BASH (BOURNE-AGAIN SHELL)

Guillaume Chanel

QU'EST-CE QU'UN SHELL ?

INTERFACE TEXTE

Pourquoi utiliser une interface texte en 2024 ?

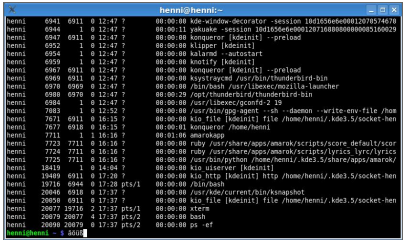
LE SHELL

- Interface textuelle, ligne de commande
- presque toutes les fonctionnalités d'Unix sont accessibles
- Petits outils simple que l'on peut composer
- Possibilité d'écrire des **scripts**

TERMINAL & PSEUDO-TERMINAL

terminal

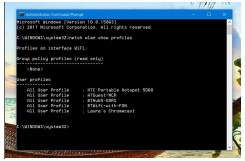
VT220



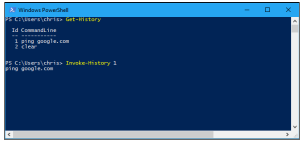
Pseudo-terminal

AUTRE INTERFACES EN LIGNE DE COMMANDE

WINDOWS

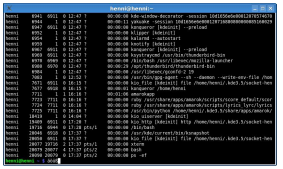


CMD



PowerShell

LINUX / MAC / UNIX



sh, **bash**, csh, zsh, fish, etc.

PRÉPARATION DE L'ENVIRONEMENT DE TRAVAIL

- Ouvrez un pseudo-terminal (si il n'est pas déjà ouvert)

- Qu'observez-vous ?

Tapez la commande suivante:

```
$ git clone https://gitlab.unige.ch/outils-info/bash.git
```

MANUEL

ACCÈDER AU MANUEL (`man`)

- Un manuel très complet est présent sur les systèmes Unix
- Commande: `man [section] <sujet>`
- Pour sortir du manuel, appuyer sur la touche `q`
- Pour chercher le texte "montexte", taper `/montexte` (valider par entrer)

Exemples

```
$ man pwd
```

```
$ man 3 getcwd
```

```
$ man man
```

```
$ apropos sujet  # permet de lister les pages d'un sujet
```

SECTIONS DU MANUEL SUR GNU/LINUX

1	Commandes générales
2	Appels systèmes
3	Librairie C
4	Fichiers spéciaux
5	Formats de fichiers et conventions
6	Jeux
7	Divers
8	Commandes d'administration et démons

RTFM !

SE LOCALISER ET SE DÉPLACER

PWD, LS ET CD

- Regarder dans quel dossier vous êtes
- Aller dans le dossier `/tmp`
- Vérifiez que vous êtes bien dans le dossier `/tmp`
- Lister le contenu du dossier
- Revenez dans votre dossier initial (i.e. dossier home)
- Allez dans le dossier bash (celui créer par la commande git)
- Lister le contenu du dossier /tmp depuis le dossier bash

CHEMIN D'ACCÈS

Les conventions suivantes (POSIX) s'appliquent:

- La racine est représentée par `/`
- Les répertoires sont séparés par `/`
- Tout chemin qui ne commence par `/` est relatif au répertoire courant.
- Le répertoire courant est symbolisé par `.`
- Le réperoire parent est symbolisé par `..`
- Le répertoire *home* est symbolisé par `~`

CHEMIN D'ACCÈS - EXERCICES

- Quel est le résultat des commandes ci-dessous ?

```
$ cd ~/./././bash
```

```
$ ls ~/./bash/../bash/../.
```

- Comment lister le contenu de votre dossier home depuis n'importe quel dossier (a tester)?

