le fichier /etc/passwd

- Le fichier **/etc/passwd** contient toutes les informations relatives aux utilisateurs (login, mots de passe, ...). Seul le superutilisateur (root) doit pouvoir le modifier. Il faut donc modifier les droits de ce fichier de façon à ce qu'il soit en lecture seule pour les autres utilisateurs.
- Ce fichier possède un format spécial permettant de repérer chaque utilisateur, chacune de ces lignes possède le format suivant:

```
nom_du_compte : mot_de_passe : numero_utilisateur : numero_de_groupe : commentaire : répertoire :
    programme_de_demarrage
root:x:0:0:root:/root:/bin/bash
Mohamed:x:210:520:Mohamed ben mohamed : /home/mohamed:/bin/bash
```

Sept champs sont explicités séparés par le caractère ":" :

- le **nom du compte** de l'utilisateur
- le **mot de passe** de l'utilisateur (codé bien sûr)
- l'**entier** qui identifie **l'utilisateur** pour le système d'exploitation (UID=User ID, identifiant utilisateur)
- l'**entier** qui identifie le **groupe** de l'utilisateur (GID=Group ID, identifiant de groupe)
- le **commentaire** dans lequel on peut retrouver des informations sur l'utilisateur ou simplement son nom réel
- le **répertoire de connexion** qui est celui dans lequel il se trouve après s'être connecté au système
- la **commande** est celle exécutée **après connexion** au système (c'est fréquemment un interpréteur de commandes)

La commande cut

La commande **cut** permet d'afficher des zones spécifiques d'un fichier

- On peut également spécifier un *séparateur de champs* avec l'option *-d*. Par exemple :
- cut -d: -f6 /etc/passwd affichera le 6eme champ du fichier /etc/passwd, dont le séparateur de champs est le caractére double point (``:'').

Exemple :

- récupérez la liste de tous les noms de login autorisés en local sur la machine (cut -d: -f 1 /etc/passwd)
- récupérez la liste de tous les noms de login autorisés en local sur la machine avec le champ commentaire (cut -d: -f 1,5 /etc/passwd)

Les scripts

- Un script est un fichier contenant un ensemble de commandes exécutées séquentiellement
  - Sous forme de fichier texte contenant les commandes
- Le langage de script shell est un langage évolué offrant de nombreuses possibilités
  - Boucles, variables, tests avec if, création de fonctions, …
- Dans quels cas utilise-t-on les scripts ?
  - Pour effectuer un travail répétitif
  - Pour des tâches d'administration système
  - Pour installer des programmes
  - Au démarrage du système pour démarrer les services et applis
  - Bref : Tout le temps !!!

Les variables d'environnement

- Ces variables sont définies à l'ouverture de session
- Leurs valeurs dépendent de l'utilisateur connecté
- Exemple : Variable PATH
  - Défini les différents chemins où chercher les commandes

```
root@fredon:~# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/X11R6/bin
```

- La commande « export » permet de créer/modifier une variable
  - Cette modification est temporaire

```
root@fredon:~# export VAR=VALEUR
```

- Pour modifier la variable PATH sans effacer son contenu

```
root@fredon:~# export PATH=$PATH:/nouveau/repertoire
```


Format de script

- Il s'agit d'un fichier texte au format suivant :

```
#! /bin/sh

# Commentaires : Fonction du script
# Auteur : toto
# Version : 1.0 - Oct 2008

# Début du script
…
```

- Un script peut accepter des arguments
  - Il faut donc vérifier ces arguments avant de commencer le traitement
  - Rappeler le fonctionnement du script par un message d'erreur

```
Usage de la commande : script.sh arg1 arg2 arg3

arg1 : 1er argument , arg2 : 2ème  argument, arg3 : 3ème  argument
```


Variables de substitution

- Elles sont définies implicitement et peuvent être utilisées à tout moment dans le script
- Quelques variables utiles
  - $0 : Nom du script (Utile lorsqu'on renomme le script)
  - $1 à $9 : Argument 1 à 9 passés au script
  - $# : Nombre d'arguments passés au script
  - $? : Résultat de la commande précédente
  - Exemple

```
#! /bin/sh
cp $1 tata.txt
echo $?
```

  - Exécution

```
root@fredon:~/Documents/cours-shell# ./script.sh file.txt
cp: ne peut évaluer `file.txt': Aucun fichier ou dossier de ce type
1
```

Structure de contrôle « if » (1)

- Permet d'effectuer une exécution conditionnelle

```
if [ expression ]
then
    # commandes à exécuter si la cond. est vraie
else
    # commandes à exécuter si la cond. est fausse
fi
```

- L'expression est constituée d'opérateurs
  - Liste des opérateurs numériques
    - -eq : Egalité (Equals)
    - -ne : Non égalité (Non Equals)
    - -lt : Infériorité (Less than)
    - -le : Infériorité ou égalité (Less than)
    - -gt : Supériorité (Greater then)
    - -ge : Supériorité ou égalité (Greater equals)

Structure de contrôle « if » (2)

- Liste des opérateurs sur chaîne de caractère
  - -z : Chaîne vide
  - -n : Chaîne non vide
  - = : Egalité de chaîne
  - != : Non égalité de chaîne
- Liste des opérateurs sur fichiers
  - -L : Lien symbolique
  - -d , -f : Répertoire, Fichier
  - -s : Fichier vide
  - -r, -w, -x : Droits qui s'appliquent (Lecteur, Ecriture, Exécution)
  - -nt : Plus récent (Newer than)
  - -ot : Plus vieux (Older than)

Structure de contrôle « if » (3)

- Liste des opérateurs logiques
  - ! : Négation
  - -a : Et (And)
  - -o : Ou (Or)
- Exemples

```
# Teste si le paramètre $1 est egal à 2
if [ $1 -eq 2 ]
then
   # Commande
fi
```

```
# Teste si le paramètre $1 est un fichier OU un répertoire
if [ -d $1 -o -f $2 ]
then
   # Commande
fi
```

```
# Teste si le fichier existe
if [ ! -f "/etc/toto.conf«  ]
then
   # Commande exécutée si le fichier n'existe pas
fi
```


Exercices

- Tester si le paramètre 2 est un fichier

```
if [ -f $2 ]; then ; fi
```

- Tester si le paramètre 1 égale à toto ET paramètre 2 supérieur à 3

```
if [ $1="toto" -a $2 -gt 3 ]; then ; fi
```

- Tester si le paramètre 1 n'est pas un répertoire ET s'il a le droit d'écriture

```
if [ ! -d $1 -a -w $1 ]; then ; fi
```

- Script capable de créer le répertoire passé en argument et de vérifier que sa création n'a pas provoqué d'erreurs

```
if [ ! -d $1 ]; then
   mkdir $1
   if [ $? -eq 0 ]; then
      echo "Erreur lors de la création de $1"
      exit 1
   fi
fi
```