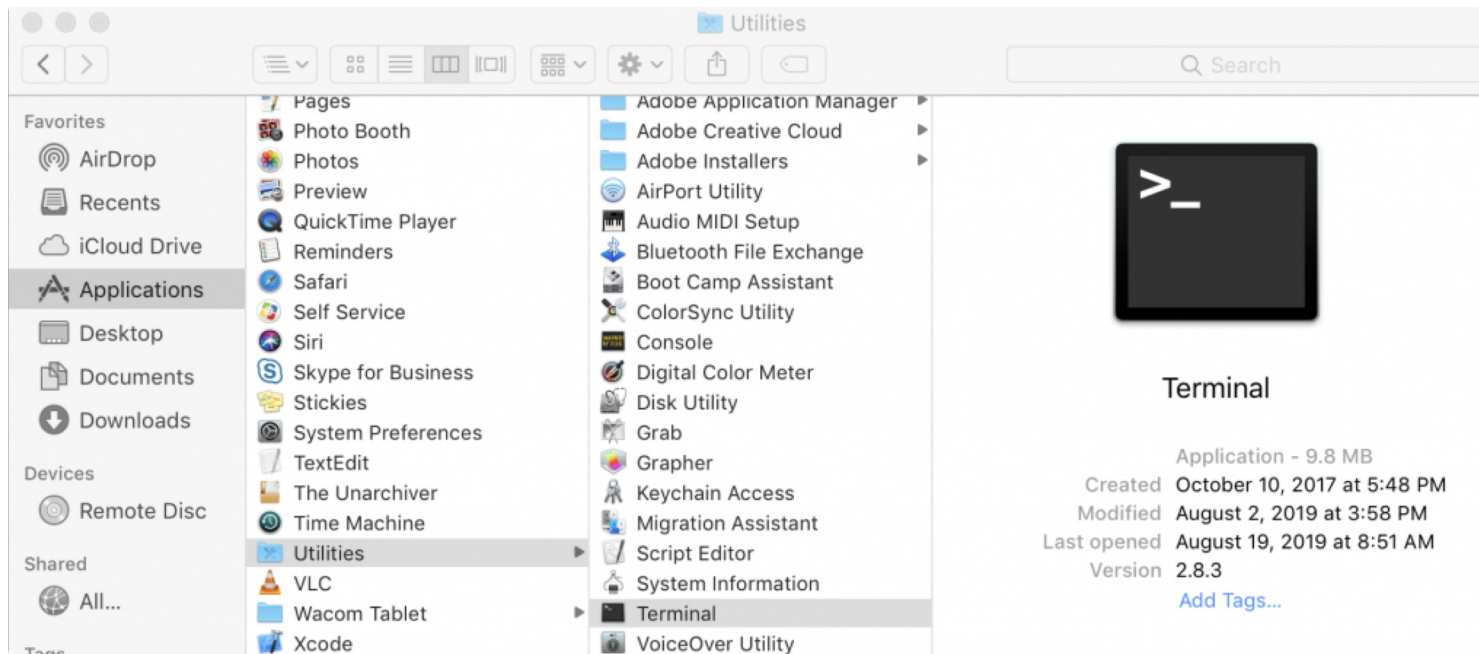
- En général, les OS ont **deux types d'interface**
 - **Graphique** (GUI : Graphical User Interface)
 - En **ligne de commande** (CLI : Command Line Interface)

Pourquoi les lignes de commandes

- Une interface **GUI** permet
 - de *naviguer* facilement parmi les fichiers,
 - *copier, déplacer et supprimer* des fichiers et des répertoires
- Une interface **CLI**
 - Permet tout cela
 - Mais aussi des tâches beaucoup **plus complexes**
 - E.g. **trouver**, dans votre arborescence, tous les fichiers *dont le nom se termine par ".txt", qui datent de plus de 3 jours et dont le deuxième mot de la troisième ligne est "pixels"* et, dans tous ces fichiers, il vous permet de **remplacer** ce mot par un autre mot.

Accès à un système Unix

- **Mac OS X** ou **Linux** : vous avez un accès « direct »
 - **Sous Linux** : `ctrl + alt + T`
 - **Sous Mac OS X**
 - Icône **Terminal** dans le Launchpad
 - ou :



Unix sous Windows

Soit la solution CygWin

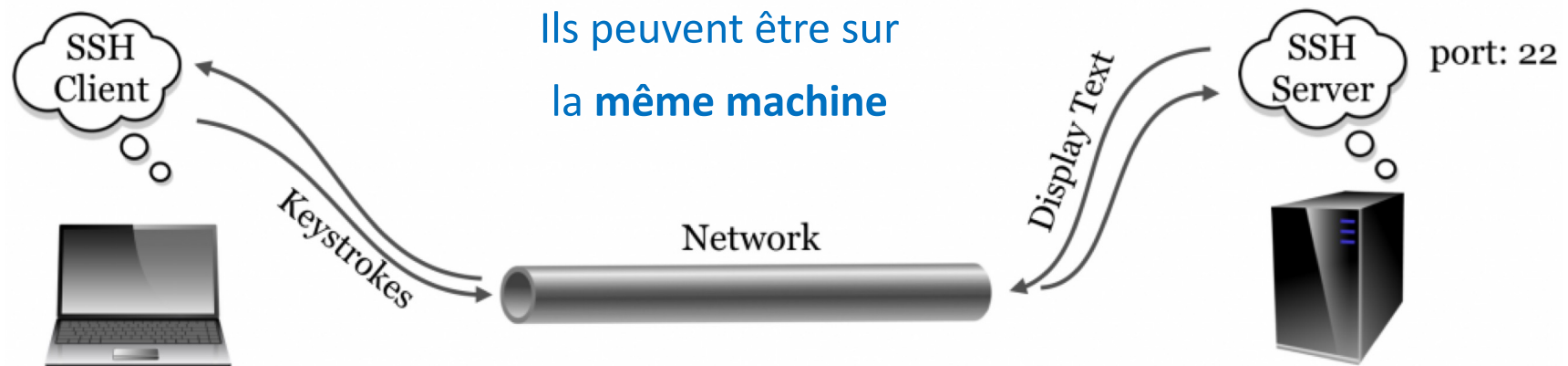
Soit la solution WSL

- **Cygwin** est essentiellement un ensemble de bibliothèques Windows qui permettent la compilation de sources *nix vers des binaires Windows.
- **WSL** est un outil officiel de Windows. Il permet de faire tourner un vrai système Linux à l'intérieur de Windows.
 - Vous obtenez un **terminal complet** avec un **noyau Linux** sans ralentir votre PC avec une machine virtuelle lourde.
 - Vous pouvez exécuter des **applications graphiques** sur WSL2, y compris un environnement de bureau complet avec une accélération **GPU** complète.

Accès à un système Unix sous **Windows**

- Accès à une **machine virtuelle** Unix ou Linux
- Utilisation de **WSL** (Windows Sous Linux)
 - Le Sous-système Windows pour Linux permet aux développeurs d'exécuter un environnement GNU/Linux (et notamment la plupart des utilitaires, applications et outils en ligne de commande) **directement sur Windows**, sans modification et tout en évitant la surcharge d'une machine virtuelle traditionnelle ou d'une configuration à double démarrage.
 - <https://learn.microsoft.com/fr-fr/windows/wsl/>

ssh

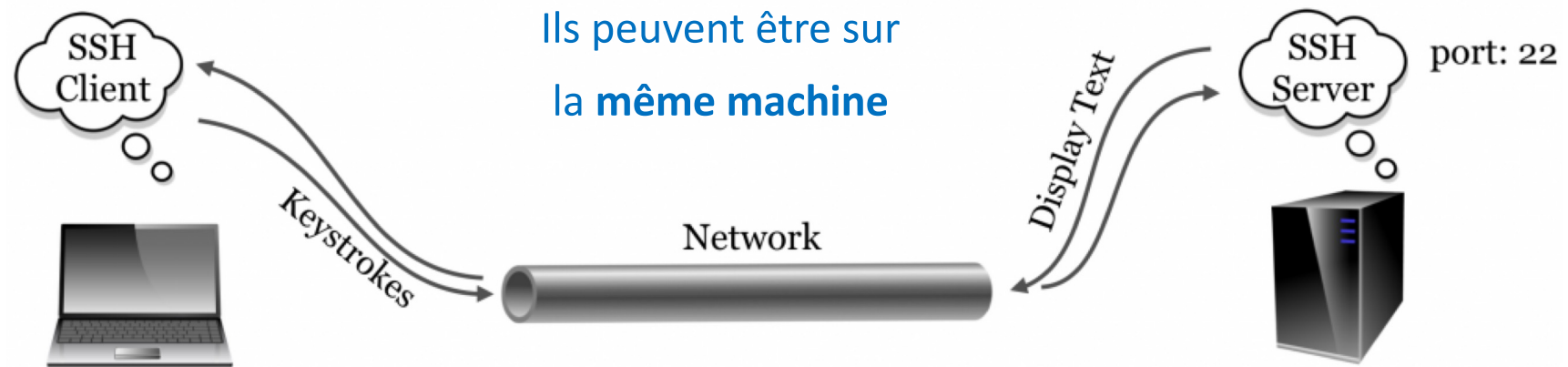


Client

Serveur

- Le **protocole SSH** (Secure Shell) est un **protocole de communication** réseau qui permet de se connecter et d'administrer un serveur distant en toute sécurité
- **Utilisations courantes :**
 - Prise de **contrôle d'un terminal distant** (shell) par ligne de commande pour configurer un serveur
 - **Transfert de fichiers** de manière sécurisée via des protocoles dérivés tels que **SFTP** ou SCP.

ssh



Client

Serveur

- Le **serveur** ssh et le **client** ssh communiquent par le **protocole ssh**
 - **Sécurisé** (par cryptographie à clé publique avancée)
 - Le **client** envoie des *commandes*
 - Le **serveur** envoie du *texte*
 - Par convention, SSH se connecte sur le **port 22** (et HTTP sur le port 80)

Le shell

- Notion de **shell**
 - **Interface** permettant d'interagir avec un système d'exploitation
 - Un OS est composé d'un **noyau** (kernel) et d'une **coque** (shell)
 - le **shell** est un programme qui **reçoit des commandes** qu'on va écrire depuis le clavier (ou par une interface graphique + souris) et qui **les passe au système d'exploitation** afin qu'elles soient exécutées.
- Le **terminal**
 - Se borne à **permettre l'interaction avec le shell** en lui transmettant les **entrées** clavier de l'utilisateur et en affichant ses **réponses**

Bash

- **Shell : Interface** permettant d'interagir avec un système d'exploitation
 - nous allons devoir **envoyer nos commandes** sous un certain **format** compréhensible par notre OS. Ce « format » est en fait **un langage de programmation** spécifique au shell utilisé.
 - Le plus utilisé est **Bash** (*"Bourne Again SHell"*). 
 - Bash est notamment le shell utilisé par défaut par les systèmes OS X (en fait zsh depuis Catalina)
 - Il existe aussi **sh** (Bourne shell) ; **cs**h (C shell) ; **ksh** (Korn shell) ; ...
- **Terminal**
 - Le **Terminal** est l'interface de ligne de commande de **Mac**.
 - Nous allons utiliser le Terminal pour envoyer nos commandes à notre OS et pour communiquer avec lui.
 - L'invite de commande ou "command prompt" ou CMD est l'interpréteur de ligne de commande **Windows**.

Bash

- Ouverture de **Terminal**



- Lorsqu'on ouvre le Terminal, une **fenêtre** noire ou blanche s'ouvre. Cette fenêtre contient un **invite de commande**, c'est-à-dire une ligne nous indiquant que le shell attend qu'on lui passe des commandes.
- Par défaut, cette ligne contient le **nom de votre machine** suivi de **deux points** suivi du caractère **tilde** (~) suivi de votre **nom d'utilisateur** suivi du **signe** dollar (\$) (ou % pour Zsh)).
- Le **tilde** est une abréviation pour indiquer qu'on se situe actuellement **dans le dossier "home"**, c'est-à-dire dans le répertoire de base lié à notre nom d'utilisateur.
- Le signe **dollar** (ou %) indique qu'on est **connecté en tant qu'utilisateurs classiques** avec des privilèges normaux.

Que voudrait-on que le système d'exploitation (OS)
permette comme opérations ?