NOMMER UN ENSEMBLE DE FICHIERS

Les noms de fichiers suivent les règles suivantes:

- Maximum 255 caractères
- Tous les caractères sont autorisés, mais sont déconseillés:
  - espaces
  - `. * ? / !`
- Si le nom du fichier commence par un point, il est **caché**.

On peu référencer plusieurs fichiers en utilisant les symboles suivants:

***** remplace zéro, un ou plusieurs caractères

? un seul caractère

≡

FICHIERS CACHÉS ET SELECTION - EXERCICES

- Avec le manuel trouvez l'option de `ls` qui permet de lister les fichiers cachés
- Utiliser la pour lister les fichiers cachés de votre dossier home

---

- Rendez vous dans le dossier bash/ressources
- Listez dans le dossier images toutes les images au format JPEG
- Lister toutes les images d'oiseaux
- Lister toutes les images d'oiseaux au format PNG
- Lister les images de chat portant un numéro entre 90 et 99 (inclus)

LES UTILISATEURS, LES GROUPES ET LES DROITS

INFORMATIONS SUR LES FICHIERS/DOSSIERS

- Trouvez dans le manuel ce que fait la command `ls -lh`
- Observons ensemble le résultat...

UTILISATEURS

- Chaque utilisateur se loge sur une machine Unix avec une identité (*user*)
- Un utilisateur est décrit par:
  - Nom d'utilisateur
  - Mot de passe
  - Identifiant numérique (*userid*)
  - Groupe par défaut
  - Répertoire `home`
  - Description
  - Shell par défaut

SUPER-UTILISATEUR (ROOT)

- Il existe un super-utilisateur: **root**
- Son userid est le 0
- Il a le droit de tout faire.

  Accès au mot de passe root: faille de sécurité majeure.

GROUPES

Les utilisateurs peuvent faire partie d'un ou plusieurs **groupes**:

- Groupe par défaut
- Autres groupes

Ceci permet par exemple de:

- restreindre l'accès à un répertoire à un seul groupe
- donner accès à un périphérique à un seul groupe

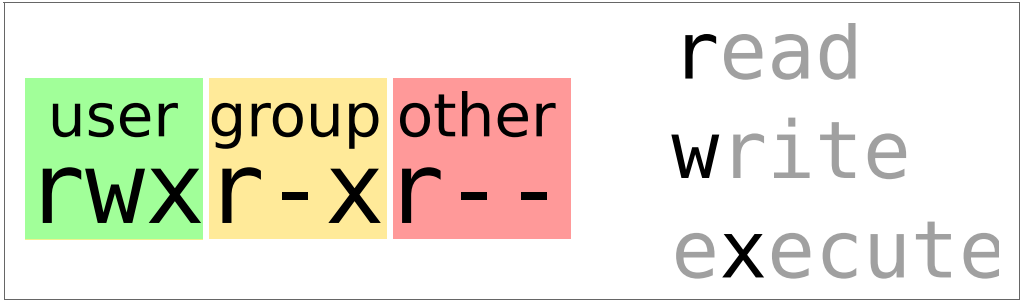
UTILISATEURS ET GROUPES - EXERCICES

- A l'aide de la commande `whoami` obtenez votre login
- A l'aide de la commande `groups` obtenez vos groupes
- Retrouvez ces information avec la commande `id`

La commande **`cat`** permet d'afficher un ou plusieurs fichiers à l'écran (concatenation, voir manuel):

- utilisez cat sur le fichier /etc/passwd et retourvez vos informations utilisateur
- utilisez cat sur le fichier /etc/group et retrouvez les informations de vos groupes

