demo shell

Lorsqu'on ouvre un terminal virtuel, celui-ci exécute un interpréteur shell (par défaut `bash` sous Debian).

Prompt

La ligne courante du shell est introduite par une *invite* de commandes (en: prompt). Par défaut sous Debian, elle est de la forme :

```
<user_name>@<host_name>:<current_directory>$
```

ce qui est très utile pour se souvenir du contexte lorsqu'on jongle avec plusieurs terminaux connectés sur des systèmes différents.

Par exemple, si le shell est exécuté par l'user `alice` sur le système nommé `babasse`, et que le répertoire courant est `/etc/`, alors le prompt ressemblera à :

```
alice@babasse:/etc$
```

Pour l'user `root`, le prompt se finit par un `#` au lieu d'un `$` :

```
root@<host_name>:<current_directory>#
```

Il est possible de modifier le prompt (par exemple en y rajoutant l'heure, ou le code d'erreur de la commande précédente) en modifiant la variable `PS1`.

Dans les documentations (et pour certains shells comme `dash`), le prompt est réduit à sa plus simple version : `$` pour les users non-privilégiés et `#` pour l'user `root`.

REPL (read-eval-print loop)

L'interpréteur shell fonctionne sous la forme d'une boucle infinie "lire, évaluer, afficher" (en: REPL, pour read-eval-print loop).

Ce qui signifie que l'utilisat·rice tape quelque chose à droite du prompt courant, le shell l'évalue lorsque l'user appuie sur la touche `[ENTER]` et écrit le résultat en dessous, puis recommence cette boucle en proposant un nouveau prompt :

```
$ whoami
alice
$ echo hop
hop
$
```

Commande, arguments, options

Souvent, une *commande* est constituée de plusieurs *arguments*, séparés par des espaces. La commande suivante liste les fichiers nommés par le répertoire `/usr/bin` :

```
$ ls /usr/bin
```

Les *arguments* de cette commande sont : `ls` et `/usr/bin`.

- le *nom de commande* est `ls`, c'est l'argument numéroté `0`.
- l'argument numéroté `1` est `/usr/bin`

Dans l'exemple précédent, il est possible d'obtenir les tailles (en: size) des fichiers en plus de leurs noms :

```
$ ls --size /usr/bin
```

L'argument `--size` est une *option,* sa fonction est de modifier le comportement de la commande `ls /usr/bin`

Souvent, une option admet une écriture longue introduite par `--` et une écriture raccourcie introduite par `-`. Les écritures raccourcies peuvent parfois se concaténer, par exemple les trois commandes sont équivalentes :

```
$ ls --size --human-readable /usr/bin
$ ls -s -h /usr/bin
$ ls -sh /usr/bin
```

PATH

Dans l'exemple précédent, si on regarde en détail le contenu du répertoire `/usr/bin/` :

```
$ ls -l /usr/bin
```

on observe tout un tas de noms de fichiers avec la permission d'exécution. Le répertoire `/usr/bin/` contient de nombreux *exécutables,* qui peuvent être exécutés comme des commandes par le shell, dont l'exécutable `ls` :

```
$ ls -l /usr/bin/ls
-rwxr-xr-x 1 root root 147176 24 sept.  2020 /usr/bin/ls
```

Comment le shell fait-il pour savoir ou trouver les fichiers exécutables ?

Chaque processus (et en particulier le shell) a une variable d'environnement nommée `PATH` qui contient la liste des répertoires dans lesquels chercher les exécutables.

