

# Systemes d'exploitation

Institut Supérieur du Numérique



Année Scolaire 2021/2022

Source Audrey Queudet  
Université de Nantes

# Plan du cours

 Introduction aux systèmes d'exploitation


 Présentation générale d'UNIX

 Programmation shell

 **Gestion des utilisateurs et groupes**

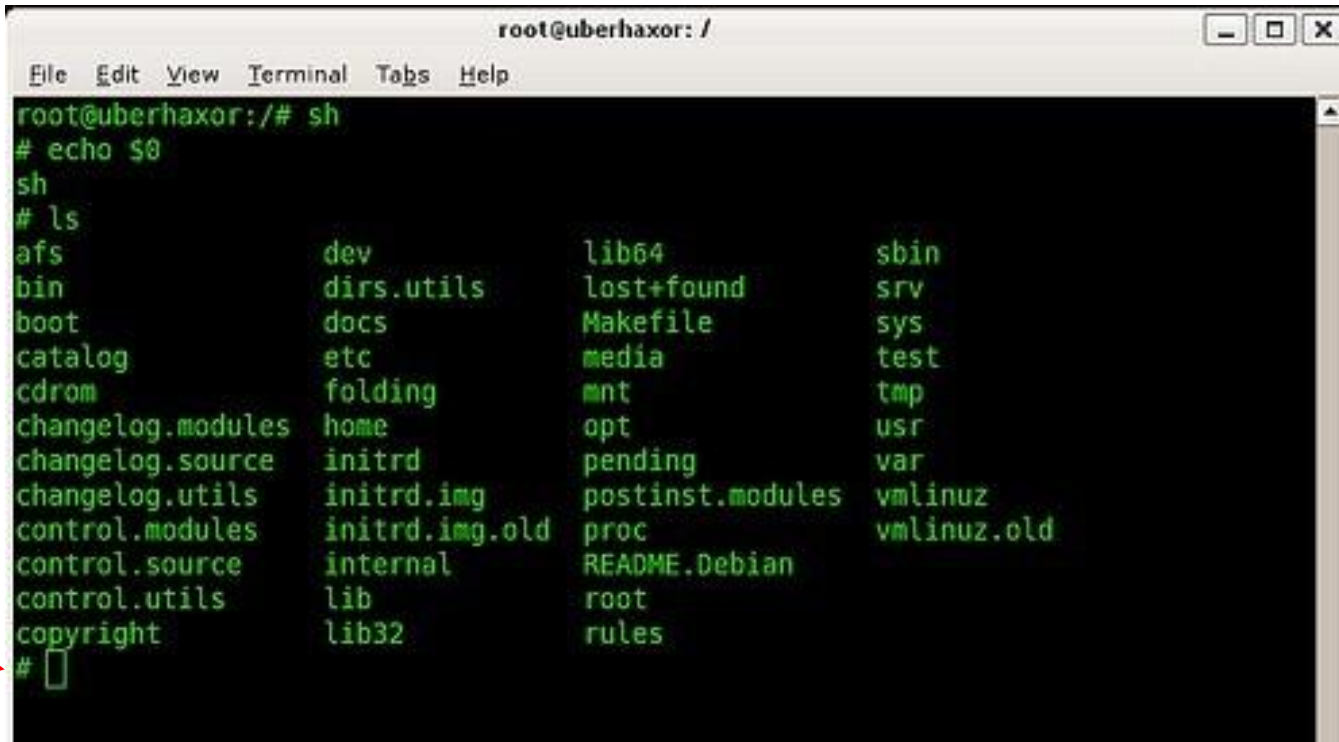
# Le Shell UNIX

## (1)

- **Interface en ligne de commande UNIX** (=IHM dans laquelle la communication entre l'utilisateur et l'ordinateur s'effectue en mode texte)
- Le shell est utilisable en conjonction avec un **terminal** 
- Lors du **login**, l'utilisateur est connecté avec un shell défini lors de la création de son compte. Possibilité de le modifier via la commande **chsh**
- 2 modes d'utilisation :
  - ➔ Simple interpréteur de commandes (mode interactif)
  - ➔ Langage de programmation interprété (scripts)