PERMISSIONS



PERMISSIONS - EFFETS

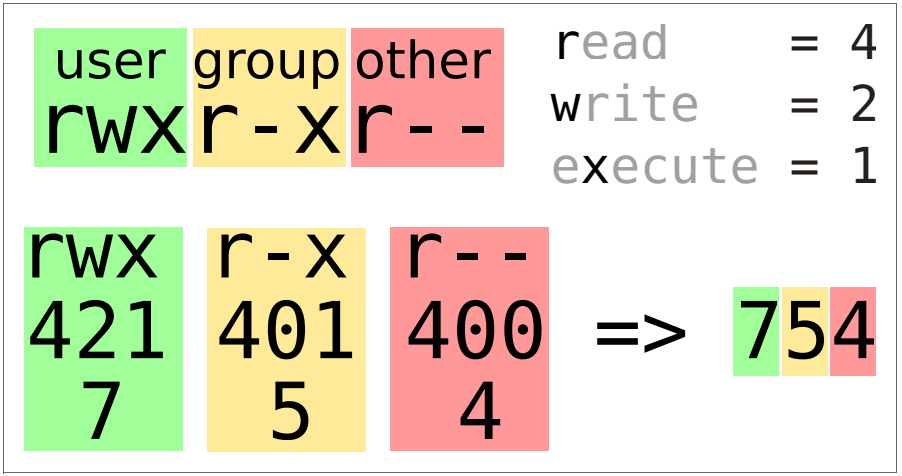
Fichier:

Read	Lire le contenu du fichier
Write	Modifer le contenu d'un fichier Warning en cas de suppression
eXecute	Exécuter le fichier (exécutable binaire ou script)

Répertoire:

Read	Lister les **noms** des fichiers inclus (mais pas les métadonnées)
Write	Créer, renommer ou détruire les fichiers inclus
eXecute	"Ouvrir" le répertoire, voir les métadonnées, accèder au contenu des fichiers (mais pas aux noms), exécuter les fichiers exécutables.

PERMISSIONS - OCTAL ET CHMOD



La commande **chmod** permet de changer les droits des fichiers:

```
$ chmod 754 monficher.c
```

PERMISSIONS - EXERCICES

- regarder le contenu du fichier bird00.jpg
- retirer les droits de lecture utilisateur du ficher images/bird00.jpg
- essayer de regarder le contenu du fichier bird00.jpg
- retirer les droits d'écriture utilisateur du ficher images/bird00.jpg
- Vérifier les changements de droit effectués
- Retirer les droits d'execution du dossier images
- Lister le contenu du dossier images avec `ls -l`
- Redonner les droits d'exécution au dossier

CRÉER ET SUPPRIMER LES FICHIERS/DOSSIERS

CRÉATION ET DÉPLACEMENT

touch ***file***	Crée un fichier vide si le fichier n'existe pas, sinon met à jour sa date de modification
mkdir ***folder***	Crée le dossier spécifié
mv ***source [sources] dest***	Déplace les sources vers la destination si la destination est un dossier, sinon renomme le fichier source en dest (possible changement de dossier)
cp ***source [sources] dest***	Copie les sources vers la destination (la destination peut être un fichier ou un dossier)

CRÉATION ET DÉPLACEMENTS - EXERCICES

touch

- créez un dossier backup
- copiez l'image images/bird01.jpg vers backup/mybird.jpg
- essayez de copier l'image images/bird00.jpg

mkdir

- copiez toutes les images de chien vers backup

mv

- créez un fichier image vide dans backup/myimage.png
- renommer ce fichier backup/myimage.jpg

cp

- déplacez en une commande les images de chats portant les numéro [00-09] et [90-99] vers le dossier backup

≡

SUPPRESSION

rm *fichier* Supprime le fichier indiqué

rmdir *dossier* Supprime le dossier indiqué

SUPPRESSION - EXERCICES

- tentez de supprimer le dossier backup, pourquoi n'est-ce pas possible?
- supprimez le fichier backup/mybird.jpg
- supprimez tout le contenu du dossier backup en une commande
- supprimez le dossier backup

- essayez de supprimer le fichier images/bird00.jpg. Refusez de supprimer le fichier. Que c'est-il passé, pourquoi ?
- ce fichier est donc supprimable, comment faire pour qu'il ne le soit plus ? Testez la solution, puis remmettez les droits à ceux d'origine

---

- copiez le dossier images vers images-backup, que ce passe-t-il ? Que veux dire l'option proposée ?