Les commandes Bash

et le système de fichiers

Premiers pas et variables d'environnement

```
oneils@atmosphere ~$ echo hello there  
hello there
```

```
oneils@atmosphere ~$ echo $USER  
oneils
```

```
oneils@atmosphere ~$ echo $0  
-bash
```

```
oneils@atmosphere ~$ tcsh  
172:~> echo $0  
tcsh
```

Changer pour le C shell tcsh

```
172:~> exit  
exit  
oneils@atmosphere ~$ echo $0  
-bash
```

...

Arborescence de fichiers

Se diriger dans le système de fichiers

- Quand vous vous loggez pour la 1^{ère} fois, vous arrivez dans votre « home directory »
- **PWD** : Present Working Directory

```
oneils@atmosphere ~$ echo $HOME
/home/oneils
oneils@atmosphere ~$ echo $PWD
/home/oneils
oneils@atmosphere ~$ pwd
/home/oneils
```

Se diriger dans le système de fichiers

- Vous pouvez avoir la **liste des fichiers** de votre PWD par la commande **ls**

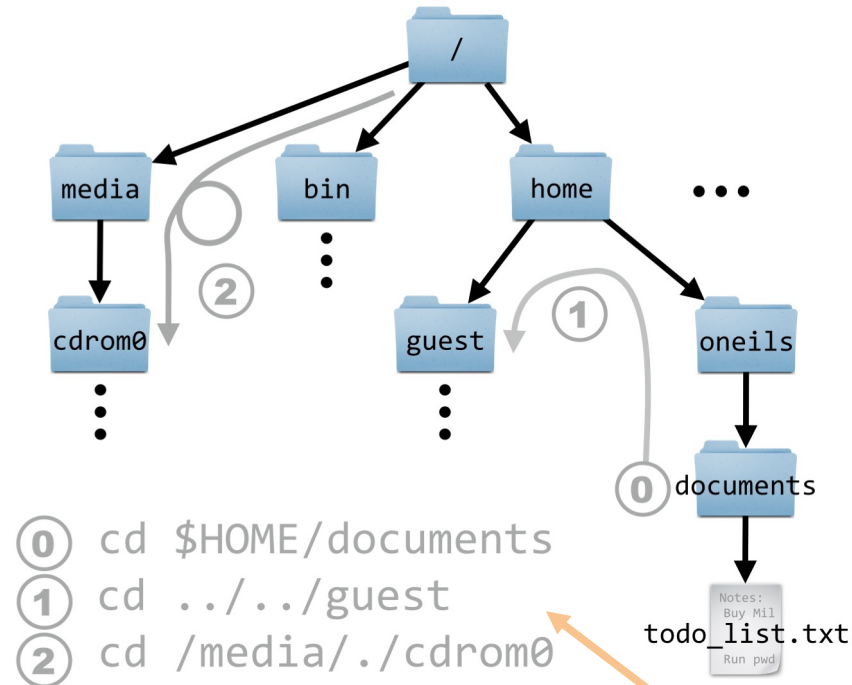
```
oneils@atmosphere ~$ ls
apcb      Documents Music      Public      todo_list.txt
Desktop   Downloads Pictures   Templates   Videos
```

- On peut **changer de répertoire** (directory) par la commande **cd** (change directory)

```
oneils@atmosphere ~$ cd /home
oneils@atmosphere /home$ echo $PWD
/home
oneils@atmosphere /home$ ls
lost+found oneils
```

- On peut **revenir** à son « home directory » par :
 - `cd $HOME`
 - `cd`
 - `cd ~`

Se déplacer dans la hiérarchie des répertoires



\$HOME vaut /home/oneils

Chemin **relatif** partant du répertoire courant

Chemin **absolu** partant de la racine (root)

Chemins **absolus** et chemins **relatifs**

Chemin absolu : `/home/oneils/Pictures/profile.jpg`

Chemin absolu : `/home/oneils/todo_list.txt`

- Supposons être **dans l'home directory**
 - Chemin relatif : `oneils/Pictures/profile.jpg`
 - Chemin relatif : `oneils/todo_list.txt`
- Supposons être **dans home/oneils**
 - Chemin relatif : `Pictures/profile.jpg`
 - Chemin relatif : `todo_list.txt`

Remarque :

- Les chemins **absolus** commencent par `/` (car à partir de la racine `/`)
- Pas les chemins **relatifs**

Chemins **absolus** et chemins **relatifs**

Supposez que votre répertoire ait
la structure suivante :

```
/
└─ home/
    └─ alice/
        ├── notes.txt
        └─ projects/
            └─ code.py
```

- **Absolute path** to *notes.txt*:
 - `/home/alice/notes.txt`
- **Relative path** (if you are in `/home/alice`):
 - `notes.txt`
- **Relative path** (if you are in `/home/alice/projects`):
 - `../notes.txt`

Chemins **absolus** et chemins **relatifs**

Règles à suivre :

- Utiliser les chemins **absolus** dans l'**écriture de scripts**
(pour qu'ils fonctionnent quelque soit le répertoire où ils sont placés)
- Utiliser les chemins **relatifs** pour une **navigation rapide**
quand vous savez où vous vous trouvez dans la hiérarchie des dossiers

Remarques

- Dans les systèmes Unix, les commandes sont **sensibles à la casse**
- Les espaces comptent
 - On donnera donc des **noms** de fichiers et de répertoires **sans espaces**
 - Eg. `todo_list.txt`
 - Sinon il faut utiliser `'todo list.txt'` ou `todo\list.txt` dans les commandes (à **éviter !!!**)

Les fichiers cachés

- Par défaut les fichiers commençant par `.` sont « cachés »
- Il faut utiliser l'option `-a` pour les voir (`ls -a`)

```
oneils@atmosphere ~/Pictures$ cd $HOME
oneils@atmosphere ~$ ls -a
.          .config    .gvfs       .profile    .vim
..         .dbus      .gvfs       Public      .viminfo
apcb       Desktop   .ICEauthority .pulse      .vimrc
.bash_history Documents .local      .pulse-cookie .vnc
.bash_login  Downloads Music       .ssh        .Xauthority
.bash_logout .gconf    .netrc     Templates   .Xdefaults
.bashrc      .gnome2   Pictures   todo_list.txt .xscreensaver
.cache       .gnupg    .pip       Videos     .xsession-errors
```

- Ce sont en général des **fichiers de configuration** utilisés par des programmes variés

Commande **ls** et ses **options**

<code>ls</code>	affiche la liste des fichiers et sous-répertoires du répertoire courant
<code>ls rep1/toto</code>	affiche la liste des fichiers et sous-répertoires du répertoire rep1/toto
<code>ls -l</code>	affiche une liste détaillée (droit, propriétaire, taille, etc.)
<code>ls -a</code>	affiche aussi les fichiers cachés
<code>ls -t</code>	affiche par ordre de date de dernière modification
<code>ls -F</code>	Distingue les fichiers des sous-répertoires

- Option **-h** pour « human readable »
 - Donne les tailles mémoire en Ko ou Mo

```
oneils@atmosphere ~$ ls -lah
total 168K
drwxr-xr-x 25 oneils iplant-everyone 4.0K Sep 23 22:40 .
drwxr-xr-x  4 root    root           4.0K Sep 15 09:48 ..
drwxr-xr-x  4 oneils iplant-everyone 4.0K Sep 15 11:19 apcb
-rw-----  1 root    root           2.2K Sep 15 10:49 .bash_history
-rw-r--r--  1 oneils iplant-everyone   61 Sep 16 19:46 .bash_login
-rw-r--r--  1 oneils iplant-everyone  220 Apr  3  2012 .bash_logout
-rw-r--r--  1 oneils iplant-everyone  3.6K Sep 15 09:48 .bashrc
drwx-----  7 oneils iplant-everyone 4.0K Sep 15 09:52 .cache
...
```

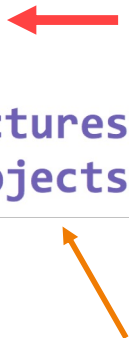
- réfère au répertoire courant
- réfère au répertoire home
 - E.g. **cd ..** remonte au répertoire home

```
oneils@atmosphere ~$ echo $PWD
/home/oneils
oneils@atmosphere ~$ cd .
oneils@atmosphere ~$ echo $PWD
/home/oneils
oneils@atmosphere ~$ cd ..
oneils@atmosphere /home$ echo $PWD
/home
```

Créer de nouveaux répertoires

mkdir (make directory)

```
oneils@atmosphere ~$ ls
apcb      Documents Music      Pictures  Templates Videos
Desktop  Downloads p450s.fasta Public    todo_list.txt
oneils@atmosphere ~$ mkdir projects ←
oneils@atmosphere ~$ ls
apcb      Documents Music      Pictures  Public    todo_list.txt
Desktop  Downloads p450s.fasta projects  Templates Videos
```



Copier ou renommer un fichier ou un répertoire

mv (move)

```
oneils@atmosphere ~$ ls
apcb      Documents Music      Pictures  Public    todo_list.txt
Desktop   Downloads p450s.fasta projects  Templates Videos
```



```
oneils@atmosphere ~$ mv p450s.fasta p450s.fa
oneils@atmosphere ~$ mv p450s.fa projects
oneils@atmosphere ~$ mv projects projects_dir
oneils@atmosphere ~$ ls
apcb      Documents Music      projects_dir Templates  Videos
Desktop   Downloads Pictures    Public      todo_list.txt
oneils@atmosphere ~$
```

- Si la **destination** n'existe pas, elle est créée
- Si la **destination** existe
 - Si c'est un répertoire, la **source est déplacée** dans celui-ci
 - Si c'est un fichier, celui-ci est **écrasé (!!!)** et le contenu remplacé par celui de la source

Copier un fichier ou un répertoire

cp (copy)

- Avec l'option **-r** (récuratif) si l'on veut copier tout le contenu d'un répertoire avec ses sous-répertoires et ses fichiers

```
oneils@atmosphere ~$ cp todo_list.txt todo_copy.txt  
oneils@atmosphere ~$ cp -r projects projects_dir_copy
```

- Option **-i** : demande à l'utilisateur confirmation (e.g. si va écraser un fichier)
- Option **-n** : n'écrasera pas un fichier existant

Retirer ou effacer un fichier ou un répertoire

rm (remove)

```
oneils@atmosphere ~/projects$ mkdir tempdir
oneils@atmosphere ~/projects$ ls
p450s.fasta  tempdir  todo_list.txt
oneils@atmosphere ~/projects$ rm todo_list.txt
oneils@atmosphere ~/projects$ rm -rf tempdir/
oneils@atmosphere ~/projects$ ls
p450s.fasta
```

- **ATTENTION !!!**
 - Les fichiers ou répertoires effacés **ne peuvent pas être récupérés (!!!)**
 - Pas de Ctrl z
 - Pas de « corbeille »

Caractères **spéciaux** utiles

- **?** : remplace **un caractère** quelconque
 - `mv ../data/out0?.dat ~/poub/`
 - déplace tous les fichiers `.dat` de nom commençant par `out0` **et un caractère** du répertoire `../data` dans le répertoire `poub` du répertoire personnel (indiqué par le caractère spécial `~`) (e.g. `home/dupont`)
- ***** : remplace une **chaîne de caractères** quelconque
 - `rm rep1/*.dat` : détruit tous les fichiers du répertoire `rep1` dont le nom fini par `.dat`

Obtenir de l'information sur les utilisateurs

finger

```
[(base) MacBook-Pro-de-ANTOINE-5:Man_Unix antoinecornuejols$ finger antoine
Login: antoinecornuejols                Name: ANTOINE CORNUEJOLS
Directory: /Users/antoinecornuejols      Shell: /bin/zsh
On since Mer 23 août 17:41 (CEST) on console, idle 5 days 1:44 (messages off)
On since Mer 23 août 18:19 (CEST) on ttys000, idle 5 days 0:39
On since Mer 23 août 18:47 (CEST) on ttys001, idle 4 days 1:35
On since Jeu 24 août 17:51 (CEST) on ttys002
On since Mer 23 août 18:02 (CEST) on ttys003 (messages off)
No Mail.
No Plan.
(base) MacBook-Pro-de-ANTOINE-5:Man_Unix antoinecornuejols$
```

En général, pas installé par défaut. Il faut charger la commande.

Droits d'accès aux fichiers et répertoires

Les permissions

- Tous les fichiers et répertoires sont associés à un **utilisateur** (le propriétaire) et un **groupe**, plus « **les autres** ».
- Les **permissions** déterminent ce que peuvent faire l'utilisateur, le **groupe** et les **autres**.
- Elles sont décrites par une **combinaison de permissions** de :
 - Lecture (**r** : read)
 - Écriture (**w** : write)
 - Exécution (**x** : execute)

Il existe plusieurs façons de **connaître les groupes** auxquels un utilisateur appartient.

