Shell - ssh
Partie 1/2

Pauline POMMERET

3 novembre 2015

Plan

ssh, *secure shell*

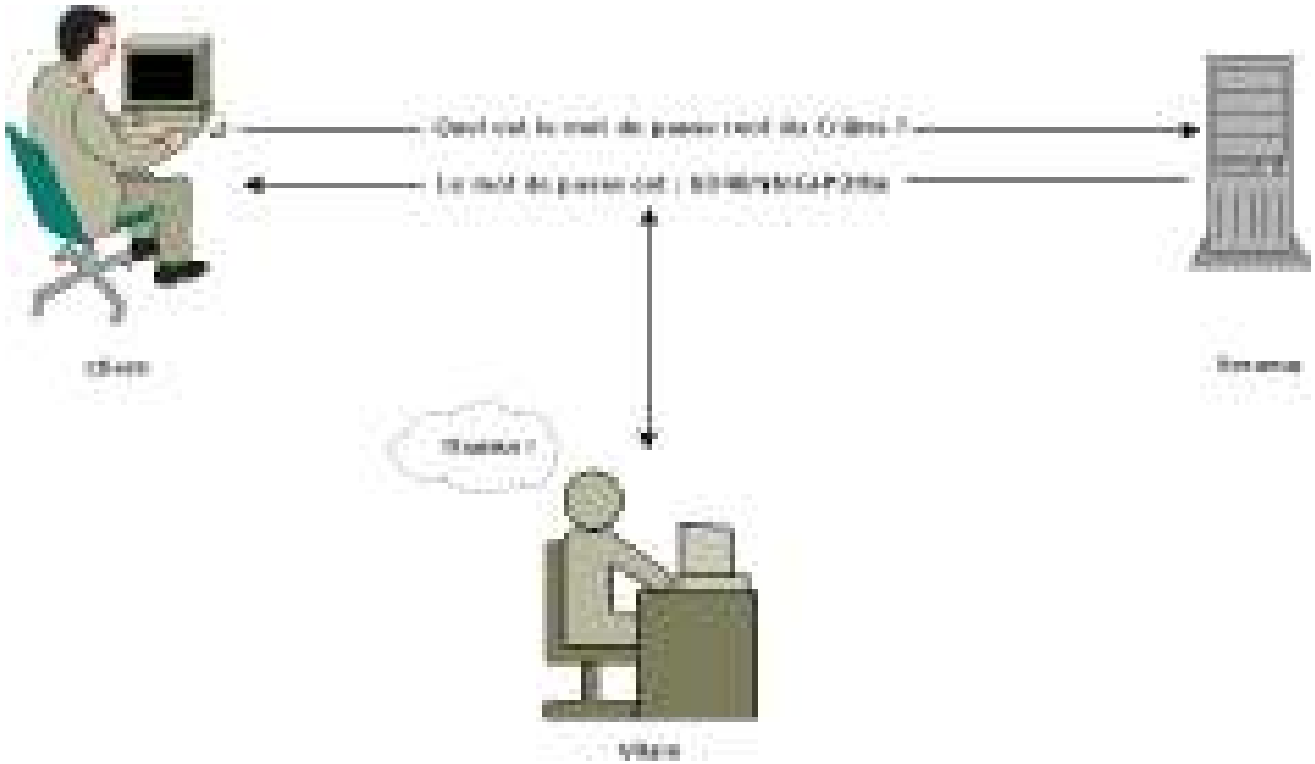
ssh

ssh désigne :

- une commande ;
- un protocole de communication sécurisée (les données sont chiffrées) entre un client et un serveur distant.

Le ssh permet de se connecter à une machine distante et d'y travailler.

