

# 1 Présentation du module

## Référentiel : Introduction aux systèmes d'exploitation et à leur fonctionnement

Descriptif détaillé

**Objectif :** L'objectif de cette ressource est de comprendre le rôle, les composants et le fonctionnement d'un système d'exploitation. Cette ressource permet de découvrir les principes d'un système d'exploitation, leur mode de fonctionnement et les différents types existants. Elle contribue à comprendre comment installer un système sur une machine et à le personnaliser en développant des fonctions simples facilitant la configuration et le paramétrage.

Savoirs de référence étudiés

- Caractéristiques et types de systèmes d'exploitations
- Langage de commande (commandes de base, introduction à la programmation des scripts)
- Gestion des processus (création, destruction, suivi...)
- Gestion des fichiers (types, droits...)
- Gestion des utilisateurs (caractéristiques, création, suppression...)
- Principes de l'installation et de la configuration d'un système : notion de noyau, de pilotes, de fichiers de configuration, boot système...
- Les différents savoirs de référence pourront être approfondis

Mots clés **Système d'exploitation**  
**utilisateurs**

**Langage de commande**

**Installation système**

**Gestion**

Cursus S1 tous parcours

Prérequis : R1.03 Intro. archi.

Heures totales (27h).....7h TD et 20h TP

[QCM sur moodle](#)

## 1.1 histoire SE

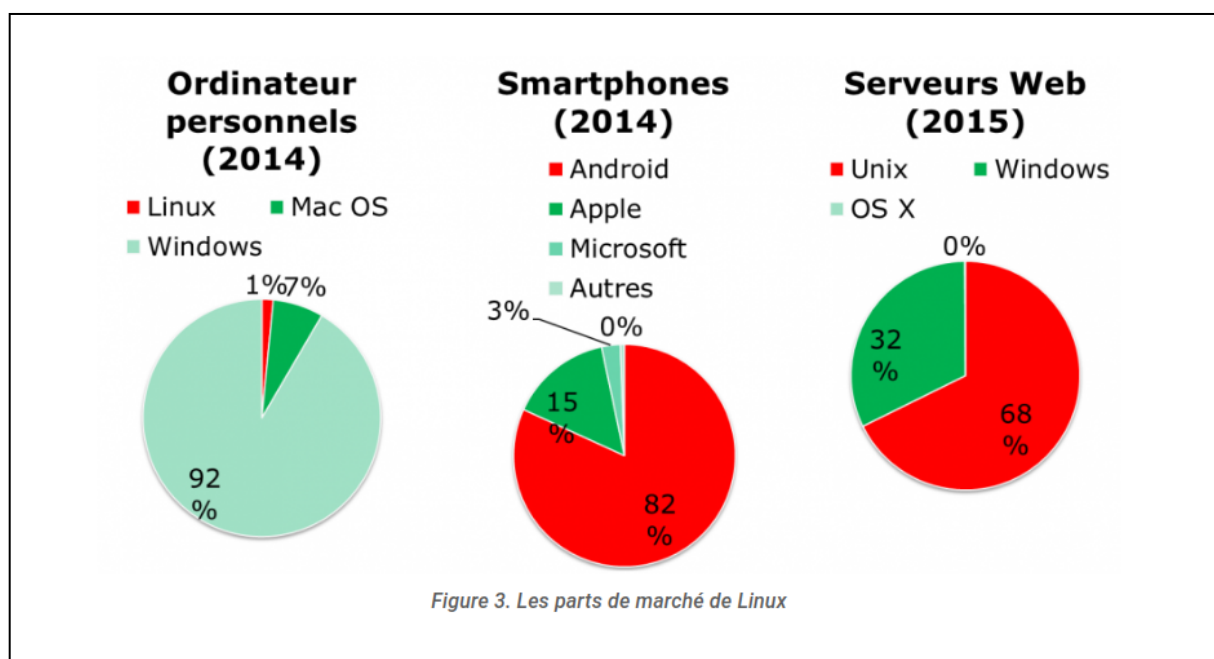
1970 Unix Time-Sharing System (Bell Labs) BSD (Berkeley Software Distribution)

SACHA KRAKOWIAK

Présence des systèmes d'exploitation

- Ordinateurs personnels : Windows, macOS(iOS) ...
- Serveurs : Linux, Windows ....
- Téléphone : Android, Apple, ...
- Systèmes embarqués : Industriel et médical, Télécom (Box ...), Aéro/Défense, Automobile, Electro-ménager (AppleTV AndroidTv) ....
- Objets connectés : Montres, Robots .....