Le **groupe de l'utilisateur principal** est stocké dans le **fichier /etc/passwd** et les **groupes supplémentaires**, le cas échéant, sont répertoriés dans le **fichier /etc/group**.

Les permissions

- Exemple :

```
oneils@atmosphere ~/apcb/intro$ ls -l
total 20
-rwxrwxrwx 1 oneils iplant-everyone 15891 Oct 20 17:42 p450s.fasta
drwxr-xr-x 2 oneils iplant-everyone  4096 Oct 20 17:40 temp
```

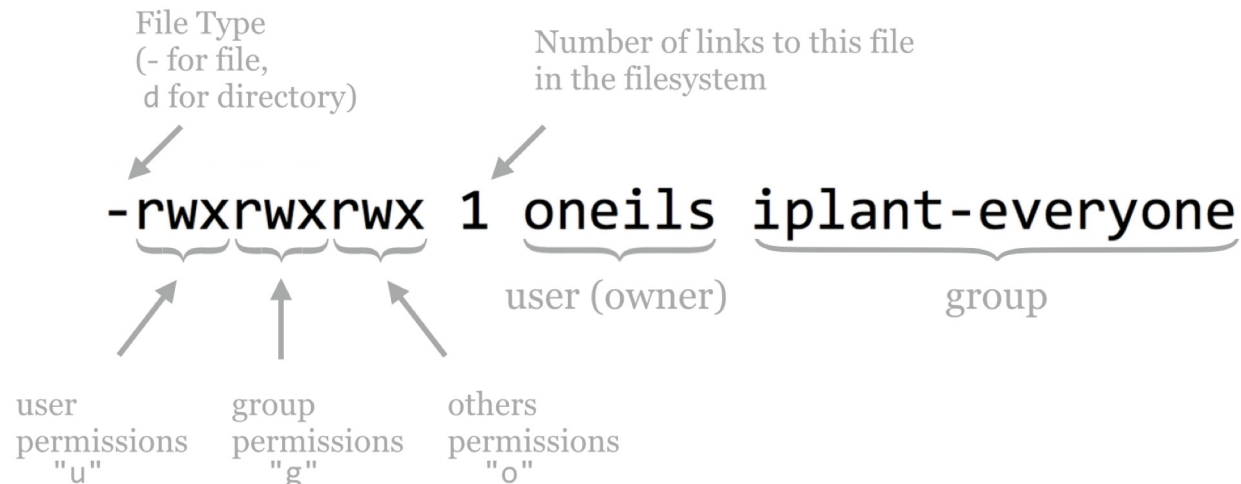
Propriétaire

Groupe

Tout le monde

- Pour le 1er fichier : la combinaison **rwxrwxrwx** permet à tout le monde de tout faire
- Pour le répertoire :
 - Est accessible par **tout le monde** : **x**
 - Peut être lu par les membres du **groupe** : **rx**
 - Et modifié seulement par le **propriétaire** : **rwX**

Les permissions

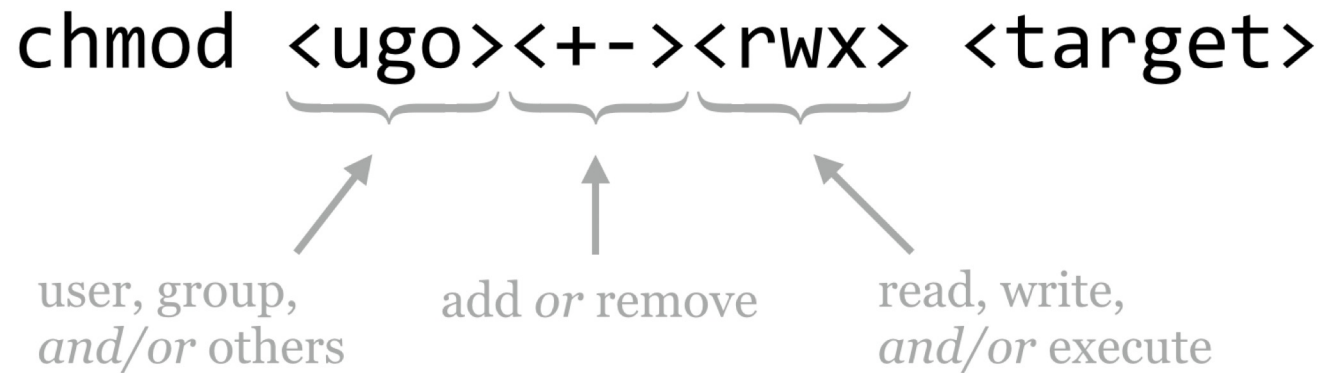


Code	Meaning for Files
r	Can read file contents
w	Can write to (edit) the file
x	Can (potentially) "execute" the file

Code	Meaning for Directories
r	Can see contents of the directory (e.g., run <code>ls</code>)
w	Can modify contents of the directory (create or remove files/directories)
x	Can <code>cd</code> to the directory, and potentially access subdirectories

...

Les modifications de permissions



Exemples :

Command	Effect
<code>chmod go-w p450s.fasta</code>	Remove write for group and others
<code>chmod ugo+r p450s.fasta</code>	Add read for user, group, and others
<code>chmod go-rwx p450s.fasta</code>	Remove read, write, and execute for group and others
<code>chmod ugo+x p450s.fasta</code>	Add execute for user, group, and others
<code>chmod +x p450s.fasta</code>	Same as <code>chmod ugo+x p450s.fasta</code>

Commandes Bash

- **pwd** **print working directory**
 - Renvoie le répertoire du chemin courant
- **mkdir** **make dir**
 - Permet de créer de nouveaux répertoires
- **chmod**
 - Change les permissions d'accès aux fichiers
- **ls** **list files and directories**
- **cd** **change directory**
- **file** **retourne le type de fichier**
- **man**
 - Fournit une description de la commande fournie en argument

Commandes Bash de manipulation de fichiers

- **cp** **copy file**
- **mv** **move**
 - déplace ou renomme des fichiers
- **rm** **remove**
 - Supprime ou renomme des fichiers ou des répertoires

Exercices

1. Dans votre home, tapez `pwd`. Qu'est-ce qui est retourné ?
2. Créez un répertoire `Man_Shell` dans `home` et déplacez vous dedans
3. Tapez `pwd`. Qu'est-ce qui est retourné ?
4. Créez un fichier `test.txt` dans `home/Man_Shell`
5. Grâce à un éditeur de texte, tapez quelques lignes dans ce fichier
6. Quelles sont les permissions par défaut données au fichier ?
7. Modifiez ces permissions pour le rendre lisible par le groupe
8. Copiez ce fichier dans un fichier `test2.txt` dans `home/Man_Shell`

Commandes Bash de **recherche de fichiers**

- **locate** <nom> recherche les fichiers de nom <nom> dans tous les répertoires (avec n'importe quelle extension)

- **Attention** : si <nom> est très courant, énormément de réponses
(e.g. **locate bin** renvoie des 10^4 lignes)

On peut utiliser un pipe (voir plus loin)

E.g. **locate bin | grep pdf** renvoie les fichiers bin dont le nom contient pdf

- Nécessite d'avoir au préalable créé la table locate

Les processus

- Processeur

- Un **processeur** est une composante électronique capable d'exécuter des instructions de calcul très rapidement. Pendant longtemps, chaque ordinateur contenait un seul processeur.

- Programme

- Un **programme**, comme ceux que vous écrivez en python, sont des fichiers composés d'instructions que le processeur peut exécuter.

- Processus

- Un *programme* lancé (par l'utilisateur ou par d'autres programmes) et chargé en mémoire est appelé un **processus**.
- Un même programme peut être à l'origine d'un ou plusieurs processus.
 - E.g. lancer plusieurs éditeurs gedit

- Process Identifier ou **pid**

- Chaque processus lancé sur une machine a un **identifiant** unique sous la forme d'un entier, et qui est appelé le *process identifier* ou, plus communément le **pid**

- User Identifier ou **uid**

- Chaque processus a un **propriétaire** qui est, presque toujours, l'utilisateur qui a lancé ce processus. Les droits qu'a un processus de lire/écrire/exécuter un fichier ou un répertoire sont ceux de son propriétaire
- Le propriétaire est identifié par un **uid**

Lister les processus

- Les commandes **top** et **ps** permettent de lister les processus ouverts sur une machine, en fournissant les informations :

PID	son identificateur
PPID	l'identificateur de son processus parent
UID	l'identificateur de son propriétaire
S	son statut (R pour <i>running</i> , S pour <i>sleeping</i> , Z pour <i>Zombi</i>)
N	son nice
PR	sa priorité effective
%CPU	le pourcentage du temps de processeur consommé
%MEM	le pourcentage de la mémoire consommée
SZ	la taille mémoire allouée
TTY	le terminal depuis lequel il a été lancé
TIME	le temps processeur consommé
CMD	la commande qui a lancé le processus

Chercher les processus en cours

- Voir quels sont les **programmes qui tournent** : **top**
 - Fournit le pourcentage de CPU consommé, combien de mémoire, les utilisateurs, ...
 - **Mis à jour** en temps réel. Liste pouvant être très longue. On quitte par **q**

```
top - 02:23:20 up 15 days, 16:34, 2 users, load average: 0.03, 0.02, 0.05
Tasks: 127 total, 1 running, 126 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni, 99.7%id, 0.3%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 4050124k total, 1801608k used, 2248516k free, 142652k buffers
Swap: 0k total, 0k used, 0k free, 1256080k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	90540	4264	2696	S	0.0	0.1	0:07.85	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:43.50	ksoftirqd/0
5	root	20	0	0	0	0	S	0.0	0.0	0:00.35	kworker/u:0
6	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0

- Voir quels sont les programmes qui tournent et qui **vous appartiennent** : **ps**
 - **Pas mis à jour**

```
(base) mbpdeantoine5:myproject antoinecornuejols$ ps
  PID TTY          TIME CMD
 24351 ttys000    0:00.15 -bash
```

Lister les processus par la commande **top**

- La commande **top** permet de connaître la liste des processus qui tournent sur une machine en temps-réel.
 - la touche **i** permet d'afficher seulement les processus en cours d'exécution (qui consomment du temps processeur)
 - la touche **espace** rafraîchit l'affichage à la demande
 - la touche **q** permet de quitter top