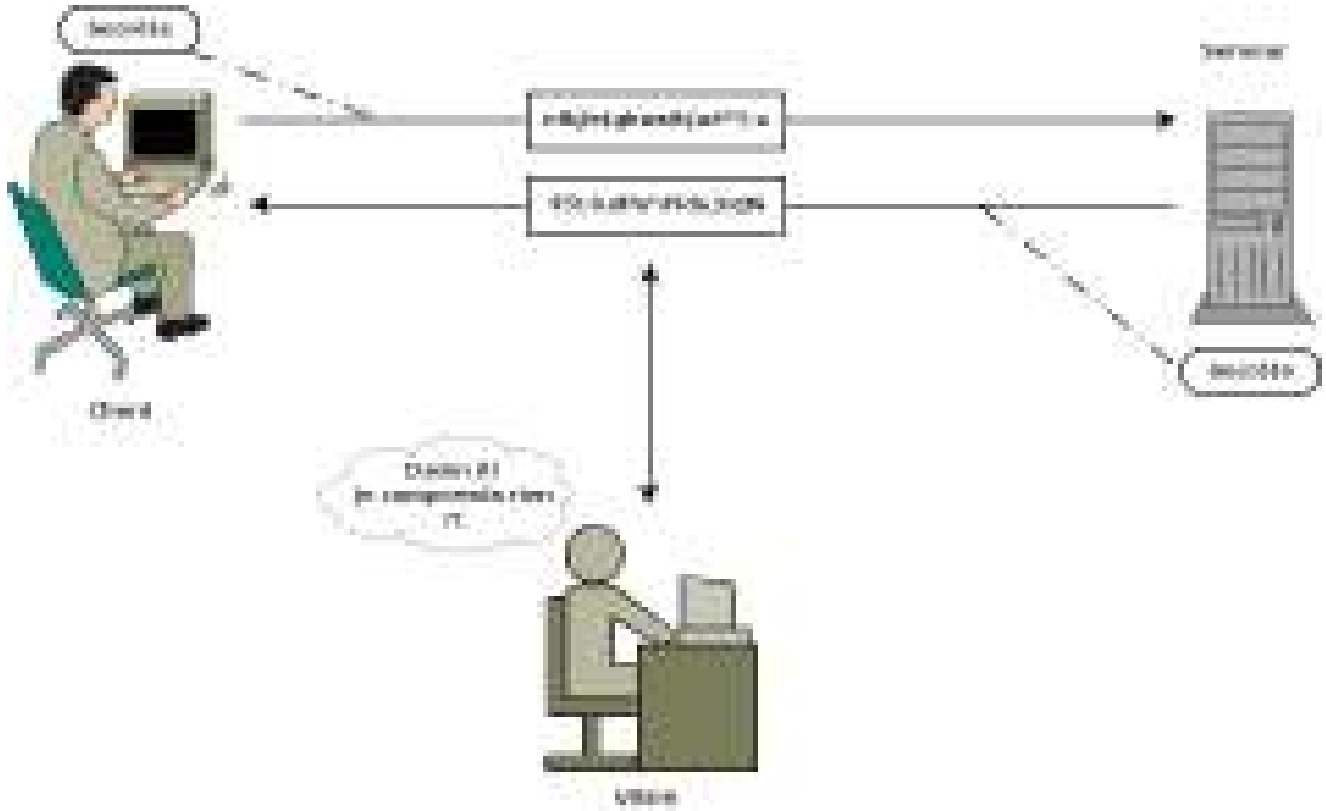
Communication avec `telnet`



→ `telnet` est simple mais dangereux

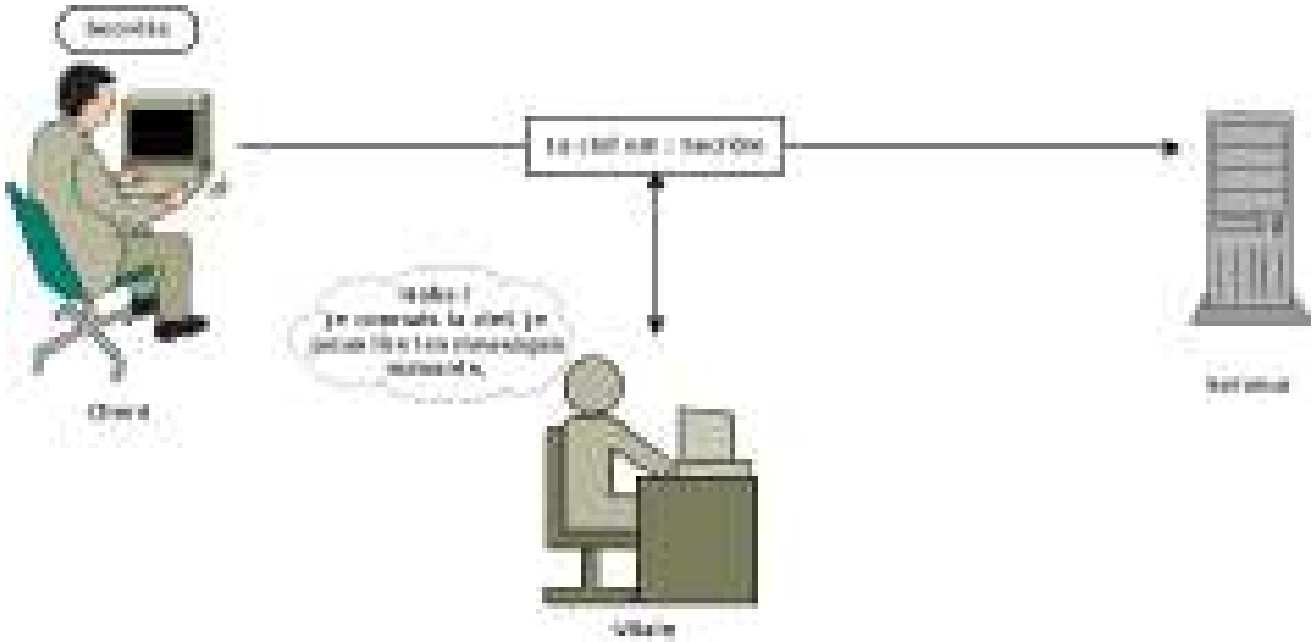
→ nécessité de chiffrer ses données

Chiffrement symétrique (1/2)