## Le Shell UNIX (2)

- Le shell affiche une invite en début de ligne, appelée **prompt** ('\$' ou '#' ou '%'), pour indiquer à l'utilisateur qu'il attend l'entrée d'une commande



```
root@uberhaxor: /  
File Edit View Terminal Tabs Help  
root@uberhaxor: /# sh  
# echo $0  
sh  
# ls  
afs          dev          lib64        sbin  
bin          dirs.utils  lost+found   srv  
boot        docs        Makefile     sys  
catalog     etc         media        test  
cdrom       folding     mnt          tmp  
changelog.modules home        opt          usr  
changelog.source initrd      pending      var  
changelog.utils  initrd.img postinst.modules vmlinuz  
control.modules  initrd.img.old proc         vmlinuz.old  
control.source   internal   README.Debian  
control.utils    lib        root  
copyright        lib32      rules  
#
```

A red arrow points to the prompt character '#' at the end of the last line.

# Pourquoi utiliser un shell : avantages ?

- Dans de nombreux contextes, on ne dispose pas d'interface graphique
- Travail en ligne de commande souvent plus efficace qu'à travers une interface graphique
- Automatisation de tâches répétitives
- Meilleure compréhension du système UNIX (fichiers de configuration...)



# Pourquoi ne pas utiliser un shell : inconvenients ?

- Documentation difficile d'accès pour le débutant
- Syntaxe cohérente mais parfois obscure (concision vs. clarté)
- Messages d'erreurs parfois difficilement exploitables
- Relative lenteur





# Les différents shells

- Shell de Stephen R. Bourne :
  - Bourne shell : **sh**
  - Bourne-Again shell : **bash**
- Shell de David Korn :
  - Korn shell : **ksh**
- C shell : **csh**
- Tenex C shell (version moderne du csh) : **tcsh**
- Shell de Kenneth Almquist prenant peu de place sur le disque :
  - Almquist shell : **ash**
  - Debian Almquist shell : **dash**
- Z Shell (**zsh**), intégrant les fcts les plus pratiques de bash, ksh et tcsh

# Scripts shell en bash : les concepts de base

- **Caractères spéciaux**
- **Variables**
  - variables d'environnement
  - variables de l'utilisateur
- **Opérateurs**
- **Structures de contrôle**
  - exécution conditionnelle
  - choix multiple
  - boucle for
  - boucles while et until
- **Expressions régulières**





# Caractères spéciaux (ou métacaractères)

<b><i>Caractère</i></b>	<b><i>Description</i></b>
<b>*</b>	Métacaractère qui remplace n'importe quelle chaîne de caractères (même vide)
<b>?</b>	Métacaractère qui remplace un caractère quelconque
<b>;</b>	Permet de séparer plusieurs commandes écrites sur une même ligne
<b>( )</b>	Regroupe des commandes
<b>&amp;</b>	Permet le lancement d'un processus en arrièreplan
<b> </b>	Permet la communication par tube entre deux commandes
<b>#</b>	Introduit un commentaire. Tout ce qui suit dans une ligne est ignoré par le shell
<b>\</b>	Déspecialise le caractère qui suit
<b>' . . . '</b>	Définit une chaîne de caractères qui ne sera pas évaluée par le shell
<b>" . . . "</b>	Définit une chaîne de caractères dont les variables seront évaluées par le shell
<b>` . . . `</b>	Définit une chaîne de caractères qui sera interprétée comme une commande et remplacée par la chaîne qui serait renvoyée à l'exécution de la dite commande

# Variables d'environnement (1)

<i><b>Variable</b></i>	<i><b>Description</b></i>
<b>PWD</b>	Stocke le chemin et le nom du répertoire courant
<b>HOSTNAME</b>	Nom du serveur
<b>HISTSIZE</b>	Taille de l'historique des dernières commandes passées au shell
<b>LANGUAGE</b>	Suffixe de la langue du système
<b>PS1</b>	Chaîne apparaissant à l'invite du Shell
<b>USER</b>	Nom de l'utilisateur
<b>DISPLAY</b>	Adresse du terminal d'affichage
<b>SHELL</b>	Chemin et nom du programme Shell
<b>HOME</b>	Chemin du répertoire de connexion
<b>PATH</b>	Liste des répertoires où chercher les exécutable des commandes externes

# Variables d'environnement (2)

- Les variables d'environnement sont manipulées via les commandes :

- **printenv** : affiche la liste des variables d'environnement
- **export VARIABLE=VALEUR** : donne une valeur à une variable
- **echo \$VARIABLE** : affiche la valeur de la variable

- Exemples :

```
printenv
PWD=/home/Olivier
LANG=fr
SHELL=/bin/bash

printenv LANG
fr
```