[Parts de marché](#)



## 2 Découvrir le système d'exploitation

### 2.1 Qu'est-ce que la ligne de commande ?

### 2.2 Trouver de l'aide

### 2.3 Gérer les répertoires et les fichiers

### 2.4 Les utilisateurs et leurs droits

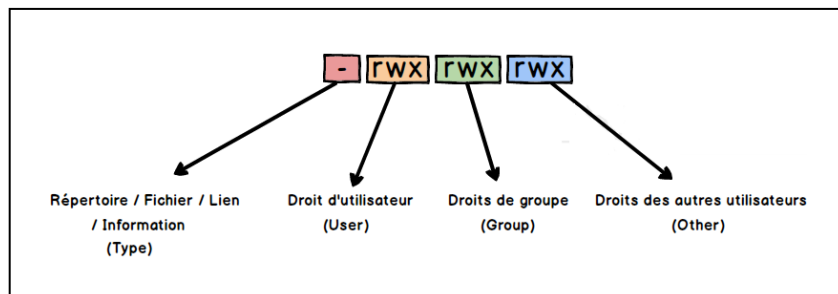
Le premier caractère peut être :

- d** = répertoire
- = fichier régulier
- l** = lien symbolique
- s** = socket du domaine Unix
- p** = pipe nommée
- c** = fichier de périphériques de type caractère
- b** = fichier de périphériques de type bloc

Les caractères suivants peuvent être :

- r** = permission de lecture
- w** = permission d'écriture
- x** = permission d'exécuter
- = pas de permission

La première colonne définit un répertoire, un fichier ou un lien, les trois colonnes suivantes définissent les permissions pour User, Group et Other.



test sur les fichiers : exemple documentation

```
cd /tmp
touch test.sh
ls -la t*
test -x test.sh
echo $?
chmod a-x test.sh
ls -la t*
test -x test.sh
echo $?
```

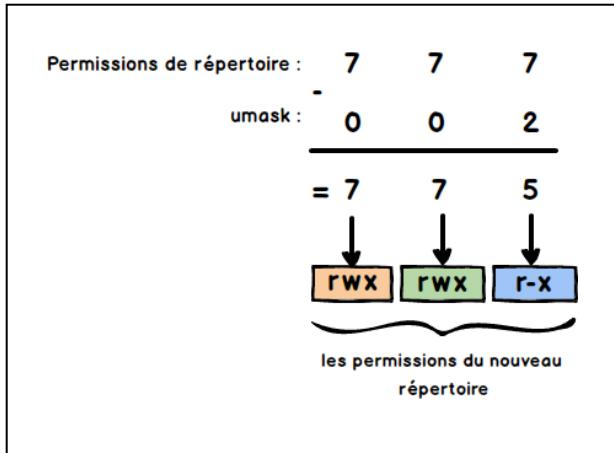
#### 2.4.1 la commande `umask`

prérequis `chmod`

```
cd /tmp
umask
umask -S
old_umask=$(umask)
echo $old_umask
touch test2.sh
ls -la t*
umask 070
```

```
touch test3.sh
ls -la t*
umask $old_umask
```

[exemple documentation](#)



Remarque : Lorsque on ferme la session, la valeur de **umask** revient à sa valeur par défaut qui devrait être dans `/etc/login.defs` (parfois `/etc/profile`)

Définition et modification de [umask](#) pour la session système

## 2.5 Traiter un fichier Texte

## 2.6 Quelques commandes

- [grep](#), exemple `ps -ax | grep python`
- [find](#), exemple `find ~ -name *.sql`
- [sed](#), exemple `sed -i 's/password=.*password="motDePasse"/g' app.py`
- [awk](#), exemple `awk '{print $1,$3,$4}' FS=":" OFS=";" /etc/passwd`

## 3 Interagissez avec le Bash

### 3.1 Aide à l'interaction

- ☐ permet de compléter le nom d'une commande, d'un répertoire ou d'un fichier en cours de saisie
  - ☐ propose des choix
  - Si rien n'est proposé, selon le contexte, une erreur est possible
- [history](#)
- 

ou   : se déplacer dans la ligne de commande

~~ou   ou   ne fonctionnent pas~~

### 3.2 Caractères spéciaux

Abréviations pour le nom des fichiers

**Caractères spéciaux (wildcards) :**

Les caractères "spéciaux" se substituent à un ensemble de caractères : Il s'agit d'un caractère ou d'une chaîne de caractères qui vont être utilisés pour prendre un certain nombre de valeurs.