→ parfait le chiffrement symétrique, non ?

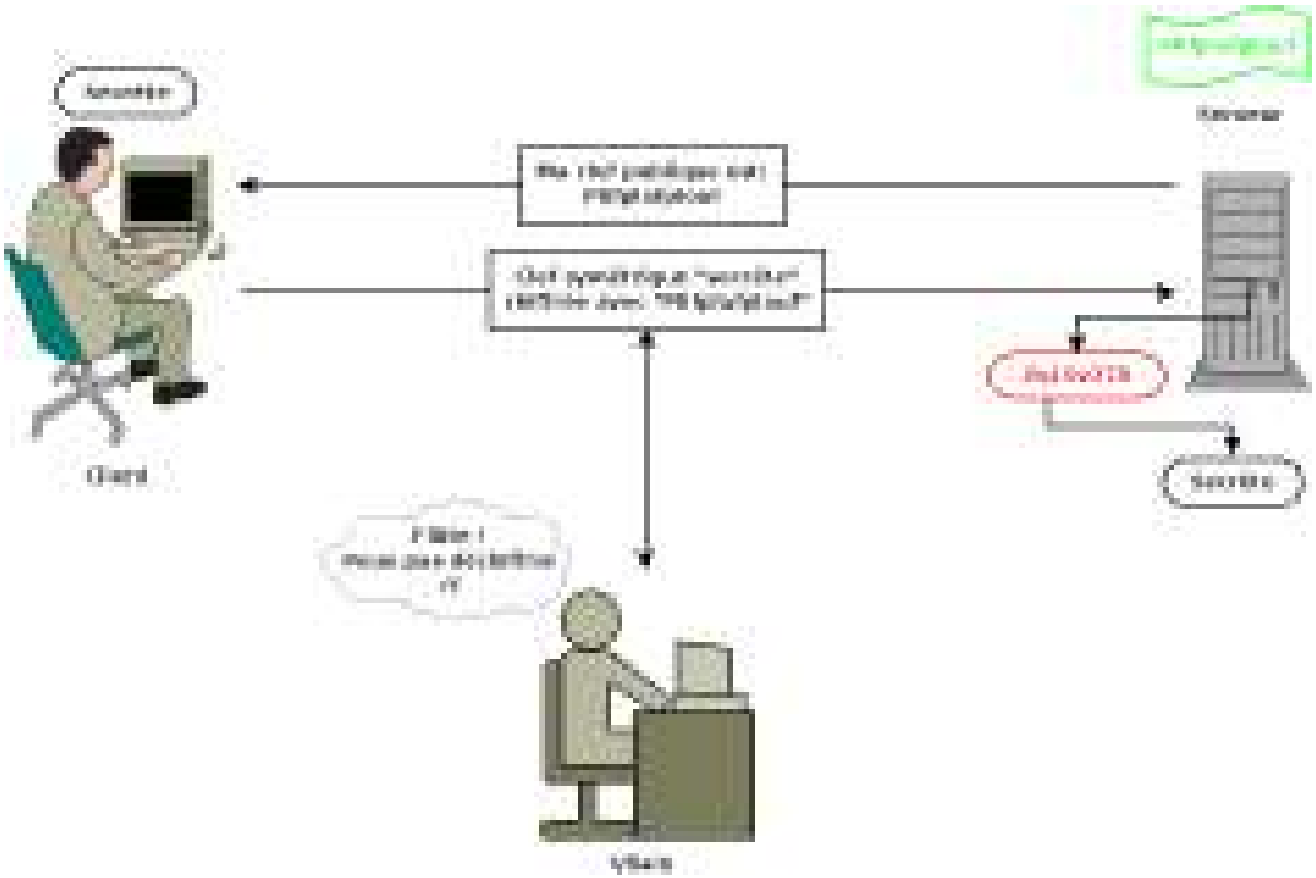
Chiffrement symétrique (2/2)



→ nécessité de transmettre *discrètement* la clef

→ difficilement faisable en pratique

Chiffrement asymétrique (1/2)



→ parfait !