- effectuez la copie en utilisant l'option adéquate
- en regardant le manuel de la commande `rm`, supprimez le dossier images-backup et son contenu en une commande

LE CAUCHEMAR

```
$ rm -rf ~/*.txt
$ rm -rf ~/* .txt
```

EDITER UN FICHIER SUR UNE INTERFACE TEXTE

Plusieurs éditeur shell sont disponibles, on peut mentionner:

nano Editeur léger, le plus facile a appréhender, les raccoucis clavier sont indiqués en bas

vi Editeur historique, forte courbe d'apprentissage

vim Nouvelle version de vi avec de nombreux plugins disponibles

emacs Editeur historique, forte courbe d'apprentissage, plus qu'un éditeur, il **fait même le café**

≡

EDITER UN FICHIER

- Créer un nouveaux fichier contenant la liste de vos courses en utilisant un éditeur
- Vérifier le contenu du fichier en utilisant une commande déjà vue (autre que l'éditeur)

REDIRECTION DE FLUX ET PIPES

FLUX DE SORTIE

Le flux de sortie d'un programme est dirigée par défaut sur le terminal
La commande **`echo`** affiche les paramètres sur la sortie standard

```
$ echo ce cours est vraiment génial
ce cours est vraiment génial

$ echo "ce cours est vraiment génial"
ce cours est vraiment génial
```

Ce flux peut être redirigé vers un fichier avec `'>'` :

```
$ echo ce cours est vraiment génial > /tmp/qualite_du_cours.txt
```

FLUX DE SORTIE - EXERCICES

- stockez la liste des fichiers du dossier images, ainsi que leurs attributs, dans un fichier nommé result
- copiez un fichier vers le fichier result sans utiliser la commande `cp`
- regarder le contenu de result

- stockez la liste des fichiers du dossier images, ainsi que leurs attributs, dans un fichier nommé result
- **En utilisant '>>' au lieu de '>'** copiez un fichier vers le fichier result sans utiliser la commande `cp`
- regarder le contenu de result, que concluez vous ?

FLUX D'ENTRÉE

Le flux d'entrée d'un programme est par défaut associé au clavier du terminal
La plus part des commandes nécéssitant une entrée en paramètre (par exemple un nom de fichier) utilisent l'entrée standard si le parmètre n'est pas donné

```
$ cat               # va attendre et répeter les entrées utilisateur
$ cat > livre.txt   # va concatener tous ce qui est tapé dans livre.txt -> UN NOUVEL
EDITEUR :-) !
```

Un fichier peut-être redirigé vers ce flux d'entrée en utilisant `'<'` :

```
$ cat < livre.txt   # l'entrée standard est le fichier livre
$ cat livre.txt     # le programme lit depuis le fichier livre, pas depuis l'entrée
standard
$ # LE RESULTAT EST LE MEME
```

`Ctrl+D` permet d'intérrompre le flux d'entrée

FLUX D'ERREUR

Les message d'erreur des programme ne sont pas redirigées vers le flux de sortie mais vers un flux d'erreur
Par défaut le flux d'erreur est redirigé vers le terminal
Ce flux peut être redirigé vers un fichier avec `'2>'` :

```
$ ls non_existant_folder > /tmp/error.txt   # produit une erreur sur le terminal et un fichier vide
$ ls non_existant_folder 2> /tmp/error.txt  # produit un fichier contenant l'erreure
```

FLUX D'ERREUR - EXERCICES

- Utilisez la commande `ls -R /tmp` pour lister récursivement le contenu de /tmp
- Utilisez la même commande mais redirigez la sortie standard et l'erreur standard vers deux fichiers différents
- Observez ces fichiers