```
top - 09:24:08 up 2 min, 1 user, load average: 1,43, 0,80, 0,32
Tâches: 261 total, 2 en cours, 181 en veille, 0 arrêté, 0 zombie
%Cpu(s): 6,2 ut, 4,0 sy, 0,0 ni, 78,7 id, 10,5 wa, 0,0 hi, 0,7 si, 0,0 st
KiB Mem : 8049736 total, 4478848 libr, 1016244 util, 2554644 tamp/cache
KiB Éch: 0 total, 0 libr, 0 util. 6334448 dispo Mem
```

PID	UTIL.	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TEMPS+	COM.
13357	ahabibi	20	0	3033752	311384	152980	S	13,9	3,9	0:10.45	firefox
2379	ahabibi	20	0	482032	85340	58648	S	10,3	1,1	0:05.69	Xorg
2888	ahabibi	20	0	3068912	228504	144464	S	9,6	2,8	0:09.09	gnome-shell
13570	ahabibi	20	0	2668424	210468	150436	S	4,3	2,6	0:03.07	Web Content
3800	ahabibi	20	0	658528	56040	34652	S	2,0	0,7	0:01.01	x-terminal-emul
3081	ahabibi	20	0	896448	49364	36252	S	1,7	0,6	0:01.51	nautilus
36	root	20	0	0	0	0	I	1,3	0,0	0:00.62	kworker/1:1
53	root	20	0	0	0	0	I	1,0	0,0	0:00.36	kworker/u8:1
132	root	20	0	0	0	0	I	1,0	0,0	0:00.39	kworker/u8:2
184	root	20	0	0	0	0	I	1,0	0,0	0:00.44	kworker/u8:3
577	root	0	-20	0	0	0	I	1,0	0,0	0:00.40	kworker/1:1H
448	root	20	0	0	0	0	I	0,7	0,0	0:00.28	kworker/0:3
3911	root	20	0	0	0	0	R	0,7	0,0	0:00.20	kworker/2:9
14648	ahabibi	20	0	54572	4596	3780	R	0,7	0,1	0:00.32	top

Lister les processus par la commande **ps**

- La commande **ps** permet de connaître la liste des processus qui tournent sur une machine et d'en **envoyer le résultat à d'autres commandes**
 - **ps** : sans option affiche la liste des processus lancés **depuis le shell courant**
 - **ps -u dupont** : affiche la liste des processus lancés par **l'utilisateur dupont**
 - **ps -e** : affiche la liste de **tous les processus** existants sur la machine
 - **ps -l** : comme **ps** sans option mais avec plus d'informations

PID	TTY	TIME	CMD
3874	pts/1	00:00:00	bash
12063	pts/1	00:00:00	gedit
21445	pts/1	00:00:01	gimp
21453	pts/1	00:00:00	script-fu
21815	pts/1	00:00:00	ps

Avec **ps** sans option

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
0	S	127257	3874	3800	0	80	0	-	9598	wait	pts/1	00:00:00	bash
0	S	127257	12063	3874	0	80	0	-	162803	poll_s	pts/1	00:00:00	gedit
0	S	127257	21445	3874	1	80	0	-	233831	poll_s	pts/1	00:00:01	gimp
0	S	127257	21453	21445	0	80	0	-	74891	poll_s	pts/1	00:00:00	script-fu
4	R	127257	22367	3874	0	80	0	-	7234	-	pts/1	00:00:00	ps

Avec **ps -l**

« tuer » un processus par la commande **kill**

- La commande **kill** permet d'interrompre un processus
 - `kill -numero pid :`
 - Si **numero = 2** (e.g. `kill -2 60533`) interrompt le processus 60533.
Équivalent à Ctrl + C
 - Si **numero = 9** (e.g. `kill -9 60533`) interrompt le processus 60533.
Sans possibilité de rattrapage ou d'être ignoré

Les commandes Bash

et le contenu des fichiers

Comparer des fichiers grâce à **diff**

1. Créez fichier_1.txt avec dedans le texte « Ceci est une chaîne de caractères »
2. Copiez fichier_1.txt dans fichier_2.txt
 - Que donne `$ diff fichier_1.txt fichier_2.txt ?`
 - Et `$ diff -s fichier_1.txt fichier_2.txt ?`
3. Modifiez le contenu de fichier_2.txt en remplaçant des **minuscules** par des **majuscules**
 - Que donne `$ diff fichier_1.txt fichier_2.txt ?`
 - Que donne `$ diff fichier_1.txt fichier_2.txt -i ?`
 - Pourquoi ?

Comparer des fichiers grâce à **diff**

- Modifiez le contenu de fichier_1.txt et de fichier_2.txt en ajoutant respectivement « ligne 1 » et « ligne 2 »
- Que donne maintenant `$ diff fichier_1.txt fichier_2.txt` ?

Pour en **savoir davantage** utilisez `diff --help`

Commande cat

- de l'anglais **catenate**, synonyme de **concatenate** (concaténer), est une commande Unix standard permettant de
 - **concaténer** des fichiers
 - ainsi que **d'afficher leur contenu** sur la sortie standard
- `cat fichier.txt`
 - Affiche le contenu de fichier.txt
- `cat fichier.txt | more`
 - Affiche la 1^{ère} page de fichier.txt et chaque page suivante si on fait return (sortie par 'q' (quit))
- `cat -n fichier.txt`
 - Affiche le contenu de fichier.txt en numérotant les lignes (voir `cat -b` qui ne numérote que les lignes non vides)
- `cat fichier1.txt fichier2.txt`
 - Affiche le contenu des deux fichiers
- `cat source1.txt source2.txt > destination.txt`
 - Met le contenu des fichiers source1.txt et source2.txt dans le fichier destination.txt
 - Si `>>` au lieu de `>` : concatène le contenu de source2.txt et de source1.txt à la fin de destination.txt
- `cat -v nomdufichier.txt`
 - Affiche les caractères non imprimables du fichier

Pipe (se prononce païpe)

Redirection

```
$ cat > nouveaufichier.txt  
This is a new file
```

```
^D
```

```
$ cat >> nouveaufichier.txt  
With a new line
```

```
^D
```

```
$ cat -n nouveaufichier.txt
```

```
$ cat nouveaufichier.txt > newfile.txt
```

Commandes Bash

wc (word count) pour compter

- **wc -c** : compte le nombre de **caractères** du fichier
- **wc -w** : compte les **mots** du fichier
- **wc -l** : compte le nombre de **lignes** (en fait les newlines) du fichier

```
$ wc -l programming.txt  
10 programming.txt
```

```
$ cat programming.txt | wc -l  
10
```

pipe



```
$ ls -l | wc -l  
1184
```

Nombre de lignes (donc de fichiers)
dans le répertoire courant

Commandes Bash

wc (word count) pour compter

- Compter le **nombre d'apparitions d'un mot** dans un fichier est **complexe**.
- Compter le **nombre de lignes où il apparaît** est plus **facile**

```
$ cat notes | grep the | wc -l
32
$ cat notes | grep [Tt]he | wc -l
40
```

Nombre de lignes avec le mot « the »

Nombre de lignes avec le mot
« the » ou « The »



Nouvelle commande

Exercices

9. Quelle commande devez-vous utiliser pour avoir la liste des fichiers de `home/Man_Shell` ?
10. Copiez maintenant le fichier `microbiome.txt` disponible sur e-campus
11. Affichez les 5 premières lignes de `microbiome.txt`.
12. Affichez les 4 dernières lignes de `microbiome.txt`.
13. Comptez le nombre de lignes de `microbiome.txt`.

Commande grep

- `grep [options] <expression> <fichiers/répertoires>`

Global Regular Expression Parser

- **grep** **cherche** la **chaîne de caractères** <expression> dans les **fichiers** ou des **répertoires** donnés en argument et **affiche** les lignes correspondantes.
- Cette commande permet l'utilisation d'**expressions régulières** (voir plus loin)

Commande grep

- La commande **grep** permet de capturer un motif dans un texte (**G**lobal **R**egular **E**xpression **P**arser)
 - `grep "Paris" fichier`
 - Quel effet ?
 - `grep -C "Paris" fichier`
 - Quel effet ?
 - `grep "^Paris" fichier`
 - Quel effet ?
 - `grep "Paris" repertoire1/*`
 - Quel effet ?
 - `grep -v "Paris" fichier`
 - Quel effet ?
 - `grep -i -v "Paris" fichier`
 - Quel effet ?

Commande grep

- La commande **grep** permet de capturer un motif dans un texte (**G**lobal **R**egular **E**xpression **P**arser)
 - `grep "Paris" fichier`
 - Affiche toutes les lignes de fichier qui contiennent « Paris »
 - `grep -c "Paris" fichier`
 - Affiche le nombre de lignes de fichier qui contiennent « Paris »
 - `grep "^Paris" fichier`
 - Affiche les lignes de fichier qui commencent par « Paris »
(le méta caractère ^ signifie « qui commence par »)
 - `grep "Paris" repertoire1/*`
 - Affiche la liste des lignes des fichiers de repertoire1 qui contiennent « Paris »
 - `grep -v "Paris" fichier`
 - Affiche toutes les lignes de fichier qui **ne** contiennent **pas** « Paris »
 - `grep -i -v "Paris" fichier`
 - Affiche toutes les lignes de fichier qui **ne** contiennent **pas** « Paris » quelque soit la casse
(-i signifie ne pas faire attention à la casse)

Commande grep

- grep -A3 "Paris" fichier
 - Quel effet ?
- grep -B2 "Paris" fichier
 - Quel effet ?
- grep -C5 "Paris" fichier
 - Quel effet ?
- grep -e "Paris" -e "Londres" fichier
 - Quel effet ?
- grep -E "Paris|Londres" fichier
 - Quel effet ?
- grep "^#" fichier
 - Quel effet ?
- grep "\. \$" fichier
 - Quel effet ?

Commande grep

- grep **-A3** "Paris" fichier
 - Affiche 3 lignes de fichier **après** (After) le mot« Paris »
- grep **-B2** "Paris" fichier
 - Affiche 2 lignes de fichier **avant** (Before) le mot« Paris »
- grep **-C5** "Paris" fichier
 - Affiche 5 lignes de fichier **avant et après** le mot« Paris »
- grep **-e** "Paris" **-e** "Londres" fichier
 - Affiche toutes les lignes de fichier qui **contiennent** « Paris » ou « Londres »
- grep **-E** "Paris|Londres" fichier
 - Idem en utilisant une **expression régulière** (voir plus loin)
- grep **"^#"** fichier
 - Pour afficher les lignes commençant par # (donc les lignes de commentaires)
- grep **"\.\$"** fichier
 - Le backslash \ pour afficher les lignes se terminant par un point (\$ signifie se terminant)
Sinon, renvoie les lignes se terminant par n'importe quel caractère (sauf return)

Exercices

14. Comptez le nombre de 'e' dans le texte
15. Recherchez tous les chiffres dans `microbiome.txt` et les afficher
16. Combien y en a-t-il ?

Exercices

14. Comptez le nombre de 'e' dans le texte

```
grep -o 'e' microbiome.txt | wc -l
```

-o : ne s'apparie qu'avec (**only**) le pattern (ne renvoie pas la ligne contenant un ou des 'e', mais seulement une ligne par 'e' rencontré).

15. Recherchez tous les chiffres dans `microbiome.txt` et les afficher

```
grep -o -E '[0-9]' microbiome.txt
```

-E : Car on utilise une **expression régulière**. Mais pas obligatoire ici.

16. Combien y en a-t-il ?

```
grep -o -E '[0-9]' microbiome.txt | wc -l
```

`wc -l` : Car renvoie **une ligne par chiffre**. Si `wc -c`, alors on compte deux fois, car le 'return' à la fin de chaque ligne est compté comme un caractère

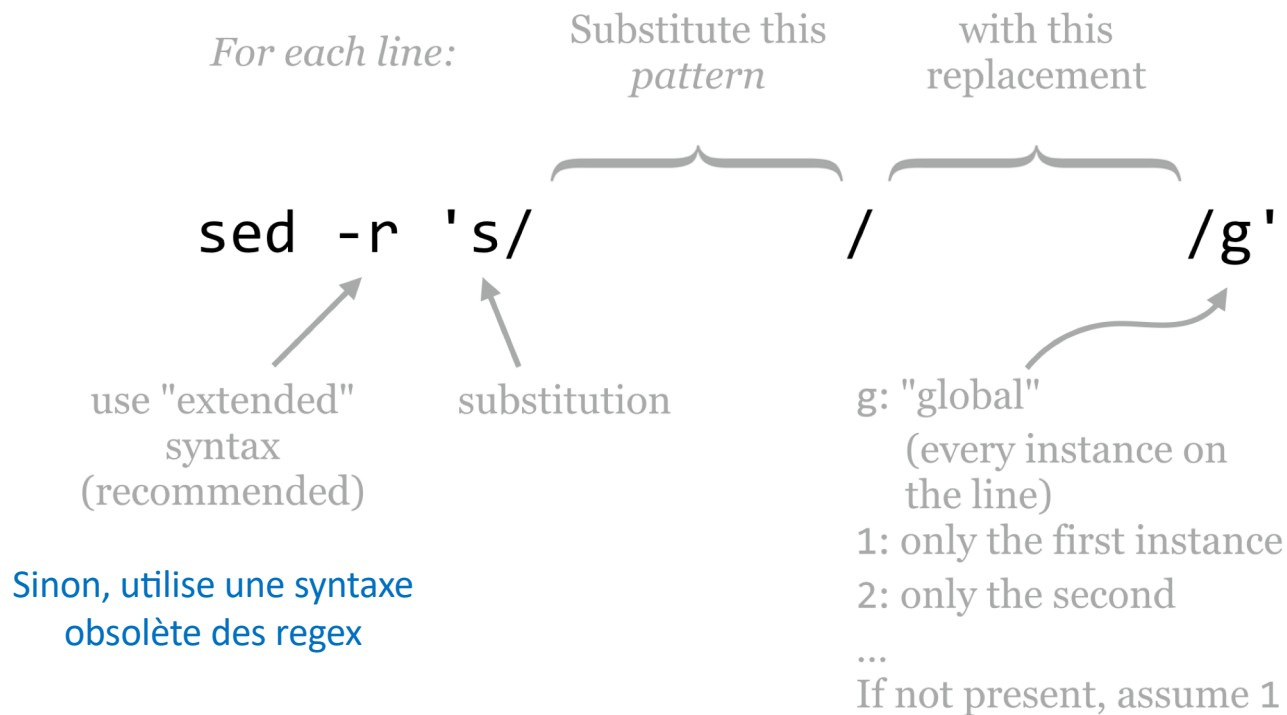
Fin 1^{er} cours

Commande **sed** : manipulation de **textes**

- `sed [options] 'fonctions' <fichier-entrée>`
 - La commande `sed` (Stream **ED**itor) **reçoit du texte en entrée**, que ce soit à partir de `stdin` ou d'un fichier, réalise certaines opérations sur les lignes spécifiées de l'entrée, une ligne à la fois, puis **sort le résultat vers `stdout`** ou vers un fichier.
 - `sed` détermine sur quelles lignes de son entrée **elle va opérer** à partir d'une *plage d'adresses* qui lui aura été fournie. Cette plage d'adresses est définie soit par des *numéros de ligne* soit par un *motif* à rechercher.
Par exemple, `3d` indique à `sed` qu'il doit supprimer la ligne 3 de l'entrée et `/windows/d` dit à `sed` que vous voulez que toutes les lignes de l'entrée contenant « windows » soient supprimées.
 - La commande **`sed`** peut exécuter plusieurs fonctions à la fois, fonctions séparées par un « ; ».
 - Cette commande permet l'utilisation d'**expressions régulières** (voir plus loin)
- `sed '4d; 7d' test.txt`
 - les lignes 4 et 7 du fichier `test.txt` sont effacées et le résultat est affiché à l'écran
- `sed '4d; 7d' test.txt > test2.txt`
 - les lignes 4 et 7 du fichier `test.txt` sont effacées et le résultat est mis dans le fichier `test2.txt` (en l'écrasant s'il existe)

Commande sed

- Le plus souvent employé pour réaliser des **substitutions**



- `sed 's/Ontario/Amérique/g' fichier.txt`
 - Toutes les occurrences** de « Ontario » sont remplacées (substituées) par « Amérique » dans fichier.txt (seulement la 1^{ère} occurrence si on ne met pas l'option 'g')

Options sed

- **-i** : **modifie le fichier sur place** au lieu d'afficher le résultat sur la sortie standard. À manier avec **précaution**, car il n'y a pas de retour en arrière.
- **-i.bak** : même chose, mais en conservant une copie de sauvegarde du fichier original (ici suffixée en .bak).
- **-E** : active les **expressions régulières** étendues (ERE), plus lisibles pour les motifs complexes (parenthèses, +, ? sans échappement).
- **-n** : masque l'affichage automatique de chaque ligne, à coupler avec la commande d'affichage **p** pour n'afficher que ce qui vous intéresse.
- **-e** : permet d'**enchaîner plusieurs expressions** sed dans une même commande.