Chiffrement asymétrique (2/2)

Pour le chiffrement asymétrique, il faut :

- une clef **publique** qui sert à chiffrer,
- une clef **privée** qui sert à déchiffrer.

Le chiffrement asymétrique consomme plus de ressources, aussi il n'est utilisé qu'au *début de la communication* pour permettre l'échange sécurisé de la clef symétrique.

Établissement d'une connexion `ssh`

1. Le serveur envoie sa clef publique au client. Celui-ci vérifie que c'est la clef du serveur (si déjà reçue).
2. Le client génère une clef secrète et l'envoie au serveur en la chiffrant avec la clef publique reçue. (*chiffrement asymétrique*)
3. Le serveur chiffre un message standard avec la clef secrète et l'envoie au client qui le déchiffre. (*prouve que le serveur est bien le vrai serveur*)
4. Établissement d'un canal sécurisé grâce à la clef secrète commune. (*chiffrement symétrique*)
5. Le client peut alors envoyer le login et son mot de passe de l'utilisateur pour vérification.

Première connexion à `zamok`

Pour se connecter en `ssh` sur `zamok` :

```
user@host: $ ssh loginCrans@zamok.crans.org
```

damn it !

```
The authenticity of host 'zamok.crans.org (138.231.136.1)' can't be established.
ECDSA key fingerprint is SHA256:NuNyHJiEHVpKWWygwzx6JP1sn9hcn3iHHmSR+N3JltQ.
+---[ECDSA 521]---+
|    o=+          |
| + o.*o. . .   . |
|. O +.o . o   . E|
| + = . o . o + o |
|. . + o S + . *  |
| . . = * O o .   |
|    . * = * .    |
|       o o o     |
|          .      |
+----[SHA256]-----+
Are you sure you want to continue connecting (yes/no)?
```

Si vous tapez `yes` comme ça pouf, vous risquez de m'énerver très fort.

C'est quoi un fingerprint `ssh` ?

- une version courte de la clef publique du serveur

→ permet de *vérifier* que c'est la vraie clef du serveur

→ permet de se prémunir d'une attaque *man in the middle*

→ nécessité de **vérifier** le fpr

```
pommeret@zamok $ for file in /etc/ssh/*sa_key.pub; do ssh-keygen -lf $file; done
1024 50:f5:72:cd:2d:2b:be:e3:5c:a7:17:28:4c:6d:6c:70
     /etc/ssh/ssh_host_dsa_key.pub (DSA)
521  63:e0:64:78:56:ff:e3:4b:15:17:a8:2f:43:6c:d2:cf
     /etc/ssh/ssh_host_ecdsa_key.pub (ECDSA)
4096 b4:85:bc:47:dd:ee:d0:80:3c:20:ee:f0:de:d3:8e:3a
     /etc/ssh/ssh_host_rsa_key.pub (RSA)
```

Quand doit-on vérifier le fingerprint `ssh` ?

- à chaque première connexion
- à chaque fois que la commande `ssh` prévient que le fingerprint à changé : ça peut être totalement légitime ou une attaque

Vérifier le fingerprint via le DNS

Pour vérifier le fingerprint à partir des entrées SSHFP du resolver DNSSEC :

```
user@host: $ ssh -o VerifyHostKeyDNS=yes
                loginCrans@zamok.crans.org
```

Récupérer une configuration

Un `.ssh/config` commenté et expliqué a été écrit avec amour par des membres actifs du Crans :

```
user@host $ git clone
  git@gitlab.crans.org:membres-actifs/fichiers_configuration.git
```

Sinon, consulter la page `wiki.crans.org/VieCrans/FichiersConfiguration`.

Authentification par clefs

Au lieu de s'authentifier par mot de passe, il est possible de s'authentifier par un couple de clef privée/publique, c'est-à-dire par cryptographie asymétrique.

+ plus sûr puisqu'il faut que l'attaquant se procure et la clef privée et la passphrase pour pouvoir s'authentifier
+ l'utilisateur a de meilleures garanties que le serveur auquel il se connecte est bien celui auquel il souhaite se connecter
+ possibilité de « déverouiller » la clef pour une période de temps donné et de pouvoir accéder sans taper de mot de passe aux serveurs (confort)
− si la clef n'est pas protégée par une passphrase, cela retire une couche de sécurité

Génération de clefs

Pour générer un couple de clef RSA, il faut exécuter :

```
user@host: $ ssh-keygen -t rsa
```

Les clefs sont stockées :

- dans `~/.ssh/id_rsa` pour la clef privée (permission 600)
- dans `~/.ssh/id_rsa.pub` pour la clef privée (permission 644)

Passphrase

Lors de la création de la clef privée, OpenSSH demande l'entrée d'une *passphrase* qui servira à chiffrer la clef privée. Il *faut* en mettre une.