# Variables de l'utilisateur

- L'utilisateur peut déclarer facilement de nouvelles variables par l'affectation directe d'une valeur (numérique, chaîne de caractères) :

```
ma_variable=valeur
```

- Exemples :

```
EMAIL=audrey.queudet@univ-nantes.fr
moi=audrey
vous=L2
phrase1="Bonjour $vous, moi c'est $moi"
phrase2='Bonjour $vous, moi c'est $moi'
echo $phrase1
Bonjour L2, moi c'est audrey
echo $phrase2
Bonjour $vous, moi c'est $moi

rep=`pwd`
echo $rep
/home/queudet/data
```

# Opérateurs sur les fichiers

<b><i>Opérateur</i></b>	<b><i>Description</i></b>
<b><i>-e filename</i></b>	Vrai si <i>filename</i> existe
<b><i>-d filename</i></b>	Vrai si <i>filename</i> est un répertoire
<b><i>-f filename</i></b>	Vrai si <i>filename</i> est un fichier ordinaire
<b><i>-L filename</i></b>	Vrai si <i>filename</i> est un lien symbolique
<b><i>-r filename</i></b>	Vrai si <i>filename</i> est lisible(r)
<b><i>-w filename</i></b>	Vrai si <i>filename</i> est modifiable(w)
<b><i>-x filename</i></b>	Vrai si <i>filename</i> est exécutable(x)
<b><i>file1 -nt file2</i></b>	Vrai si file1 plus récent que file2
<b><i>file1 -ot file2</i></b>	Vrai si file1 plus ancien que file2

# Opérateurs sur les chaînes

<i><b>Opérateur</b></i>	<i><b>Description</b></i>
<i><b>-z chaîne</b></i>	Vrai si la chaîne est vide
<i><b>-n chaîne</b></i>	Vrai si la chaîne est non vide
<i><b>chaîne1 = chaîne2</b></i>	Vrai si les deux chaînes sont égales
<i><b>Chaîne1 != chaîne2</b></i>	Vrai si les deux chaînes sont différentes

# Opérateurs arithmétiques

<i><b>Opérateur</b></i>	<i><b>Description</b></i>
<b>+</b>	addition
<b>-</b>	soustraction
<b>*</b>	multiplication
<b>/</b>	division
<b>**</b>	puissance
<b>%</b>	modulo

- Expressions arithmétiques :

```
$(( ... ))  
  
n=1  
echo $(( 5*n+1 ))
```

# Opérateurs de comparaison numérique

<i>Opérateur</i>	<i>Description</i>
<i>num1 -eq num2</i>	égalité
<i>num1 -ne num2</i>	inégalité
<i>num1 -lt num2</i>	inférieur (<)
<i>num1 -le num2</i>	inférieur ou égal ( $\leq$ )
<i>num1 -gt num2</i>	supérieur (>)
<i>num1 -ge num2</i>	supérieur ou égal ( $\geq$ )



# Opérateurs booléens

<i><b>Opérateur</b></i>	<i><b>Description</b></i>
<b>-a</b>	ET logique
<b>-o</b>	OU logique
<b>!</b>	NON logique

# Structures de contrôle : exécution conditionnelle

- L'instruction **if** permet d'exécuter des instructions si une condition est vraie

- ➔ Le bloc **if/then**

```
if [ condition ]  
then  
    actions  
fi
```

- ➔ Le bloc **if/then/else**

```
if [ condition ]  
then  
    action1  
else  
    action2  
fi
```

- ➔ Enchaînement de plusieurs conditions

```
if [ condition1 ]  
then  
    action1  
elif [ condition2 ]  
then  
    action2  
elif [ condition 3 ]  
then  
    action3  
else  
    action4  
fi
```

# Structures de contrôle : choix multiple

- L'instruction **case** permet de choisir une suite d'instructions suivant la valeur d'une expression

```
case "$x" in
    case1)
        actions1
        ;;
    case2)
        actions2
        ;;
    ...
    caseN)
        actionsN
        ;;
esac
```

# Structures de contrôle :

## boucle for

- L'instruction **for** permet une exécution répétitive d'une suite d'instructions

### → Schéma classique

```
for VAR in LISTE
do
    actions
done
```

### → Schéma alternatif

```
for ((initialisation de VAR; contrôle de VAR; modification de VAR))
do
    actions
done
```

# Structures de contrôle : boucles while et until

- L'instruction **while** permet une exécution répétitive d'une suite d'instructions tant qu'une condition est vraie

```
while [ condition ]  
do  
    actions  
done
```