FLUX STANDARD DANS LES LANGAGES DE PROGRAMMATION

	C	*C++*	*Java*	*Python*	*Ruby*
Entrée	stdin	std::cin	System.in	sys.stdin	$stdin
Sortie	stdput	std::cout	System.out	sys.stdout	$stdout
Erreur	stderr	std::cerr	System.err	sys.stderr	$stderr

Exemple:

```
System.out.println( "Hello world!" );
```

```
$ java HelloWorld > hello.txt
```

PIPES OU TUBES

Les pipes permettent de rediriger la sortie standard d'un programme vers l'entrée standard d'un autre
On utilise le symbole '|' entre deux commandes pour les utiliser:

```
$ less file            # affiche le contenu du fichier file avec scrolling
$ ls -R /tmp | less    # affiche la sortie de ls AVEC scrolling
```

Combinés à la phylosophie UNIX, les pipes sont au coeur de la puissante des interfaces en ligne de commande

GREP

grep *fichier pattern*	cherche et affiche les lignes du fichier qui contienne un pattern (e.g. un mot)
grep *pattern*	cherche le pattern dans l'entrée standard et affiche les lignes le contenant

PIPES ET TUBES - EXERCICES

Quelques commandes utiles pour ces exercices (voir le manuel):
sort, uniq, grep, head, du

Utilisez la commande `ls -l` pour lister les fichiers de chats du dossier `images` mais sans ajouter d'arguments à `ls`

Le dossier `mails` contients des listes d'adresses. A partir de ces fichiers obtenez une seule liste des membres de HEPIA, triée par ordre alphabétique, sans doublons

---

Lister les 2 plus gros fichiers contenus dans le dossier `mails`

SYSTÈME DE FICHIERS

SYSTÈME DE FICHIERS (1)

- Manière d'organiser les données sur le disque:
  - Noms
  - Répertoires
  - Permissions
  - Metadonnées
  - Maintien de l'intégrité
  - …

SYSTÈMES DE FICHIERS (2)

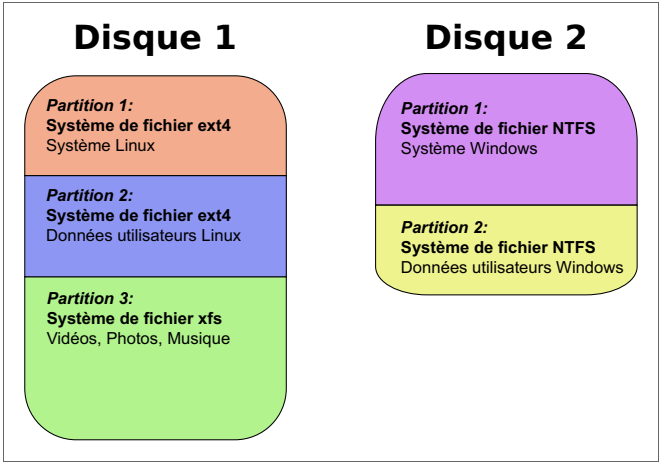
Il existe plusieurs types de systèmes de fichiers. Par exemple:

VFAT	Ancien système de fichiers de Windows, très utilisé sur les clé-usb, lecteurs MP3, etc.
NTFS	Système de fichiers récent sur Windows.
ext4	Système de fichiers par défaut sous beaucoup de distributions GNU/Linux
ISO9660	Système de fichiers des CD-ROMs
UDF	Système de fichiers des DVDs
HFS+	Système de fichiers sur MacOSX
NFS	Système de fichiers **réseau** sous Unix
IPFS	Interplanetary file system - système de fichiers distribué et décentralisé.