La *passphrase* sera demandée à chaque utilisation de la clef privée.

Ajouter temporairemnt la clef au `ssh-agent` (programme qui conserve en mémoire les clefs privées)

```
user@host $ ssh-add -t 7200 ~/.ssh/id_rsa
```

Changer la *passphrase*

```
user@host: $ ssh-keygen -p
```

Autoriser la clef publique

→ copier la clef *publique* sur le serveur distant auquel on souhaite pouvoir s'authentifier par clef

→ l'ajouter à son `~/.ssh/authorized_keys`

```
user@host: $ ssh-copy-id -i ~/.ssh/id_rsa.pub
             loginCrans@zamok.crans.org
```

Transfert du serveur graphique

Si on veut utiliser le serveur graphique d'une machine distante, c'est possible ! Il suffit de faire :

```
user@host: $ ssh -X loginCrans@vo.crans.org
```

On peut alors lancer des programmes comme `iceweasel`, `inkscape`, ...

Plan

Qu'est ce qu'un shell ?

shell

Couche logicielle qui fournit l'interface utilisateur d'un système d'exploitation. Il correspond à la couche la plus externe du système d'exploitation.

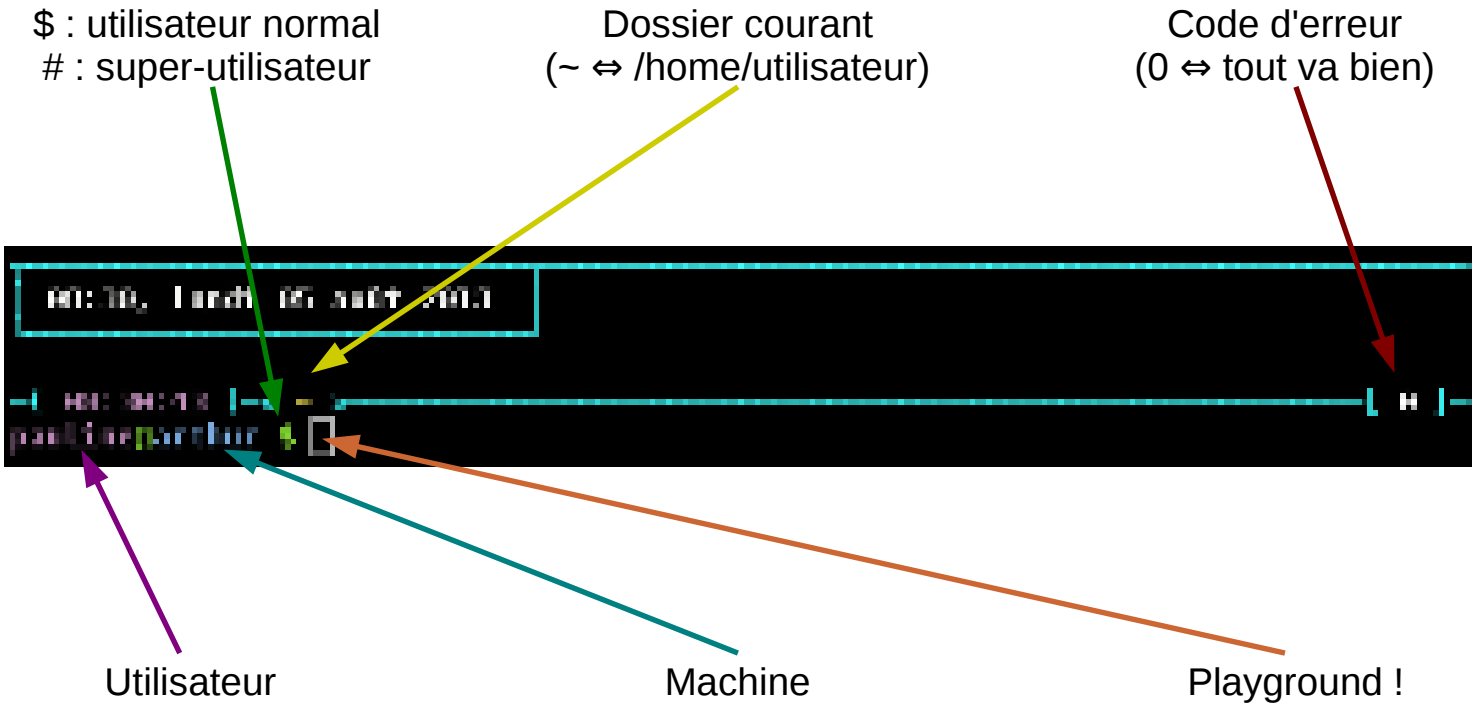
- interface en ligne de commande dite *CLI*, *Command Line Interface*, où l'utilisateur lance des instructions sous forme de texte ;
- interface graphique dite *GUI*, *Graphical User Interface*, où l'utilisateur utilise sa souris.