```
flo@debian-13:~/Documents$ cat test.txt
```

```
first line
```

```
second line
```

```
third line
```

```
fourth line
```

```
flo@debian-13:~/Documents$ sed -i '1d' test.txt
```

```
flo@debian-13:~/Documents$ cat test.txt
```

```
second line
```

```
third line
```

```
fourth line
```

```
flo@debian-13:~/Documents$ █
```

On efface la 1^{ère}
ligne du fichier



• ...

Soit le nouveau fichier test.txt

```
(base) MacBook-Pro-de-ANTOINE-5:Man_Unix antoinecornuejols$ cat > test.txt
```

```
hello
```

```
# world
```

```
# it connect
```

```
it connect
```

- On veut **supprimer** toutes les lignes commençant par **#**

```
sed -i '/^#/d' test.txt
```

```
sed -i '' '/^#/d' test.txt
```

 (avec **Mac OSX** : ajout d'un 1^{er} argument vide)

Exercices

- 17.** Remplacez tous les 'a' de `microbiome.txt` par des points d'interrogation
- 18.** Supprimez la 3^{ème} ligne de `microbiome.txt` et mettez le résultat dans un fichier `microbiome_bis.txt`.
- 19.** Supprimez les lignes de `microbiome.txt` contenant le mot 'clock' et mettez le résultat dans un fichier `withoutclock.txt`.
- 20.** Remplacez toutes les minuscules par des majuscules dans le texte « BonJouR
lA vIe et bIENvEnUE »
- 21.** Charger le fichier 'Saccharomyces_cerevisiae.R64-1-
1.112.chromosome.Mito.gff3' le **copier** dans votre répertoire et le
renommer 'exemple_genome.gff3'

Commande awk

- Pour sélectionner des *lignes* et des *colonnes*

```
oneils@atmosphere ~/apcb/intro/blast$ cat yeast_blastp_yeast_top2.txt | \
> grep -v '#' | \
> awk '{print $1,$2}'
YAL001C YAL001C
YAL002W YAL002W
YAL003W YAL003W
...
```

```
oneils@atmosphere ~/apcb/intro/blast$ cat yeast_blastp_yeast_top2.txt | \
> grep -v '#' | \
> awk '{print $1"::"$2}'
YAL001C::YAL001C
YAL002W::YAL002W
YAL003W::YAL003W
...
```

```
oneils@atmosphere ~/apcb/intro/blast$ cat yeast_blastp_yeast_top2.txt | \
> grep -v '#' | \
> awk '{print NR,$0}'
1 YAL001C      YAL001C 1161    1161    1161    1      1161    1      1161
2 YAL002W      YAL002W 1275    1275    1275    1      1275    1      1275
3 YAL003W      YAL003W 207     207     207     1      207     1      207
...
```

The diagram illustrates the mapping of logical operators to their awk syntax equivalents. It features five columns of text at the top, each with a downward-pointing arrow leading to a corresponding part of an awk if-statement. The columns are: 'Not equal to' (arrow to '!='), 'And' (arrow to '&&'), 'Less than or equal to' (arrow to '<='), 'Or' (arrow to '||'), and 'Equal to' (arrow to '=='). Below these, the full awk command is shown: `awk '{if($1 != $2 && ($10 <= 1e-30 || $2 == "YAL044C")) {print $0}}'`. A large curly brace underlines the expression `($10 <= 1e-30 || $2 == "YAL044C")`, with the word 'grouping' centered below it.

Not equal to	And	Less than or equal to	Or	Equal to
↓	↓	↓	↓	↓
<code>awk '{if(\$1 != \$2 && (\$10 <= 1e-30 \$2 == "YAL044C")) {print \$0}}'</code>				
⏟ grouping				

• • •

Commande awk

```
oneils@atmosphere ~/apcb/intro/blast$ cat yeast_blastp_yeast_top2.txt | \
> grep -v '#' | \
> awk '{ \
> if($10 < 1e-30) {print "great",$1,$2,$10} \
> else if($10 < 1e-20) {print "good",$1,$2,$10} \
> else {print "ok",$1,$2,$10} \
> }'
...
great YAL018C YAL018C 0.0
good YAL018C YOL048C 4e-25
great YAL019W YAL019W 0.0
great YAL019W YOR290C 1e-84
great YAL020C YAL020C 0.0
great YAL021C YAL021C 0.0
good YAL021C YOL042W 6e-27
great YAL022C YAL022C 0.0
...
```

...

Commande awk

```
oneils@atmosphere ~/apcb/intro/blast$ cat yeast_blastp_yeast_top2.txt | \
> grep -v '#' | \
> awk '{if($1 == "YAL054C") {print $0}}'
```

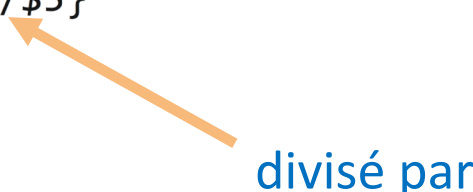
YAL054C	YAL054C	714	714	714	1	714	1	714	0.0
YAL054C	YLR153C	645	714	684	73	712	37	674	0.0

```
oneils@atmosphere ~/apcb/intro/blast$ cat yeast_blastp_yeast_top2.txt | \
> grep -v '#' | \
> awk '{print $1,$2,$4/$5}'
```

...

YAL017W	YAL017W	1
YAL017W	YOL045W	1.2314
YAL018C	YAL018C	1
YAL018C	YOL048C	0.950437
YAL019W	YAL019W	1
YAL019W	YOR290C	0.664319
YAL020C	YAL020C	1

...



Exercices avec awk

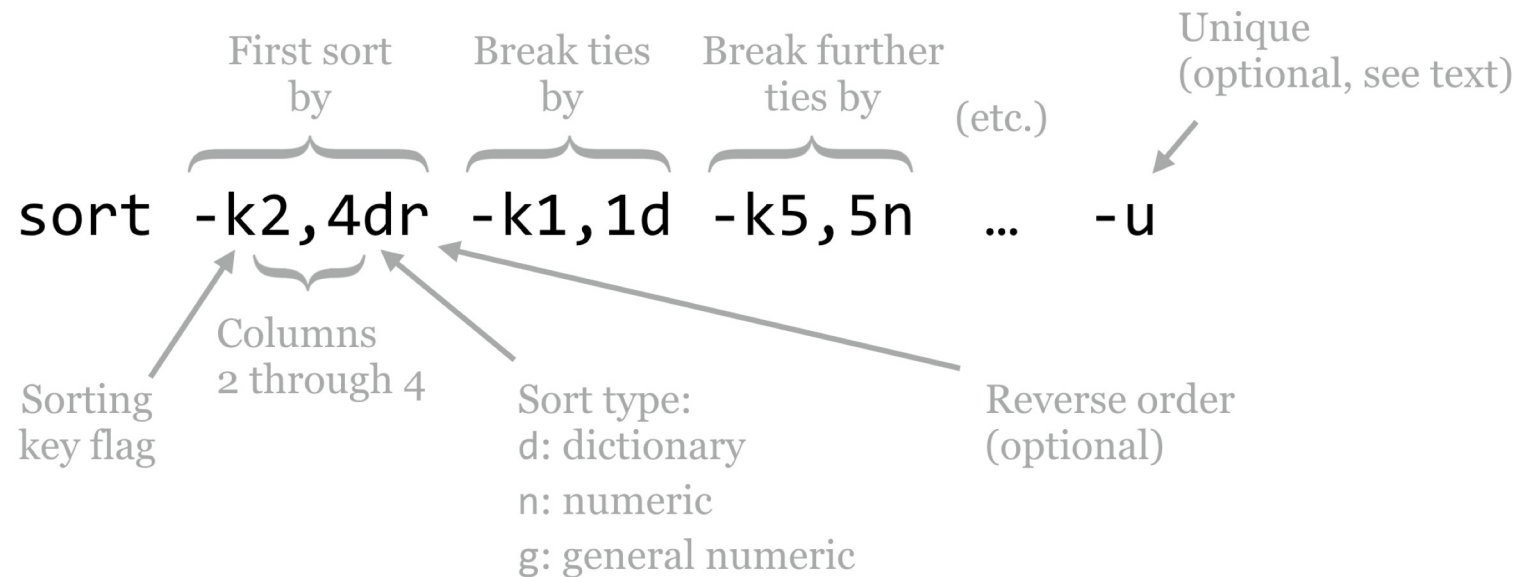
- 22. Imprimez le contenu des 10 1^{ères} lignes de `exemple_genome.gff3`
- 23. Imprimez les dix 1^{ères} lignes de la 1^{ère} colonne de `exemple_genome.gff3`
- 24. Imprimez les dix 1^{ères} lignes des colonnes 1 et 5 de `exemple_genome.gff3`
- 25. Imprimez les coordonnées de début et de fin **seulement si** la coordonnée de début (\$4) est > 10 000
- 26. Imprimez les coordonnées de début (\$4) et de fin (\$5) et la longueur de la séquence **seulement si** la coordonnée de début (\$4) est > 10 000
- 27. Imprimez les lignes qui **contiennent le mot 'intron'**
- 28. Imprimez la colonne 3 et le numéro de ligne des lignes **contenant le mot 'intron'**
- 29. Imprimez le nombre de lignes du fichier en utilisant la condition **END** vraie à la fin du fichier
- 30. Imprimez la **somme de toutes les longueurs d'introns**. Devrait imprimer : total length of introns = 48890

Exercices

31. Copiez le fichier '`comptage.txt`' sur e-campus
32. À partir de ce fichier créez un fichier nommé '`phylum.txt`' ne contenant que les noms de phylum (colonne 1)

Commande sort

- Pour trier les lignes