Condition de continuation de la boucle

- L'instruction **until** permet une exécution répétitive d'une suite d'instructions jusqu'à ce qu'une condition soit vraie

```
until [ condition ]  
do  
    actions  
done
```



Condition d'arrêt de la boucle

# Expressions régulières : définition

- Une **expression régulière** est un patron qui recouvre un ensemble de chaînes de caractères
- Les expressions régulières sont puissantes pour extraire des lignes particulières d'un fichier ou d'un résultat
- Beaucoup de commandes UNIX emploient des expressions régulières
- Bash a des fonctionnalités intégrées pour cibler des patrons et peut reconnaître des classes de caractères et des intervalles.

# Expressions régulières : les opérateurs

<i><b>Opérateur</b></i>	<i><b>Description</b></i>
.	Correspond a tout caractère
?	L'élément précédent est optionnel et sera présent au plus une fois
*	L'élément précédent sera présent zéro fois ou plus
+	L'élément précédent sera présent une fois ou plus
{N}	L'élément précédent sera présent exactement N fois
{N, }	L'élément précédent sera présent N ou plus de fois
{N,M}	L'élément précédent sera présent au moins N fois, mais pas plus de M fois
-	Représente l'intervalle s'il n'est pas le premier ou le dernier dans une liste
^	Correspond à une chaîne vide au début de la ligne; Représente aussi les caractères ne se trouvant pas dans l'intervalle d'une liste
\$	Correspond à la chaîne vide à la fin d'une ligne
\b	Correspond à la chaîne vide au début ou à la fin d'un mot

# Structure d'un script shell

- Un script bash est un simple fichier texte exécutable (droit x) dont la première ligne doit obligatoirement être `#!/bin/bash`  
  
#! Sur la première ligne : interpréteur du présent script (# ! suivi du chemin complet du shell utilisé plus d'éventuels arguments)  
  
#commentaires Les ligne de commentaire sont précédées de #
- Dans un éditeur de texte, écrivons le script suivant :

```
#!/bin/bash
#
# Shell-script affichant "bonjour" sur la sortie standard
#
message='bonjour'
echo $message
```

- Enregistrons ce script sous le nom `bonjour.sh`



# Exécution de scripts bash

- Dans un terminal, en ligne de commande, rendons le script exécutable :

```
chmod u+x bonjour.sh
```

- Exécutons le script (plusieurs solutions) :

```
bonjour.sh
```

ou 

```
. bonjour.sh
```

ou 

```
sh bonjour.sh
```

ajouter le chemin qui contient le script à la variable PATH  
**export PATH=\$PATH:/home/mes\_scripts**

ou 

```
exec bonjour.sh
```

# Passage de paramètres à un script

- Il est possible d'exécuter un script en lui passant un certain nombre de paramètres (ou arguments), comme pour n'importe quelle autre commande :

```
mon_script.sh arg1 arg2 ... argN
```

- En bash, les arguments de la ligne de commande sont stockées dans des variables spéciales :

<i><b>Variable</b></i>	<i><b>Description</b></i>
<b>\$#</b>	Le nombre de paramètres passés au script shell
<b>\$* et \$@</b>	Tous les paramètres passés au script shell
<b>\$0</b>	Nom de la commande
<b>\$1</b>	Valeur du premier paramètre
<b>\$i</b>	Valeur du ième paramètre si i compris entre 1 et 9
<b>\$9</b>	Valeur du neuvième paramètre

# Passage de paramètres : exemple

```
# !/bin/sh
# Mon programme qui affiche les parametres de #la ligne
de commande
echo "* Le nom du programme est : $0"
echo "* Le troisieme parametre est : $3"
echo "* Le nombre de parametre est : $#"
```

*(The following lines are commented out in the original image)*

```
echo "* Tous les parametres (mots individuels) : $*"
echo "* Tous les parametres : $@"
exit 0
$
```

\$ **./script2.sh un "deux" "trois quatre" cinq**

```
* Le nom du programme est : ./script2.sh
* Le troisieme parametre est : trois quatre
* Le nombre de parametre est : 4
* Tous les parametres (mots individuels) : un deux trois
quatre cinq
* Tous les parametres : un deux trois quatre cinq
$
```

# Passage de paramètres : précaution

```
# !/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12
exit 0

$ ./script3 faux.sh un deux trois quatre cinq six
sept huit neuf dix onze douze
un deux trois quatre cinq six sept huit neuf un0 un1
un2
$

# !/bin/sh
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ${12}
exit 0

$ ./script3 vrai.sh un deux trois quatre cinq six
sept huit neuf dix onze douze
un deux trois quatre cinq six sept huit neuf dix onze
douze
$
```

# Opérateurs logiques du shell : &&, ||

- Souvent utilisé pour une forme compacte et élégante.