ARBORESCENCE (1)

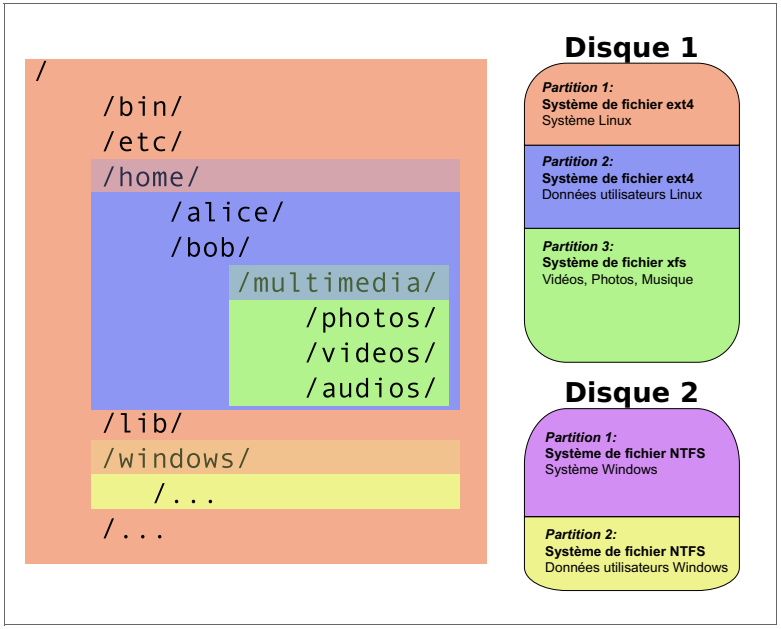
- Tous les systèmes de fichiers sont "montés" sur une seule et même arborescence.
- Seul `root` peut monter/démonter des systèmes de fichiers (par défaut)
- La plupart des installations GNU/Linux actuelles, montent automatiquement les disques externes (CD, DVD, usb)

ARBORESCENCE (2)



Partitionnement des espaces

ARBORESCENCE (3)



Montage des systèmes de fichiers dans l'arborescence

≡

ARBORESCENCE UNIX STANDARD

`/bin/`	Principaux exécutables
`/boot/`	Fichiers de démarrage
`/dev/`	*Périphériques*
`/etc`	Configuration système
`/home/`	Données utilisateurs
`/lib/`	Librairies système
`/media/`	Montage des périphériques de stockage
`/mnt/`	Point de montage manuel
`/proc/`	*Processus*
`/root/`	Répertoire perso du superutilisateur
`/sbin/`	Exécutables pour la maintenance
`/sys/`	*Informations système*
`/usr/`	Applications et librairies utilisateurs
`/var/`	Données variables (logs, spool, mail…)

EVERYTHING IS A FILE

- Plusieurs concepts sont représentés par des **fichiers synthétiques**.
- Par exemple:
  - Les périphériques sont visibles dans `/dev/`
  - Les processus (et certaines info système) sont visibles dans `/proc/`
  - Informations sur les périphériques dans `/sys/`

EXEMPLES D'UTILISATION DE `/dev/`

Faire une image ISO à partir d'un CD

```
$ dd if=/dev/cdrom of=~/image.iso
```

Remplir une partition de données pseudo-aléatoires

```
$ dd if=/dev/urandom of=/dev/sdb1 bs=1M
```

EXERCICES - SYSTÈME DE FICHIERS

- Grâce à la commande `mount`, trouvez où sont montés les systèmes de fichier de type ext4, fat et tmpfs ?
- sachant qu'il existe deux types de périphériques, bloc (resp. caractère) indiqué dans le mode du fichier par le symbole b (resp. c); listez les périphériques de type bloc.
- grâce au device `zero`, remplissez un fichier de 1MB de zeros (attention la commande peu donner lieu à de GROS fichier si mal utilisée).
- grâce au device `null`, lister recursivement le contenu de /tmp sans que le terminal n'affiche de sortie d'erreur.

PROCESSUS

QU'EST-CE QU'UN PROCESSUS ?