```

oneils@atmosphere ~/apcb/intro/fasta_stats$ ./fasta_stats pz_cDNAs_sample.fasta
# Column 1: Sequence ID
# Column 2: GC content
# Column 3: Length
# Column 4: Most common 5mer
# Column 5: Count of most common 5mer
# Column 6: Repeat unit of longest simple perfect repeat (2 to 10 chars)
# Column 7: Length of repeat (in characters)
# Column 8: Repeat type (dinucleotide, trinucleotide, etc.)
Processing sequence ID PZ7180000031590
PZ7180000031590 0.378 486 ACAA 5 unit:ATTTA 10 pentanucleotide
Processing sequence ID PZ718000000004_TX
PZ718000000004_TX 0.279 1000 AAATA 12 unit:TAA 12 trinucleotide

```

```

oneils@atmosphere ~/apcb/intro/fasta_stats$ ./fasta_stats pz_cDNAs.fasta | \
> grep -v '#' | \
> sort -k7,7nr | \
> less -S

```

Trier sur la 7^{ème} colonne en ordre inverse des nombres

7^{ème} colonne

PZ805359	0.0	101	ATATA	47	unit:AT 94	dinucleotide
PZ796092	0.365	361	TACGT	9	unit:GTACGT	48 hexanucl
PZ7180000019700	0.375	564	GAGTG	12	unit:GAGTG	30 pentanuc
PZ7180000028921	0.31	561	TGTAA	8	unit:CTGTG	30 pentanuc
PZ851952	0.399	338	TATAT	12	unit:AT 24	dinucleotide
PZ7180000000664_B	0.3	652	TTTTT	18	unit:TAAAATTAT	18
PZ7180000023622	0.31	687	TTAAT	9	unit:TGA	18 trinucle
PZ7180000023665_ATQ	0.401	508	ACTGA	5	unit:TGACACTGA	18
PZ7180000025478	0.516	829	TGATG	18	unit:ATG	18 trinucle
PZ7180000030412	0.461	258	TGATG	8	unit:ATG	18 trinucle
PZ7180000036892	0.268	548	AATAA	16	unit:TAA	18 trinucle
PZ801814	0.262	255	TTACA	5	unit:TATTACAT	18 enneanuc
...						

Exercices

- 33. Sortir les lignes de `phylum.txt` en ordre alphabétique inverse
- 34. Sélectionnez les lignes de '`comptage.txt`' pour lesquelles la valeur de la colonne 2 est supérieure à 0.1 et les sortir par ordre alphabétique sur la colonne 1
- 35. Et maintenant les sortir en ordre alphabétique inverse
- 36. Sortir les lignes (colonnes 1 et 2) de '`comptage.txt`' ordonnées sur la valeur de la colonne 2 quand elle est supérieure à 1.0

-
- Trier les 5 répertoires les plus gros (en espace disque)

```
du -s */ | sort -k1,1nr | head -5
```



Permet de connaître la **taille de**
tous les sous-répertoires du
répertoire courant

Tri par ordre
numérique **croissant**

du : « disk usage »

- Trier les fichiers par taille croissante (en espace disque)

```
ls -l | awk '{print $5, "\t », $9}' | sort -k1,1n | awk '{print $2}'
```

Imprimer les colonnes 5
(taille) et 9 (noms)
en les séparant par un 'tab'

Tri par ordre
numérique **croissant**

Imprimer la colonne 2
(le nom maintenant)

Plan

1. L'environnement Unix
2. Travailler avec des motifs : les expressions régulières
3. Scripts Bash et programmes Python
4. L'installation de logiciels

Les Expressions Régulières

(regex)

- Les **expressions régulières**, ou regex, sont *un mini langage* permettant de **rechercher** des motifs génériques dans un texte ou une chaîne de caractères, ainsi que d'effectuer des **substitutions** de chaînes par motifs.
- La **plupart des langages de programmation** *offrent un module d'expressions régulières* soit directement inclus dans le langage, soit sous forme d'un paquet additionnel. (module **re** pour Python)
- *Elles sont une magie ancienne et très puissante, et donc aussi forcément dangereuses. Il faut les manipuler avec sagesse et précaution.*

-
- Numéro de téléphone
 - 416 555 1212
 - (416) 555-1212
 - 416-555-1212
 - Et si, par erreur : 416-555-!212

Comment faire

pour tous les reconnaître comme des numéros de téléphone ?

Les expressions régulières

- Permettent de **spécifier des motifs** (« *patterns* ») à vérifier dans des textes
 - Par exemple pour extraire des sous-séquences d'un texte
- Très **puissant**
- Mais **complexe**
 - Par exemple, un **même symbole** peut voir **plusieurs significations** selon le contexte
 - Existe dans plusieurs langages :
 - En tant que librairies : Python, Java, ...
 - Dans le langage lui-même : Perl, Ruby, ...

Illustrations

- `rm *.java`
- `rm t*.java`
- `rm *t*`
- `rm *`
- `rm *.htm?`
- `rm nov-0?.log`
- `cp *.Jj]ava / tmp`
- `cp version[0-9].txt /tmp`

Illustrations

- a^*
- b^+
- $ab?c???$
- $a|b$
- $a|b|c|d$
- $[abc]$
- $[acgt]^+[ACGT]^+$
- $ab|cd$
- $a(b|c)d$
- $a^*(b|c)[xy]?$

Illustrations

- a^* chaîne commençant par 'a' et se continuant par une chaîne de caractères quelconque
 - Exemple : ab^* s'apparie avec ab, suivi de n'importe quelle chaîne de caractères
- b^+ chaîne commençant par b et **continuant par des b**
- $ab?c???$ chaîne commençant par 'ab', puis un caractère, puis 'c', puis 3 caractères
- $a|b$ chaîne a ou chaîne b
- $a|b|c|d$ chaîne a ou b ou c ou d
- $[abc]$ chaîne a ou b ou c
- $[acgt]^+[ACGT]^+$ chaîne de a ou c ou g ou t ou chaînes de A ou C ou G ou T donc toute séquence d'ADN entièrement en minuscules ou en majuscules
- $ab|cd$ chaîne ab ou chaîne cd
- $a(b|c)d$ chaîne abd ou acd
- $a^*(b|c)[xy]?$ chaîne commençant par des a, puis b ou c, puis x ou y

sed et expressions régulières

```
sed -r 's/[GC]{4,8}/_X_/g'
```

- ATCCGTCT

sed et expressions régulières

```
sed -r 's/[GC]{4,8}/_X_/g'
```

- ATCCGTCT -- > pas de remplacement

sed et expressions régulières

```
sed -r 's/[GC]{4,8}/_X_/g'
```

- ATCCGTCT -- > pas de remplacement
- ATCCGCGGCTC

sed et expressions régulières

```
sed -r 's/[GC]{4,8}/_X_/g'
```

- ATCCGTCT -- > pas de remplacement
- ATCCGCGGCTC
ATCCGCGGCTC -- > AT_X_TC

sed et expressions régulières

```
sed -r 's/[GC]{4,8}/_X_/g'
```

- ATCCGTCT -- > pas de remplacement
- ATCCGCGGCTC
ATCCGCGGCTC -- > AT_X_TC
- ATCGCGCGGCCCGTTTCGGGCCT

sed et expressions régulières

```
sed -r 's/[GC]{4,8}/_X_/g'
```

- ATCCGTCT -- > pas de remplacement
- ATCCGCGGCTC
ATCCGCGGCTC -- > AT_X_TC
- ATCGCGCGGCCCGTTTCGGGCCT
ATCGCGCGGCCCGTTTCGGGCCT -- > AT_X_CCGTT_X_T

Commande sed

- `sed '/^#/d' test.txt`

Commande sed

- `sed '/^#/d' test.txt`
 - supprime toutes les lignes débutant par un dièse (^ signifie « début de ligne »)
 - Les slash (/) permettent de bien séparer les motifs recherchés (ici ^#) de la commande (ici d pour delete)

Commande sed

- `sed '/^#/d' test.txt`
 - supprime toutes les lignes débutant par un dièse (^ signifie « début de ligne »)
 - Les slash (/) permettent de bien séparer les motifs recherchés (ici ^#) de la commande (ici d pour delete)

- `sed -n '/exemple.[1-9]/p' text.txt`

Commande sed

- `sed '/^#/d' test.txt`
 - supprime toutes les lignes débutant par un dièse (^ signifie « début de ligne »)
 - Les slash (/) permettent de bien séparer les motifs recherchés (ici ^#) de la commande (ici d pour delete)

- `sed -n '/exemple.[1-9]/p' text.txt`
 - Affiche toutes les lignes contenant « exemple » suivi d'un numéro
 - -n : n'affiche que ces lignes

Commande sed

- `sed '/^#/d' test.txt`
 - supprime toutes les lignes débutant par un dièse (^ signifie « début de ligne »)
 - Les slash (/) permettent de bien séparer les motifs recherchés (ici ^#) de la commande (ici d pour delete)

- `sed -n '/exemple.[1-9]/p' text.txt`
 - Affiche toutes les lignes contenant « exemple » suivi d'un numéro
 - -n : n'affiche que ces lignes

- `sed -n -e '1p' -e '/Chardonnay/p' vin.txt`

Commande sed

- `sed '/^#/d' test.txt`
 - supprime toutes les lignes débutant par un dièse (^ signifie « début de ligne »)
 - Les slash (/) permettent de bien séparer les motifs recherchés (ici ^#) de la commande (ici d pour delete)

- `sed -n '/exemple.[1-9]/p' text.txt`
 - Affiche toutes les lignes contenant « exemple » suivi d'un numéro
 - -n : n'affiche que ces lignes

- `sed -n -e '1p' -e '/Chardonnay/p' vin.txt`
 - Affiche la première ligne de la base de données (avec les titres des colonnes)
 - Puis toutes les lignes contenant « chardonnay »
 - Étagère Région de culture Cépage Millésime
 - 1 Rhénanie-Palatinat Chardonnay 2001
 - 2 Moselle Chardonnay 1983
 - 3 Alsace Chardonnay 1981

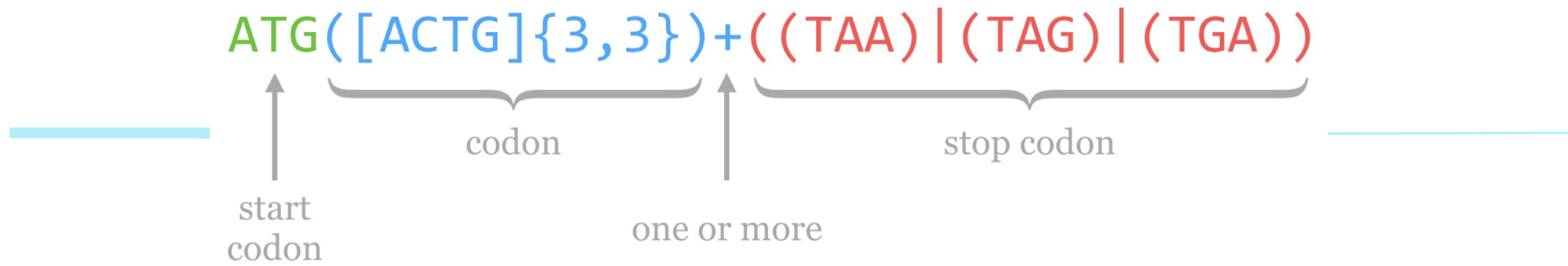
Commande sed

- Reconnaissance d'ORF (Open Reading Frame)
 - Codon de départ : ATG
 - Plus **un ou plusieurs** codons : une séquence de trois A ou T ou C ou G
 - $([ATCG]\{3,3\})^+$
 - Se terminant par TAA ou TAG ou TGA
 - $((TAA)|(TAG)|(TGA))$

$ATG([ACTG]\{3,3\})^+((TAA)|(TAG)|(TGA))$

The diagram illustrates the components of the sed command $ATG([ACTG]\{3,3\})^+((TAA)|(TAG)|(TGA))$ used for Open Reading Frame (ORF) recognition. Annotations include:

- An upward arrow pointing to **ATG** with the label "start codon".
- A bracket under **[ACTG]** with the label "codon".
- An upward arrow pointing to the **{3,3}** part of the quantifier with the label "one or more".
- A bracket under **((TAA)|(TAG)|(TGA))** with the label "stop codon".



Input: TATGCATGTTTAGCTTTTAG

ORF 1: ATGCATGTTTAG
start stop

ORF 2: ATGCATGTTTAG
start stop

ORF 3: ATGTTTAGCTTTTAG
start stop

Passage à la ligne si la commande est trop longue

- Sed est glouton : ne retourne **que le 1^{er} appariement possible**

```
(base) mbpdeantoine5:Man_Unix antoinecornuejols$ echo "TATGCATGTTTAGCTTTTAG" | \
```