ET logique :

cmd1 && cmd2  $\longleftrightarrow$  if cmd1  
then cmd2  
fi

Si la commande 1 réussit, alors faire la commande 2.

- OU logique

cmd1 || cmd2  $\longleftrightarrow$  if ! cmd1  
then cmd2  
fi

Si la commande 1 a échoué, alors faire la commande 2.

```
cat toto || touch toto
```

# Fonction

- On peut regrouper les commandes au sein d'une fonction.  
Une fonction se d'efinit de la mani`ere suivante :

```
nom_fonction ()  
{  
  liste-commandes  
}
```

- Les paramètres au sein de la fonction sont accessibles via \$1, \$2,... \$@, \$#.
- L'appel d'une fonction se fait de la manière suivante :  
 nom\_fonction parametre1 parametre2...
- Une fonction doit être déclarée avant de pouvoir être exécutée.

# Code de retour : return, exit

- **return n** :Renvoie une valeur de retour pour la fonction shell.
- **exit n**: Provoque l'arrêt du shell courant avec un code retour de n si celui-ci est spécifié. S'il n'est pas spécifié, il s'agira de la valeur de retour de la dernière commande exécutée.

# Petits calculs numériques : expr

- `expr` chaine: évalue la chaine de caractère représentant des opérations.

```
$ titi=3
$ echo $titi
3
$ titi=$titi+1
$ echo $titi
3+1
$ tutu=3
$ tutu='expr $tutu + 1'
$ echo $tutu
4
$
```



# Introduction

- Ubuntu Linux utilise des groupes pour vous aider à gérer les utilisateurs, définir les autorisations de ces utilisateurs et même surveiller le temps qu'ils passent devant leur ordinateur.
- Ubuntu Linux utilise des groupes pour vous aider à gérer les utilisateurs, définir les autorisations de ces utilisateurs et même surveiller le temps qu'ils passent devant leur ordinateur.

# Introduction

- Un utilisateur peut appartenir à plusieurs groupes.
- Un fichier peut appartenir à un seul utilisateur et à un seul groupe à la fois.
- Un utilisateur particulier, le superutilisateur "root", dispose de privilèges supplémentaires (uid = "0" dans /etc/passwd).
- Seul **root** peut changer la propriété d'un fichier.

# Comment fonctionnent les comptes d'utilisateur Linux

- Linux stocke les informations liées à tous les utilisateurs dans les fichiers suivants :

- /etc/passwd**: contient les informations de compte d'utilisateur

- /etc/groups**: fichier contient les groupes

- /etc/shadow**: contient les mots de passe

# Comment fonctionnent les comptes d'utilisateur Linux

- Nom d'utilisateur
- Mot de passe
- Par défaut, tous les répertoires personnels des utilisateurs sont créés et gérés dans le répertoire /home.
- Le répertoire personnel de l'utilisateur root est /root.

# Le superutilisateur

- Par défaut, un compte dispose de privilèges élevés pour exécuter n'importe quelle commande, accéder à n'importe quel fichier et effectuer toutes les opérations.
- Superutilisateur, alias root
- Numéro d'utilisateur et de groupe 0

# Le superutilisateur

- Limiter l'utilisation de root :
  - Les utilisateurs inexpérimentés peuvent causer des dommages graves
  - L'utilisation de root pour des tâches non privilégiées est inutile et peut être ouverte aux attaques
  - Violations de sécurité et de confidentialité – root peut voir les fichiers de n'importe qui
- Limiter ce que root peut faire à distance
- Assurez-vous d'un mot de passe root fort

# Les privilèges superutilisateur

- Ce qui fonctionne habituellement le mieux, ce sont de courtes périodes de privilège de superutilisateur, seulement lorsque cela est nécessaire
- Obtenir des privilèges, accomplir la tâche, renoncer aux privilèges
- Les moyens les plus courants sont su et sudo