On parle de **Globbing wildcards** lorsqu'ils vont matcher avec une liste de fichiers ou de répertoires

[documentation](#)

- Ils peuvent être utilisés avec la plupart des commandes comme `ls` ou `rm`.

```
cd /tmp
mkdir a ; cd a
mkdir d d1 d2 d3 dossier4 dossier5
touch f f1 f2 f3 fichier4 fichier5
```

La ligne de commande est exécutée après interprétation et substitution de ces caractères.

- `.` répertoire courant : **une seule valeur possible**
- `..` répertoire parent : **une seule valeur possible**
- `~` répertoire personnel de l'utilisateur : **une seule valeur possible**

exemple `ls` : tester l'interprétation et la substitution des 3 caractères ci-dessus avec la commande `ls`

- `*` remplace une séquence de caractères
  - `ls /*` : la portée de `*` se limite à 1 niveau de l'arborescence de fichiers
- `?` remplace un caractère
- `[]` remplace un seul caractère des caractères mentionnés


ATTENTION : il existe d'autres caractères spéciaux

## 3.3 Constructions syntaxiques

<https://abs.traduc.org/abs-5.3-fr/ch04.html#varsubn>

substitution de variable <https://abs.traduc.org/abs-5.3-fr/ch04.html#varsubn>

### 3.3.1 Comment est évalué une ligne de commande ?

- Phase 1: invite de commande
- Phase 2: saisie exemple : `ls f?`
- Phase 3: validation (retour chariot enfoncé) `ls f?` 
- Phase 4: analyse de la commande `ls f?` => `ls f1 f2`
- Phase 5: interprétation
- Phase 6: exécution

### 3.3.2 Les variables

- conserver une valeur
  - `nom_de_la_variable=valeur`
  - aucun espace autour du caractère `=`
    - `i = 1` => ~~`i=1`~~
    - `i=10`
- Accès à la valeur: `$nom_de_variable`
- l'utilisation d'une **variable indéfinie retourne vide**

exemple

```
i=1
mon_cv=Documents/cv
echo $i
pwd
cd $mon_cv
pwd
echo "::$nondefini::"
```

encore un caractère spécial `$`

- Il est préférable d'encadrer les variables avec des accolades `${variable}`

### 3.3.3 substitution de commande

- capturer la sortie d'une commande : `$(commande)` ou ``commande``
  - pour un argument `echo ici:$(pwd)`
  - pour l'affecter à une variable `rep=$(pwd) ; echo $rep`
    - fonctionne avec les résultats multiples `fichiers=$(ls) ; echo $fichiers`

## Les Inhibitions

- Inhibition totale
  - simple quote `'`
    - exemple : `echo '* $(pwd)'`
- Inhibition partielle

- double quote " : autorise la substitution de variables, l'interprétation de commandes, l'interprétation du caractère spécial \ , la substitution pour les noms de fichiers ou le séparateur espace n'est plus opérant
  - exemple : `a=* ; echo "*" :: :: $a $(pwd)`
  - tester le même exemple sans l'inhibition partielle
- Inhibition caractère
  - anti-slash \ : empêche la substitution de variable
    - exemple : `a=* ; echo \$a`
    - exemple : `echo - \ - ; echo - -`

## Rappel

Lors la phase d'interprétation :

- Étape de la Substitution de :
  - variables
  - 
  - commandes
  - Caractères spéciaux

```
a=*
b=ls

echo ${b}
echo ${a}
echo $(b)
echo $a
```