Les différentes espèces de shell Unix

Il existe de nombreux shell :

- Shell de Stephen BOURNE
  - BOURNE shell (`/bin/sh`) : ancien shell par défaut, souvent shell par défaut pour `root` ;
  - BOURNE-Again shell (`/bin/bash`) : interprète par défaut (par défaut pour Mac OS X, Cygwin) ;
- *C shell* (`/bin/csh`) : évolution du shell `sh` avec une syntaxe plus proche du C ;
- Z shell (`/usr/bin/zsh`) : sorte de BOURNE shell étendu reprenant les fonctionnalités les plus pratiques de `bash`, `ksh` et `csh`, par défaut au CR@NS.

Présentation d'un terminal



Récupérer un fichier de customisation déjà écrit

Des gentils cranseux ont publié des fichiers de configuration pour vous éviter de les écrire à la main :

```
user@host $ git clone
  git@gitlab.crans.org:membres-actifs/fichiers_configuration.git
```

Sinon, consulter la page `wiki.crans.org/VieCrans/FichiersConfiguration`.

Caractères de contrôle clavier essentiels

<tab>

<tab> permet de faire de la « tab-complétion » c'est-à-dire de compléter par exemple les noms de commandes, de fichiers, les chemins. SUPER UTILE ! !

^C

^C interrompt un processus attaché au terminal (SIGINT, signal 11)

^D

^D renvoie un caractère de fin de fichier (caractère ASCII 026), si le shell lit, il termine

Format d'une commande shell

- Une commande simple est une séquence de mots séparés par un séparateur blanc (une espace).
- Le premier mot désigne le nom de la commmande à exécuter, les mots suivants sont passés en arguments de la commande.
- La valeur retournée par la commande est celle de son code de retour.

Plan

echo

Commande

`echo` est une commande qui affiche une ligne de texte.

```
user@host $ echo "Hello world!"
```

On peut s'en servir pour retrouver le *shell* dans lequel on habite :

```
user@host $ echo $SHELL
```

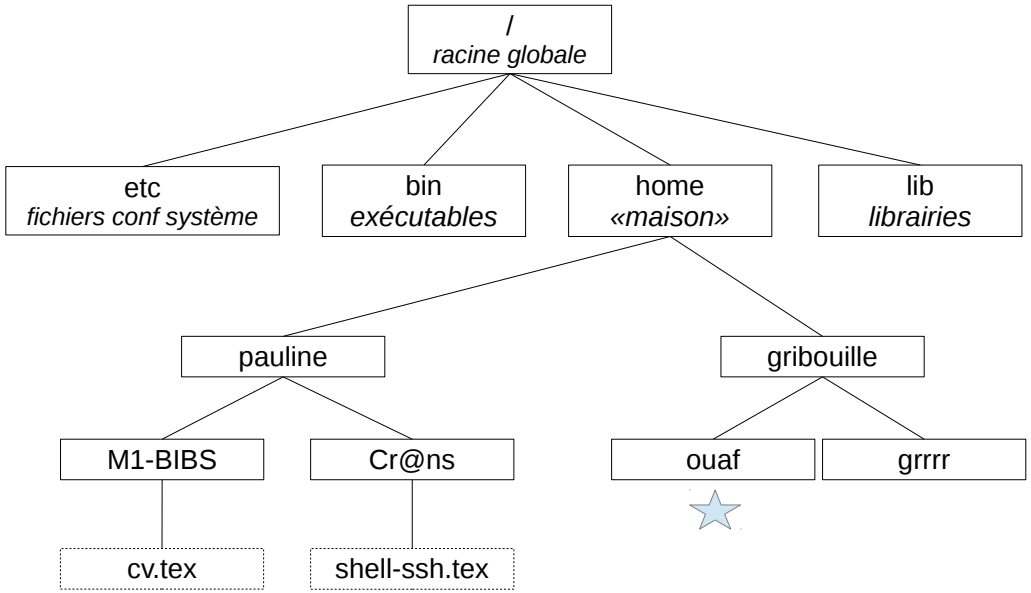
man

Commande

`man` permet d'afficher l'aide d'une commande"

```
user@host $ man echo
```

Les fichiers sont rangés dans une arborescence



Chemin absolu
/home/pauline/M1-BIBS/cv.tex

Chemin relatif depuis
../../pauline/M1-BIBS/cv.tex

Chemin relatif et chemin absolu

Le chemin absolu est la succession des répertoires à parcourir depuis la **racine** pour accéder au fichier spécifié.