Peut également utiliser la méthode setuid/setgid, mais ce n'est pas recommandé

# Su

- Abréviation du mot anglais substitute ou switch user
- Syntaxe : su [options] [nom d'utilisateur]
  - Si le nom d'utilisateur est omis, root est supposé
- Après avoir émis la commande, il est demandé le mot de passe de cet utilisateur
- Un nouveau shell est ouvert avec les privilèges de cet utilisateur Une fois les commandes émises terminées, il faut taper **exit**



# Sudo

- sudo Vous permet d'émettre une seule commande en tant qu'un autre utilisateur
- Syntaxe: `sudo [options] [-u user] [commande]`
  - Encore une fois, si aucun utilisateur n'est spécifié, root est supposé
- 
- Un nouveau shell est ouvert avec les privilèges de l'utilisateur  
La commande spécifiée est exécutée  
Le shell est fermé

# sudoers

- Il est nécessaire de configurer un utilisateur pour exécuter des commandes en tant qu'un autre utilisateur lors de l'utilisation de sudo.
- 
- Les autorisations sont stockées dans `/etc/sudoers`.
- Utilisez l'outil visudo pour éditer ce fichier (exécutez en tant que root).

Les autorisations sont accordées à des utilisateurs ou des groupes, pour certaines commandes ou toutes, avec ou sans nécessité de mot de passe.

# Création et gestion de comptes d'utilisateur

- Utilisation de **useradd**
- Utilisation de **passwd**
- Utilisation de **usermod**
- Utilisation de **userdel**

# Utilisation de useradd

- Syntaxe: `useradd options nom_utilisateur`
- Exemple: **`useradd User1`**

Un compte User1 est créé en utilisant les paramètres par défaut contenus dans les fichiers de configuration suivants:

**`/etc/default/useradd`**

**`/etc/login.defs`** Ce fichier contient des valeurs qui peuvent être utilisées pour les paramètres GID et UID lors de la création d'un compte avec `useradd`. Il contient également des paramètres par défaut pour la création de mots de passe dans `/etc/shadow`.

# Utilisation de userdel

- Syntaxe: `userdel nom_utilisateur`
- Exemple: **`userdel User1`**
- Il est important de noter que, par défaut, `userdel` ne supprime pas le répertoire personnel de l'utilisateur du système de fichiers.
- Si vous voulez supprimer le répertoire personnel lorsque vous supprimez l'utilisateur, vous devez utiliser l'option `-r` dans la ligne de commande.
- Par exemple, en entrant **`userdel -r User1`**, vous supprimerez le compte et supprimerez son répertoire personnel.

# Gestion des utilisateurs

- Ajouter un nouvel utilisateur au système  
**sudo adduser Ahmed**
- Pour vérifier : **cat /etc/passwd**
- Pour accéder au système avec le nouvel utilisateur:  
**su - ahmed**
- Pour modifier un utilisateur, utilisez la commande **usermod**.  
Pour en savoir plus sur usermod: **info usermod**

# Gestion des utilisateurs

- Ajouter un nouveau commentaire à l'utilisateur Ahmed  
**sudo usermod -c "Ahmed Ali" Ahmed**
- Pour ajouter Ahmed à un nouveau groupe supplémentaire  
:  
• **sudo usermod -aG Etudiant Ahmed**
- Pour verrouiller l'utilisateur ahmed:  
**sudo usermod -L Ahmed**  
Pour le déverrouiller :  
**sudo usermod -U Ahmed**

# Création et gestion de comptes des groupes

- Utilisation de **groupadd**  
**Utilisation de groupmod**  
Utilisation de **groupdel**
- Les groupes sont définis dans le fichier **/etc/group**.
- Chaque enregistrement est composé des quatre champs suivants:      Groupe:Mot de passe:GID:Utilisateurs
  - Groupe spécifie le nom du groupe.
  - Mot de passe spécifie le mot de passe du groupe.
  - GID spécifie le numéro d'identification de groupe (GID) du groupe.
  - Utilisateurs répertorie les membres du groupe.



# Utilisation de groupadd

- Syntaxe: `groupadd options nom_du_groupe`
- Options:
  - g spécifie un GID pour le nouveau groupe.
  - p spécifie un mot de passe pour le groupe.
  - r spécifie que le groupe en cours de création est un groupe système

# Utilisation de groupdel

- Syntax: **groupdel group\_name**
- Exemple: **groupdel Etudiant**