- Instance d'un programme en cours d'execution
- Possède:
  - Son code en langage machine
  - Des Segments de mémoire (e.g. code, données, pile, etc.)
  - Des descripteurs de fichiers (e.g. les sortie et entrées standard)
  - Un propriétaire et un ensemble de permissions
  - Un état à un moment donné (e.g. le pointeur d'instruction)
- Un coeur de processeur ne peut exécuter qu'un processus à la fois
- Le système d'exploitation peut alterner l'execution de plusieurs processus (*multi-tasking*)

PROCESSUS UNIX

La commande `ps` permet de lister les processus

Sous unix, tous les processus ont:

- un identifiant numérique (*PID*)
- un processus parent (sauf le processus 0)

PS - EXERCICES

- Tapez la commande `ps`. Qu'est ce que les processus affichés ont en commun ?
- Taper la commande `ps -ef` (voir man). Trouvez un processus qui est l'enfant d'un autre.
- Vérifiez avec la commande `pstree` que c'est bien le cas. Quel est le PID du processus initial (le plus ancien parent de tous les processus)?
- Trouvez un moyen de lister tous les processus vous appartenant.

UTILISATION DU CPU (`top`)

La commande `top` permet de voir quels processus occupent le(s) CPU(s).

SIGNAUX

- Les signaux permettent une communication limitée entre les processus.
- La commande `kill -l` permet d'afficher les différent signaux
- Quelques signaux utiles:

Nom	Action	Numero
SIGINT	Interrompt le processus	2
SIGKILL	Tue le processus immédiatement	9
SIGTERM	Demande la terminaison (propre)	15
SIGTSTP	Suspension	20
SIGCONT	Active un processus suspendu	18
SIGUSR1	Signal utilisateur 1	10
SIGUSR2	Signal utilisateur 2	12

ENVOYER UN SIGNAL (`kill`)

```
kill -signal [pids]...
```

`kill` permet d'envoyer un signal à un processus

```
$ kill -SIGKILL 2111 2120
$ kill -15 2111
```

GESTION DES SIGNAUX PAR LES PROCESSUS

- Tous les processus peuvent gérer (intercepter) un signal.
- Par exemple:
  - Sauver l'état avant de quitter lorsque `SIGTERM` est reçu
  - Redémarer le processus lorsque `SIGINT` est reçu
- Le signal `SIGKILL` ne peut être intercepté

SIGNAUX DEPUIS LE SHELL

Les raccourcis claviers suivant permettent de lancer un signal à un processus du shell:

ctrl+C Envoie `SIGINT`

---

ctrl+Z Envoie `SIGTSTP`

Attention ctl+D n'est pas un signal mais envoie "End-Of-File" (EOF) sur l'entrée standard

EXERCICE SUR LES SIGNAUX

- Lancez l'éditeur de texte grahique `gedit` depuis le terminal
- Intérrompez l'éditeur en lui envoyant SIGINT de deux manières différentes

---

- Lancer un éditeur de texte dans le shell (i.e. non grahique) et taper du text sans sauvegarder
- Depuis un autre terminal, terminez le programme avec lui envoyant le signal SIGTERM
- Recommencez mais cette fois interrompez le programme avec lui envoyant le signal SIGINT
- Finalement terminez le programme en lui envoyant le signal SIGKILL
- Quels sont les différences de comportement ?

JOBS

- Par défaut, tout processus lancé depuis le shell tourne à l'avant-plan (*foreground*)
- Un processus en avant-plan bloque le shell tant qu'il n'est pas fini.
- On peu aussi lancer un processus en arrière-plan (*background*):
  - Ajouter un & après la commande:

    ```
    $ gedit &
    ```

  - Interrompre un processus:
    - Utilisez `ctrl-Z` pour interrompre le processus
    - Utilisez la commande `bg` pour relancer un processus interrompu en arrière-plan
    - On peut aussi utiliser `fg` pour remettre le dernier processus interrompu en avant-plan

EXERCICE SUR LES JOBS

- Lancez l'éditeur de texte grahique `gedit` depuis le terminal, cette fois en arrière plan
- Mettre le processus en pause, tester sa réactivité, puis le relancer en appellant `kill`

- Editez un fichier en utilisant `vim`
- Interrompre le processus avec ctrl-Z
- Remettre le processus en avant plan

CONNEXIONS A DISTANCE: SSH (SECURE SHELL)

UTILITÉ

- travail à distance comme si l'on étais sur un terminal de la machine
- accéder à une machine qui ne possède pas de terminal
- transférer des fichiers de manière sécurisée
- connexion sécurisée pour des clients X distants (pour sur-simplifier: interface graphique distante)

COMMENT CA MARCHE ?