```
> sed -r 's/ATG([ACTG]{3,3})+((TAA)|(TAG)|(TGA))/_ORF_/g'
```

```
T_ORF_CTTT TAG      (car T ATG CAT GTT TAG TAG CTTT TAG)
```

Plan

1. L'environnement Unix
2. Travailler avec des motifs : les expressions régulières
3. Scripts Bash et programmes Python
4. L'installation de logiciels

Redirection et gestion de flux

- commande `> monfichier.txt`
 - Exécute et écrase monfichier.txt avec le résultat
 - `ls > desktop/liste_files.txt`
- commande `>> monfichier.txt`
 - Copie le résultats à la suite de monfichier.txt
- commande `< monfichier.txt`
 - La commande prend en entrée monfichier.txt
 - `sort < desktop/test.txt > desktop/ls_trie.txt`
 - Sort utilise desktop/test.txt et envoie le résultat dans le fichier desktop/ls_trie.txt
- Connecter plusieurs commandes avec `|`
 - `ls | sort`
 - Liste les fichiers par ordre alphabétique

Commande sudo

- **sudo** **super user do**

- Permet d'utiliser les droits administrateur pour exécuter une commande

- adrien@linux: ~ \$ **ls** /root

- ls: impossible d'ouvrir le répertoire /root: Permission non accordée

- adrien@linux: ~ \$ **sudo ls** /root [**sudo**]

- password **for** adrien:

- script-de-root.sh

- Le mot de passe est mémorisé par défaut pour 5mn
- **sudo -k** termine la session si nécessaire

Créer un fichier exécutable

- Qu'est-ce qu'un programme ? (du point de vue du système)

Créer un fichier exécutable

- Qu'est-ce qu'un programme ? (du point de vue du système)
 - Pour Unix, c'est un fichier avec une permission **x** (executable)
 - Les exécutables sont souvent en **format binaire**
 - Comme `echo` ou `ls`
 - Ils sont par exemple stockés dans un répertoire **bin**

```
oneils@atmosphere ~$ cd /bin
oneils@atmosphere /bin$ ls -l
total 7384
-rwxr-xr-x 1 root root 959120 Mar 28 2013 bash
-rwxr-xr-x 1 root root 31112 Dec 14 2011 bunzip2
-rwxr-xr-x 1 root root 31112 Dec 14 2011 bzip2
...
```

Créer un fichier exécutable

- Normalement, pour **appeler un programme**, il faut fournir son adresse absolue ou relative
- Pourtant pas pour echo ou ls !!??

```
oneils@atmosphere ~/apcb/intro$ /bin/echo hello  
hello
```

NON

```
oneils@atmosphere ~/apcb/intro$ ../../../../bin/echo hello  
hello
```

Ouf !!!

- Comment est-ce possible ?

- La variable \$PATH

```
oneils@atmosphere ~/apcb/intro$ echo $PATH  
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

- Permet au shell de **savoir où chercher** ces fichiers
(liste de **chemins absolus**)

Créer un fichier exécutable

- Si plusieurs chemins contiennent un fichier
- On peut savoir lequel est utilisé : commande `which`

```
oneils@atmosphere ~$ which echo  
/bin/echo
```

- Mais :

```
oneils@atmosphere ~/apcb/intro$ which cd  
oneils@atmosphere ~/apcb/intro$
```

— Pourquoi ?

Créer un fichier exécutable

- Si plusieurs chemins contiennent un fichier
- On peut savoir lequel est utilisé : commande `which`

```
oneils@atmosphere ~$ which echo  
/bin/echo
```

- Mais :

```
oneils@atmosphere ~/apcb/intro$ which cd  
oneils@atmosphere ~/apcb/intro$
```

- Pourquoi ?
- C'est une **commande** du shell et **non** un programme

La variable PATH

- Pour pouvoir utiliser un programme installé à l'emplacement `/user/local/maple/bin/maple` :

```
PATH = $PATH:/user/local/maple/bin
```

Pour utiliser directement le fichier maple.

Créer un fichier exécutable

- Soit le fichier myprog.sh

```
#!/bin/bash  
  
echo "Hello!"  
echo "This is a bit weird..."
```

Un *script* Shell correspond à un **fichier exécutable** d'extension **.sh**
et commençant par : `#!/bin/bash`

Créer un fichier exécutable

- Essayons

```
oneils@atmosphere ~/apcb/intro$ pwd
/home/oneils/apcb/intro
oneils@atmosphere ~/apcb/intro$ ./myprog.sh
Hello!
This is a bit weird...
```

Wouaouh, ça marche !!

Nous venons de créer un **script**

exécuté par un interpréteur, ici bash

Créer un fichier exécutable

- Si nous **changeons de répertoire**, ne pas oublier de faire appel au script en **changeant le chemin d'accès** (absolu ou relatif)

`$HOME = /home/oneils`

```
oneils@atmosphere ~/apcb/intro$ cd $HOME
```

```
oneils@atmosphere ~$ /home/oneils/apcb/intro/myprog.sh
```

```
Hello!
```

```
This is a bit weird...
```

chemin **absolu**

```
oneils@atmosphere ~$ apcb/intro/myprog.sh
```

```
Hello!
```

```
This is a bit weird...
```

chemin **relatif**

- Ou bien ...

Modifier le \$PATH

Créer un fichier exécutable

- Pour **ajouter** un fichier exécutable
 1. Créer un programme ou un script
 2. Le placer dans un répertoire
 3. S'assurer que le chemin absolu peut être trouvé dans \$PATH
- En général, on met ses fichiers exécutables **dans**
 - Son répertoire home
 - local
 - bin

```
oneils@atmosphere ~$ cd $HOME
oneils@atmosphere ~$ mkdir local
oneils@atmosphere ~$ mkdir local/bin
oneils@atmosphere ~$ ls
apcb      Documents  local      Pictures   Templates  Videos
Desktop  Downloads  Music     Public     todo_list.txt
oneils@atmosphere ~$ mv apcb/intro/myprog.sh local/bin
```

Créer un fichier exécutable

- Et on **modifie** la variable \$PATH

Ne pas oublier **export**



```
oneils@atmosphere ~$ export PATH="$HOME/local/bin:$PATH"
oneils@atmosphere ~$ echo $PATH
/home/oneils/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
:/usr/games
```

```
oneils@atmosphere ~$ myprog.sh
Hello!
This is a bit weird...
```

- Voir [Shawn T. O’Neil (2019) « A primer for Computational Biology », Oregon State University], p.46 pour plus de détails (modification de .bashrc)

La programmation shell

Le shell

- On a vu :
 - L'interprète de commandes (shell) permet d'interagir avec le système
 - L'exécution de commandes : modification et consultation de l'état du système
 - Utilisation avancée : combinaison de commandes, par exemple pour rediriger des entrées / sorties
- **Programmation shell**
 - Combinaison de commandes au sein d'un *script* dans le but d'automatiser certaines tâches
 - Un *script* Shell correspond à un **fichier exécutable** d'extension **.sh** et commençant par : `#!/bin/bash`

Les variables

- ATTENTION : Le caractère \$ est obligatoire pour évaluer les paramètres. Sans quoi, ils sont interprétés comme des mots.
- Penser à \$ comme un opérateur « get the value of »

Opérateurs logiques

- Sur les **fichiers**

- [**-e** fichier] : Vrai si le fichier/répertoire **existe**
- [**-s** fichier] : Vrai si le fichier a une **taille** supérieure à 0
- [**-r** fichier] : Vrai si le fichier/répertoire est **lisible**
- [**-w** fichier] : Vrai si le fichier/répertoire est **modifiable**
- [**-x** fichier] : Vrai si le fichier/répertoire est **exécutable**
- [f1 **-nt** f2] : Vrai si les deux fichiers existent et le fichier f1 est **plus récent** que le fichier f2

À voir

Opérateurs logiques

- Sur les entiers
 - [entier1 -eq entier2] : Vrai si entier1 est égal à entier 2
 - [entier1 -ge entier2] : Vrai si entier1 est supérieur ou égal à entier 2
 - [entier1 -gt entier2] : Vrai si entier1 est strictement supérieur à entier 2
 - [entier1 -le entier2] : Vrai si entier1 est inférieur ou égal à entier 2
 - [entier1 -lt entier2] : Vrai si entier1 est strictement inférieur à entier 2
 - [entier1 -ne entier2] : Vrai si entier1 est différent de entier 2

Opérateurs logiques

- Sur les **chaînes de caractères**
 - [**-n** chaine] : Vrai si chaine n'est **pas vide**
 - [**-z** chaine] : Vrai si chaine est **vide**
 - [chaine1 **=** chaine2] : Vrai si chaine1 et chaine2 sont **identiques**
 - [chaine1 **!=** chaine2] : Vrai si chaine1 et chaine2 ne sont **pas identiques**

Exemple de script

```
#!/bin/bash
if [ -e script_1.sh ]
then
    echo "Mon fichier existe"
else
    echo "Mon fichier n'existe pas"
fi
```

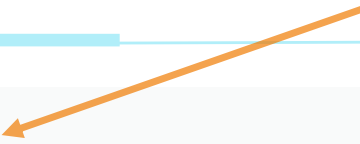
- Soit le script ci-dessus (attention aux espaces après et avant les crochets)
- Copier le dans un fichier « script_1.sh »
- Pour l'exécuter :
 `chmod u+x script_1.sh` (afin de le rendre exécutable)
 `./script_1.sh`

Exemple de script

```
#!/bin/bash
if [ -e script_1.sh ]
then
    echo "Mon fichier existe"
else
    echo "Mon fichier n'existe pas"
fi
```

- Que fait-il ?

Exemple de script avec **argument**



```
#!/bin/bash
if [ -e script_1.sh ]
then
    echo "Mon fichier existe"
else
    echo "Mon fichier n'existe pas"
fi
```

- Reprendre ce script en lui fournissant maintenant en **argument** le nom d'un fichier (eg. monfichier.txt) : script_2.sh
- Pour l'**exécuter** :
 - `chmod u+x script_2.sh` (afin de le rendre exécutable)
 - **Utilisation** : `./script_2.sh monfichier.txt`

Script avec invite d'argument

```
#!/bin/bash

echo -n "Please enter a file name : "
read filename

if [ -e $filename ]
then
    echo "Ce fichier existe : " $filename
else
    echo "Ce fichier n'existe pas"
fi

# Utilisation : ./script_3.sh
```

- À essayer
- Ne pas oublier de modifier les droits : `chmod u+x script_3.sh`

Exercice

37. Écrire un programme utilisant une **boucle while** et qui écrit
« La variable i vaut (sa valeur) » 4 fois en démarrant par $i=1$

Exercice

- Écrire un programme utilisant une **boucle while** et qui écrit « La variable i vaut (sa valeur) » 4 fois en démarrant par i=1

```
#!/bin/bash

i=1

while [ $i -le 4 ]
do
    echo "la variable i vaut $i"
    i=$((i+1))
done

# Utilisation : ./script_4.sh
```

Attention : **pas d'espace** autour
l'opérateur =

In Bash, **`$(( ))`** is used for **arithmetic expansion**. It allows you to perform mathematical operations inside the double parentheses.

Remarque : l'indentation n'est pas prise en compte par l'interpréteur. Elle permet juste d'écrire un code plus lisible.