Le chemin relatif est la succession des répertoires à parcourir depuis le **répertoire courant** pour accéder au fichier spécifié.

On désigne par `..` le répertoire parent.

`cd`, *change directory*

Commande

`cd` est une commande qui permet de naviguer dans l'arborescence des fichiers, connaissant l'emplacement du dossier que l'on cherche.

```
user@host: $ cd fichiers_configuration
```

Que ce soit pour le chemin absolu ou pour le chemin relatif, il faut connaître l'arborescence des fichiers. Il ne faut surtout pas oublier d'utiliser la *tab-complétion*, ça fait gagner beaucoup de temps.

`pwd`, *print working directory*

Commande

`pwd` est une commande qui permet d'afficher le dossier courant, d'afficher où l'on est.

```
user@host $ pwd
```

Cette commande est très pratique lorsque l'on ne dispose pas de `.bashrc` ou `.zshrc` *user-friendly* qui renouvelle l'affichage de la localisation à chaque retour de prompt.

`ls`, *lists segments*

Commande

`ls` est une commande qui permet d'afficher le contenu d'un répertoire.

```
user@host $ ls
```

les options utiles

- `ls -a` affiche tous les fichiers et dossiers du répertoires (mêmes les cachés)
- `ls -l` affiche la liste des fichiers et des dossiers, avec leurs dates de dernière modification, leurs tailles, les utilisateurs propriétaires, groupe propriétaire et les droits.
- `ls -lh` même chose que précédemment, avec les tailles en format *human readable*.

mv, *move*

Commande

`mv` sert à déplacer ou renommer des fichiers.

```
user@host $ mv .umaskrc ../what_is_that_file
user@host $ mv fichiers_configuration myconfig
```

les options

- `mv -i` *interactive* demande pour chaque fichier/répertoire s'il peut déplacer le fichier/répertoire
- `mv -u` *update* demande à `mv` de ne pas supprimer le fichier si la date de modification est la même ou plus récente que son remplaçant

cp, *copy*

Commande

`cp` permet de copier un fichier ou un répertoire.

Copier un fichier dans un autre fichier :

```
user$host $ cp what_is_that_file .umaskrc
```

Copier un fichier dans un autre dossier :

```
user@host $ cp .umaskrc myconfig
```

les options

- `cp -i` avertit de l'existence d'un fichier du même nom et demande s'il peut remplacer son contenu.
- `cp -r` permet de copier de manière récursive l'ensemble d'un répertoire et de ses sous-répertoires.
- `cp -p` préserve toutes les informations comme le propriétaire, le groupe et la date de création.

`mkdir`, *make directory*

Commande

`mkdir` permet de créer un répertoire.

```
user@host: $ mkdir Playground
```

les options

- `mkdir -p` permet de créer une suite de répertoires :

```
user@host $ mkdir -p Playground/Crans/Seminaires/Shell-ssh/2015
```

touch

Commande

`touch` sert à modifier le *timestamp* d'un fichier.

```
user@host $ touch test
```

va créer le fichier `test` dans le dossier courant, s'il n'existe pas encore.

les options

- `touch -t STAMP` utilise `STAMP` au lieu du temps présent.
- `touch -r plop -B 5 test` fait paraître le fichier test 5 secondes plus vieux que le fichier plop.
- `touch -r plop -F 5 test` fait paraître le fichier test 5 secondes plus jeune que le fichier plop.
- `touch -m` modifie la date de dernière modification.

`rm`, *remove*

Commande

`rm` permet de supprimer un fichier.

```
user@host: $ rm test
```

les options

- `rm -i` permet de demander à l'utilisateur s'il veut vraiment effacer le fichier.
- `rm -d` permet de supprimer un répertoire qu'il soit plein ou nous (dangereux).
- `rm -r` permet de supprimer un répertoire et ses sous-répertoires (très dangereux).
- `rm -f` permet de supprimer les fichiers protégés en écriture et les répertoires sans demander de confirmation (vraiment très dangereux)

`cat`, *concatenate*

Commande

`cat` permet de concatener des fichiers ou de lire un fichier.

```
user@host $ cat ~/myconfig/.ssh/config
```

Permet d'afficher sur la sortie standard le contenu du fichier de configuration pour `ssh`.

```
user@host $ cat <file_1> <file_2>
```

Permet de concaténer les 2 fichiers.

les options

- `cat -n` permet de numéroter les lignes dans la sortie standard.