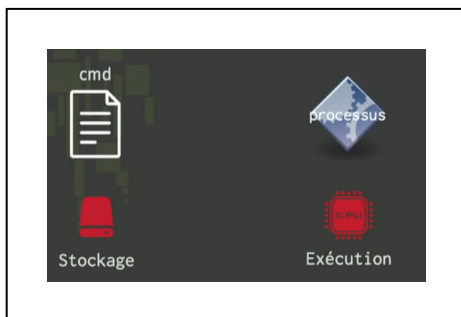
- contrôle de l'interprétation par les inhibitions

## 3.4 Contrôler l'exécution des commandes

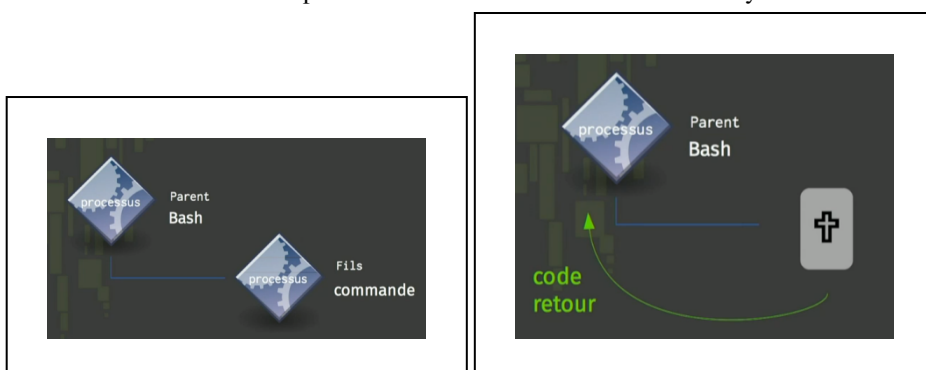
### 3.4.1 Entrées/Sorties lors de la phase d'exécution

#### Notion de Processus

- Une commande est un programme stocké sous la forme d'un fichier sur un support de stockage (D.Dur).
- Invoqué par le Bash, elle s'exécute en tant que processus (dans la RAM).



- Le système d'exploitation gère l'activité du processus depuis sa création jusqu'à sa mort.
- **Un processus est donc une instance d'une commande en exécution**
- Un processus est décrit par un **Contexte d'exécution**:
  - ressources
  - droits
  - attributs
- Le processus créé est appelé **processus fils**, le processus créateur s'appelle **processus parent**
- En fin de traitement le processus meurt et un code retour est envoyé



- Le code retour est 0 si le processus s'est bien déroulé

La convention est :

- Le code retour est 1 si le processus est terminé mais s'est mal déroulé
- Le code retour est 127 si la commande est non trouvée

tester quelques commandes et afficher leurs retours `echo $?`

### 3.4.2 attribut du contexte d'exécution

- Propriétaire **UID** (User Identifier)
- Identifiant : **PID** (Process Identifier)
- Processus parent: **PPID** (Parent Process Identifier)
- Commande, option, arguments **CMD** (Command)
- `ps -f` pour voir les différents processus
- `ps -fax`

Gérer le multitâche dans un terminal

```
sleep 5
    # tâche en arrière plan
sleep 120 &
    # consulter les tâches en arrière plan de ce terminal
jobs
sleep 100 &
jobs
    # remettre une tâche en premier plan
fg %1
    # CTRL Z => suspendre la commande

jobs
    # * remarque 1
bg
jobs

sleep 1000 &

kill -15 PID    # numéro du processus, on le retrouve avec ps
```

- remarque 1: il est possible d'indiquer le numéro du job, exemple %1
- remarque 2 :
  - `Ctrl C` : Arrêt définitif du programme
  - `Ctrl Z` : (suspend) Stoppe le programme, mais on peut le poursuivre avec fg (ou bg, voir #4)
  - `Ctrl d` : Fin de fichier, qui sert aussi à arrêter certains programmes interactifs (terminal, dc, python...)
- remarque 3 : `kill` permet d'envoyer un [signal](#)

dans un nouveau terminal

```
ps -f
sleep 1000 &
jobs
bash
jobs
bash
jobs
sleep 1200 &
jobs
ps -f
    # bien regarder les numéros de processus
# CTRL Z
```