Si je crée un script bash de 2 lignes nommé `script.sh` de type hello world (qui va au passage nous permettre de voir la numérotation des arguments) dans `/tmp` :

```
$ echo '#!/usr/bin/bash' > /tmp/script.sh
$ echo 'echo "Bonjour, je suis ${0} et vous me demandez de répéter : ${1}"' >> /tmp/script.sh
```

Si on veut l'exécuter directement :

```
$ /tmp/script.sh Coucou
bash: /tmp/script.sh: Permission non accordée
```

On le rend exécutable :

```
$ chmod u+x /tmp/script.sh
$ /tmp/script.sh Coucou
Bonjour, je suis /tmp/script.sh et vous me demandez de répéter : Coucou
```

Si on va dans le répertoire `/tmp` :

```
$ cd /tmp/
$ script.sh Coucou
bash: script.sh : commande introuvable
```

le shell ne le trouve pas bien qu'il se trouve dans le répertoire courant. Mais ça marche si on donne un de ses chemins :

```
$ ./script.sh Coucou
Bonjour, je suis ./script.sh et vous me demandez de répéter : Coucou
```

Autrement dit : si on donne le chemin (absolu ou relatif) d'un exécutable il sera exécuté, si on ne donne que son nom, le shell va chercher dans les divers répertoires indiqués par la variable `PATH` et exécutera le premier qu'il trouvera.

Imaginez qu'une personne mette dans le répertoire `/tmp/` un exécutable malveillant nommé non pas `script.sh` mais `ls`, alors un autre user qui serait dans ce répertoire et qui ferait `ls` pour voir son contenu n'exécutera pas le script malveillant `/tmp/ls` sauf s'il le demande explicitement. C'est plutôt rassurant car tout le monde peut écrire dans `/tmp`.

Dans notre cas, `/tmp` n'est pas dans le `PATH` du shell en cours :

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

On pourrait décider d'ajouter le répertoire `/tmp` au début de la liste des répertoires du `PATH`, et alors la commande `script.sh` exécuterait le fichier exécutable `/tmp/script.sh` :

```
$ PATH=/tmp:${PATH}
$ echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

$ script.sh
Bonjour, je suis /tmp/script.sh et vous me demandez de répéter :
```

Il n'y a pas de problème de sécurité car chaque processus possède sa propre variable d'environnement `PATH`, donc un adversaire ne peut pas modifier le `PATH` de mon shell pour y ajouter le répertoire contenant son exécutable malveillant, si je le change, c'est moi qui décide.

Types de commandes

Il est possible de savoir comment le shell localise une commande grâce à la commande `type` :

Pour la commande `mkdir`, le shell trouve l'exécutable dans `/usr/bin` (qui fait partie du `PATH`) :

```
$ type mkdir
mkdir est /usr/bin/mkdir
```

Une commande ne commence pas forcément par un exécutable. Le shell est un langage de programmation avec ses propres mots-clefs, par exemple dans la boucle suivante :

```
$ for i in 1 2 3 4 5 ; do echo coucou ${i} ; done
coucou 1
coucou 2
coucou 3
coucou 4
coucou 5
```

ici, `for` n'est pas une commande mais un mot-clef du shell :

```
$ type for
for est un mot-clé du shell
```

Certaines commandes sont fournies par le shell :

```
$ type cd
cd est une primitive du shell
```

Pour la commande `ls` utilisée plus haut :

```
$ type ls
ls est un alias vers « ls --color=auto »
```

Il se trouve que l'exécutable `/usr/bin/ls` ne met pas de couleur par défaut, donc pour éviter de devoir mettre l'option de coloration à chaque fois, on a fait un alias, avec le même nom que l'exécutable.

On peut décider de détruire cet alias et alors le shell trouvera l'exécutable qui se trouve dans `/usr/bin` :

```
$ unalias ls
$ type ls
ls est /usr/bin/ls
```

Mais du coup, on perd la couleur par défaut :

```
$ ls /usr/bin/
```

Pour avoir la couleur, il faudrait faire :

```
$ ls --color=auto /usr/bin/
```

Un alias est une sorte de raccourci, on peut définir un alias comme on veut, par exemple pour avoir les tailles comme plus haut on aurait pu faire :

```
$ alias lls='ls -sh'
$ lls /usr/bin
```

Voire, pour avoir la couleur et les tailles :

```
$ alias lls='ls --color=auto -sh'
$ lls /usr/bin
```

Ou encore :

```
$ alias ec='echo Coucou !'
$ ec
Coucou !
```