1. se connecter avec la commande suivante:

```
$ ssh login@mon.server.com
```

2. vérifier que le fingerprint du serveur correspond bien à celui attendu
3. entrer son mot de passe
4. on peut terminer la connexion en mettant fin au shell associé (`exit` ou Ctrl+D)

SSH - EXERCICES

- connectez-vous au serveur `10.136.26.133`, dont le fingerprint est:

```
SHA256:dMjJfl/CweskwJK6g+L4vtCh0CgYsYYR7R8aoHISA2U
```

- une fois connecté, lister le contenu de votre dossier home sur ce serveur
- editez un nouveau fichier dans un dossier nommé `votre_nom_prenom` en utilisant un editeur de texte
- terminer la connexion SSH

SCP

La commande SCP permet de copier des fichier d'un serveur vers la machine locale et vice-versa

```
$ scp login@mon.server.com:/path/to/my/file /path/to/destination
```

```
$ scp /path/to/my/file login@mon.server.com:/path/to/destination
```

Les formats de chemin habituels sont disponibles (*, ?, ~, etc.)

SCP - EXERCICES

- copier le fichier du dossier `votre_nom_prenom` localement
- modifier ce fichier localement
- copier le nouveau fichier dans le dossier distant `votre_nom_prenom`

AUTHENTIFICATION PAR CLEF PUBLIQUE

PRINCIPE

Vous avez a disposition une clef publique et une clef privée:

- la clef privée VOUS identifie sur UNE MACHINE (une clef par utilisateur et par machine)
- la clef publique est associée à la clef privée, et elle sera disbribuée à toute entité avec laquelle vous voulez communiquer
- la clef publique est utilisée pour chiffrer un message qui vous est envoyé, **seul la clef privée permet de le décoder**

Il n'est donc pas grave de distribuer sa clef publique, en revanche **la clef privée est personelle et ne doit jamais quitter la machine**

AUTHENTIFICATION PAR CLEF PUBLIQUE

PROTOCOLE

- client: *bonjour je me présente, voici ma clef publique*
- server: *ok elle fait bien partie de la liste des clefs que je connais* (il faudra donc fournir sa clef au serveur avant authentification)
- server: *voici un message chiffré par votre clef publique, serez-vous capable de le déchiffrer ?*
- client: *oui je peu, j'ai la clef privée et voici le message déchiffré*
- server: *ok je confime que vous avez la bonne clef privée car le message est bien déchiffré, vous êtes authentifié*

AUTHENTIFICATION PAR CLEF PUBLIQUE

Utilisée de plus en plus fréquement par des services en ligne (e.g. github, gitlab, etc.)

AVANTAGES

- facilité de connexion et automatisation (on peu eventuellement se passer d'un mot de passe)
- plus sécurisé qu'un mot de passe car aucune information secrète n'est transférée

INCONVENIENTS

- il est parfois difficile de gérer un grands nombre de clefs publiques/privées
- demande quelques connaissances techniques

AUTHENTIFICATION PAR CLEF PUBLIQUE

EN PRATIQUE

Créer une paire de clef publique / privée (de préférence avec passphrase)

```
$ ssh-keygen
```

Plusieurs questions vous sont posées.
Copier la clef publique sur le serveur

```
$ ssh-copy-id login@mon.server.com
```

Se connecter au serveur

```
$ ssh login@mon.server.com
```