Exercice

38. Écrire un programme utilisant une **boucle while** qui **demande un mot** et qui redemande tant que le mot donné par l'utilisateur n'est pas « fin »

Exercice

- Écrire un programme utilisant une **boucle while** qui **demande un mot** et qui redemande tant que le mot donné par l'utilisateur n'est pas « fin »

```
#!/bin/bash

echo -n "Please enter a variable name : "
read var1
echo $var1

while [ $var1 != "fin" ]
do
    echo "variable entree (quitter en entrant le mot fin) " $var1
    echo -n "Please enter a variable name : "
    read var1
done

# Utilisation : ./script_5.sh
# s'exécute jusqu'à ce que l'utilisateur entre "fin" (sans les guillemets)
```

Récupérer les appariements en Python

- C'est bien de **savoir** qu'un motif a été retrouvé dans une expression (appariement)
- C'est aussi parfois utile de savoir quel **appariement** a été réalisé
- Exemple

```
>>> m = re.match(r"(\w+) (\w+)", "Isaac Newton, physicist")
>>> m.group(0)           # The entire match
'Isaac Newton'
>>> m.group(1)           # The first parenthesized subgroup.
'Isaac'
>>> m.group(2)           # The second parenthesized subgroup.
'Newton'
>>> m.group(1, 2)        # Multiple arguments give us a tuple.
('Isaac', 'Newton')
```

Récupérer les appariements en Python

```
>>> import re
>>> m = re.search('(?<=abc)def', 'abcdef')
>>> m.group(0)
'def'
```

This example looks for a word following a hyphen:

```
>>> m = re.search(r'(?<=-)\w+', 'spam-egg')
>>> m.group(0)
'egg'
```

search: Scan through *string* looking for **the first location** where the regular expression *pattern* produces a match, and return a corresponding Match. Return **None** if no position in the string matches the pattern;

match: if zero or more characters **at the beginning of *string*** match the regular expression *pattern*, return a corresponding Match. Return **None** if the string does not match the pattern

Récupérer les appariements en Python

- C'est bien de **savoir** qu'un motif a été retrouvé dans une expression (appariement)
- C'est aussi parfois utile de savoir quel **appariement** a été réalisé

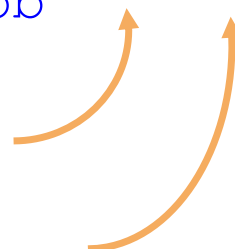
- Exemple

```
m = re.search("a(b*)c", "_abbbbc_")
```

```
— m.group(1)      →      bbbb
```

```
— m.start(1)      →      2
```

```
— m.end(1)        →      6
```



Exemple (2)

From gvwilson@cs.utoronto.ca Mon Nov 1 06:01:22 2004 -0500

Date: Mon, 29 Nov 2004 06:01:22 -0500 (EST)

From: Greg Wilson <gvwilson@cs.utoronto.ca>

To: andy@pragprog.com



Subject: Re: Unicode characters

Message-ID: Pine.GSO.4.58.0411010.1613@pterry.third-bit.com

In-Reply-To: <1099289055.4185d1df90155@pragprog.com>

MIME-Version: 1.0

Content-Type: TEXT/PLAIN; charset=US-ASCII

p.s. a few Cyrillic characters would be cool too, don't you think?

Thanks,

Greg

On aimerait récupérer le destinataire

Exemple (2)

```
From gvwilson@cs.utoronto.ca Mon Nov 1 06:01:22 2004 -0500
Date: Mon, 29 Nov 2004 06:01:22 -0500 (EST)
From: Greg Wilson <gvwilson@cs.utoronto.ca>
To: andy@pragprog.com
Subject: Re: Unicode characters
Message-ID: Pine.GSO.4.58.0411010.1613@pterry.third-bit.com
In-Reply-To: <1099289055.4185d1df90155@pragprog.com>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=US-ASCII
```

p.s. a few Cyrillic characters would be cool too, don't you think?
Thanks,
Greg

Lignes commençant par *To*:

```
import sys, re
for line in sys.stdin:
    m = re.search("To: +(.+)", line)
    if m :
        print(m.group(1))
```

Programmation en Python

Alors ?

Exemple (2)

```
From gvwilson@cs.utoronto.ca Mon Nov 1 06:01:22 2004 -0500
Date: Mon, 29 Nov 2004 06:01:22 -0500 (EST)
From: Greg Wilson <gvwilson@cs.utoronto.ca>
To: andy@pragprog.com
Subject: Re: Unicode characters
Message-ID: Pine.GSO.4.58.0411010.1613@pterry.third-bit.com
In-Reply-To: <1099289055.4185d1df90155@pragprog.com>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=US-ASCII
```

```
p.s. a few Cyrillic characters would be cool too, don't you think?
Thanks,
Greg
```

Lignes commençant par *To*:

```
import sys, re
for line in sys.stdin:
    m = re.search("To: +(.+)", line)
    if m :
        print(m.group(1))
```

```
andy@pragprog.com
<1099289055.4185d1df90155@pragprog.com>
```

Programmation en Python

- **To:** matches the literal string "To: ".
- **+** matches **one or more spaces**.
- **(.+)** captures **one or more of any character** (except newline) and **groups it** (this is the part you can extract).

Ah zut !

Exemple (2)

```
From gvwilson@cs.utoronto.ca Mon Nov 1 06:01:22 2004 -0500
Date: Mon, 29 Nov 2004 06:01:22 -0500 (EST)
From: Greg Wilson <gvwilson@cs.utoronto.ca>
To: andy@pragprog.com
Subject: Re: Unicode characters
Message-ID: Pine.GSO.4.58.0411010.1613@pterry.third-bit.com
In-Reply-To: <1099289055.4185d1df90155@pragprog.com>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=US-ASCII
```

p.s. a few Cyrillic characters would be cool too, don't you think?
Thanks,
Greg

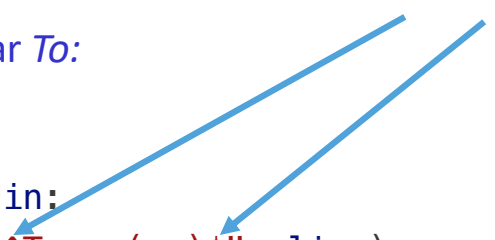
Lignes commençant par *To*:

```
import sys, re
for line in sys.stdin:
    m = re.search("^To: +(.+)$", line)
    if m :
        print(m.group(1))
```

andy@pragprog.com

Programmation en Python

« ancrés »



Gagné !

Exemple (3)

No address

Next line is blank

someone@somewhere.com

Line above matches at start, line below is indented:

 who@where.com

What about multiple matches?

first@first.com and second@second.com

What about dangling @ signs?

@nomatch.com

nomatch@ space after '@'

Let's try quotes: "a@b.com", <c@d.com>

And quotes in the middle: a'b@c.d

Longer domains with funny characters: p@q1.r-s.t-u

Short domains: x@y

On aimerait récupérer la liste de **toutes** les adresses email

Exemple (3) : 1^{ère} tentative

No address

Next line is blank

someone@somewhere.com

Line above matches at start, line below is indented:

 who@where.com

What about multiple matches?

first@first.com and second@second.com

What about dangling @ signs?

@nomatch.com

nomatch@ space after '@'

Let's try quotes: "a@b.com", <c@d.com>

And quotes in the middle: a'b@c.d

Longer domains with funny characters:

p@q1.r-s.t-u

Short domains: x@y

```
import sys, re
```

```
pattern = "([\w|\.|_]+@[ \w]+(\.|[\w|_]+)*)"
```

```
for line in sys.stdin:
```

```
    m = re.search(pattern, line)
```

```
    if m :
```

```
        address = m.group(1)
```

```
        print(address)
```

someone@somewhere.com

who@where.com

first@first.com

a@b.com

b@c.d

p@q1.r-s.t-u

x@y

Exemple (3) : 2^{ème} tentative

No address

Next line is blank

someone@somewhere.com

Line above matches at start, line below is indented:

who.name@where.com

jean-pierre.dupont@someplace.fr

marie-alice.martin@uni-best.com

What about multiple matches?

first@first.com and second@second.com

What about dangling @ signs?

@nomatch.com

nomatch@ space after '@'

Let's try quotes: "a@b.com", <c@d.com>

And quotes in the middle: a'b@c.d

Longer domains with funny characters: p@q1.r-s.t-u

Short domains: x@y

Comment obtenir :

someone@somewhere.com

who.name@where.com

x[?]?W !"#\$%&'()*+,-./0123456789:

x[?]?W !"#\$%&'()*+,-./0123456789:

first@first.com

second@second.com

a@b.com

c@d.com

b@c.d

p@q1.r-s.t-u

x@y

Exemple

No address

Next line is blank

someone@somewhere.com

Line above matches at start, line below is indented:

 who.name@where.com

jean-pierre.dupont@someplace.fr

marie-alice.martin@uni-best.com

What about multiple matches?

first@first.com and second@second.com

What about dangling @ signs?

@nomatch.com

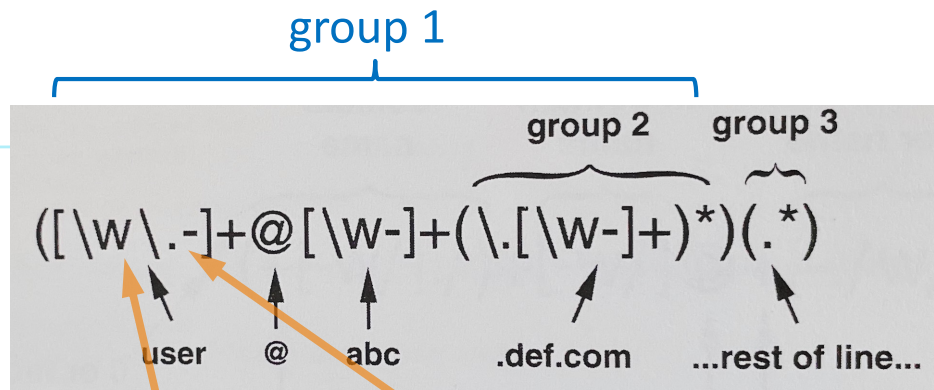
nomatch@ space after '@'

Let's try quotes: "a@b.com", <c@d.com>

And quotes in the middle: a'b@c.d

Longer domains with funny characters: p@q1.r-s.t-u

Short domains: x@y



N'importe quel
caractère = [a-zA-Z0-9_]

Pour avoir le point
et le tiret

someone@somewhere.com

who@where.com

jean-pierre.dupont@someplace.fr

marie-alice.martin@uni-best.com

first@first.com

second@second.com

a@b.com

c@d.com

b@c.d

p@q1.r-s.t-u

x@y

Exemple (3)

No address

Next line is blank

someone@somewhere.com

Line above matches at start, line below is i

who@where.com

What about multiple matches?

first@first.com and second@second.com

What about dangling @ signs?

@nomatch.com

nomatch@ space after '@'

Let's try quotes: "a@b.com", <c@d.com>

And quotes in the middle: a'b@c.d

Longer domains with funny characters: p@q1.r-s.t-u

Short domains: x@y

```
import sys, re

pattern = "([\\w\\.-]+@[\\w-]+(\\. [\\w-]+)*) (.*)"
for line in sys.stdin:
    while line:
        m = re.search(pattern, line)
        if m :
            address = m.group(1)
            remainder = m.group(3)
            print(address)
            line = remainder
        else:
            line = None
```

someone@somewhere.com

who@where.com

first@first.com

second@second.com

a@b.com

c@d.com

b@c.d

p@q1.r-s.t-u

x@y

Plan

1. L'environnement Unix
2. Travailler avec des motifs : les expressions régulières
3. Scripts Bash et programmes Python
4. L'installation de logiciels

Installation d'un logiciel

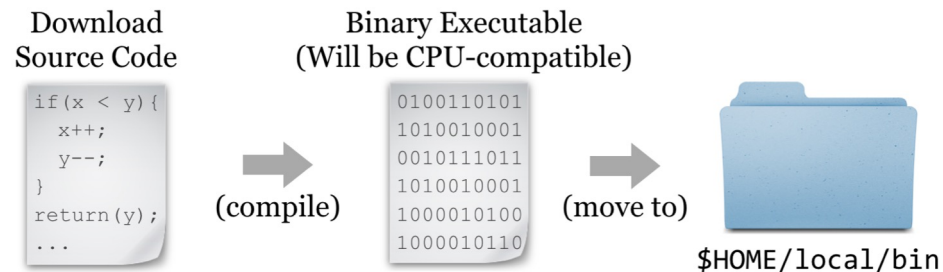
- Quelle différence entre fichier source et fichier binaire ?
- Que signifient les extensions tar.gz ?

Fichiers sources et fichiers binaires

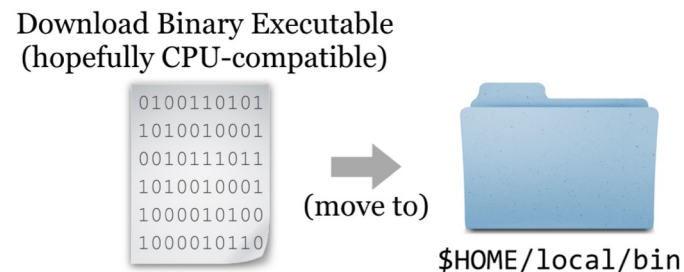
- Fichier **source**

- Fichier texte (human readable) commençant par la ligne `#!` référençant un interpréteur
- Qui a besoin d'être compilé (machine readable). Et cela dépend de la machine (e.g. processeur 32 bits ou 64 bits, Intel ou ARM (sur les nouveaux Mac))

→ Fichier **binaire**



VS



tar et gz

- tar **combine** plusieurs fichiers en une archive (tarball)
- gz (gzip) **comprime** le fichier fourni en argument