`less`, *less*

Commande

`less` lit au fur et à mesure le fichier qu'on lui donne et permet la navigation en amont et en aval.

```
user@host $ less ~/myconfig/.ssh/config
```

les options

- `less <entrée> /pattern <entrée>` permet de rechercher le *pattern* dans le fichier, en ayant son contexte.
- `less <entrée> /!pattern <entrée>` permet de rechercher les lignes ne contenant pas *pattern*.
- `^D` permet d'avancer de N lignes (par défaut, la moitié de la taille de l'écran).
- `^B` permet de reculer de N lignes (par défaut, la moitié de la taille de l'écran).

nano

Commande

`nano` est un éditeur de texte, natif sur Ubuntu et Debian.

```
user@host: $ nano <nom du fichier>
```

Permet d'éditer le fichier en question.

les raccourcis

- `^O` permet d'écrire le fichier *i.e.* de sauvegarder.
- `^X` permet de fermer le fichier.
- `^K` permet de couper les lignes.
- `^U` permet de coller les lignes.

Ce qui est pratique avec `nano`, c'est qu'il y a toujours une anti-sèche...

Il existe d'autres éditeurs de texte comme `vim` que l'on peut apprendre à dompter grâce à `vimtutor`.

`ln`, *link*

Commande

`ln` sert à faire des liens entre des fichiers

les options

- `ln -s` *symbolic* permet de créer un lien symbolique, comme un « raccourci »

```
user@host $ ln -s file1 file2
```

Permet de faire un lien depuis le fichier existant `file1` vers le fichier `file2`.

```
user@host $ ln -s ~/myconfig/.umaskrc ~
```

Permet de faire un lien depuis le fichier `.umaskrc` rangé dans `~/myconfig` vers `~`.

Plan

Obtenir un fichier de configuration pour son shell

1. Trouver quel genre de *shell* on utilise
2. Trouver un fichier rc correspondant au nom de son *shell*
3. Établir un lien symbolique entre ce fichier rc et son `$HOME`
4. Éxécuter
   `source <chemin vers le fichier rc>`

Obtenir un fichier d'alias de commande

1. Lire les 2 fichiers `_aliases`
2. Écrire un fichier `all_aliases` contenant les alias des 2 fichiers
3. Renommer le fichier avec le nom correspondant au *shell* utilisé
4. Établir un lien symbolique entre ce fichier et son `$HOME`

Configuration `ssh`

1. Se balader dans `myconfig/.ssh`
2. Lire le contenu du fichien `myconfig/.ssh/config`
3. Obéir aux instructions
4. Établir un lien symbolique entre ce fichier et son `$HOME`

wget

Commande

`wget` permet de télécharger des fichiers en ligne.

```
user@host $ wget
   perso.crans.org/pommeret/Seminaire/Shell/.irssi/config
```

Personnaliser la configuration d'`irssi`

1. Créer le dossier `$HOME/.irssi`
2. Modifier le fichier `config` téléchargé
3. Déplacer ce fichier dans le dossier `$HOME/.irssi`

```
user@zamok $ irc
```

screen

Commande

`screen` est un programme qui permet d'ouvrir plusieurs terminaux dans la console courante et de les récupérer après avoir quitté la console.

```
user@zamok $ screen -S irc
user@zamok $ irc
```

Pour détacher : `^A ^D`.

Pour rattacher :

```
user@zamok $ screen -dr irc
```

.procmailrc

`procmail` est un programme qui permet de trier ses mails dans différents dossiers dès leur arrivée dans la boîte mail. Le fichier de configuration à écrire peut être très chiant à écrire mais il y en a un tout prêt qui permet de trier tous les mails Crans, l'immense majorité des mailing-lists Crans ainsi que de trier automatiquement dans des dossiers les mails reçus sur une mailing-list (même si elle n'est pas Crans).

Pour le mettre en place, consulter https://wiki.crans.org/VieCrans/FichiersConfiguration#procmail.

Écrire `"| exec /usr/bin/procmail"` dans son `$HOME/.forward`

Plan

Pourquoi il faut que vous veniez à la 2ème partie

- plein de nouvelles commandes
- apprendre à faire fonctionner les commandes les unes avec les autres
- des scripts trop cool
- maintenant que vous avez un `procmail`, vos mails sont triés il faut que vous alliez vous abonner manuellement à vos dossiers pour recevoir vos mails et je peux faire quelque chose pour vous