## 3.5 Entrées et sorties des processus

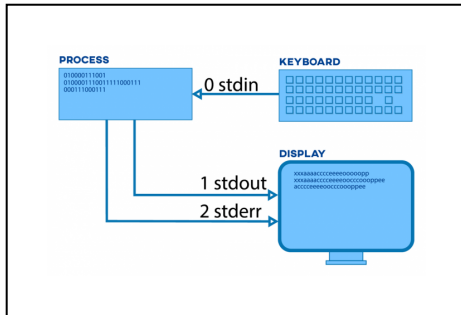
### 3.5.1 combiner des commandes séquentiellement

- Sans lien sur une même ligne : caractère `;` (aucune relation entre les commandes)
- par fichier interposé : **redirection**

- Par branchement : le **pipe** ou **|**

### 3.5.2 entrées/sorties standard d'une commande

- Entrée Standard (par défaut le clavier) : numéro **1** qui se nomme **stdin**
- Sortie Standard (par défaut l'écran) : numéro **0** qui se nomme **stdout**
- Sortie d'erreurs (par défaut l'écran) : numéro **2** qui se nomme **stderr**



### 3.5.3 les modes de redirections

#### redirection de la sortie standard

- mode **simple** **>** , exemple `ls > liste.txt` puis `cat liste.txt`
- mode **en ajout** **>>** , exemple `echo "----" > liste.txt` puis `cat liste.txt`

ou redirection vers un dispositif

- “trou noir” : `> /dev/null` , exemple `echo "----" > /dev/null` (commande muette)

#### redirection de la sortie d'erreurs

2 redirections:

- mode **simple** **2>** , exemple
  - `ls 2> log.txt` ATTENTION pas d'espace entre **2** et **>**
  - puis `cat log.txt`
  - puis `ls a`
  - puis `ls a 2> log.txt`
- mode **en ajout** **>>**
- “trou noir” : idem

#### redirection de l'entrée standard

redirection d'entrée standard **<**

- exemple
  - `wc -w` (word count -l –lines, -w –words, -m –chars, -c –bytes (octets), -L –Max-Line-Longueur)
  - `quelques mots`
  - `Ctrl d`
  - `wc -w < log.txt` , cette commande est différente de `wc -w log.txt`

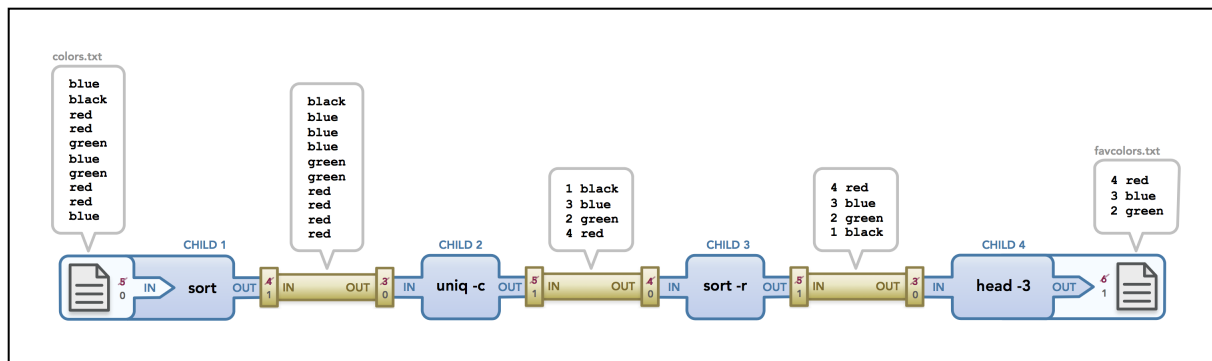
Remarque : rediriger la sortie d'erreur standard vers la sortie standard de la commande avec la syntaxe commande **2>&1**

### branchement de commandes ( tube ou pipe | )

Commande1 | Commande2

`Commande1 -> sortie -> entrée -> Commande2`

testons l'exemple ci-dessous



```

echo -e "blue\nblack\nred\nred\ngreen\nblue\nred\nred\nblue" >> colors.txt
cat colors.txt

sort -nr          # (--reverse --numeric-sort)

```

## Exercice

### exemple

- compter le nombre de commandes dans `history`
- rechercher dans `history` les commandes qui commencent par `mysql`
- rechercher les processus avec le mot `python`
  - lancer une application `flask`
  - rechercher le numéro de processus qui utilise `python` avec `grep`
  - tuer le processus `flask`
- compter le nombre de fichiers qui se terminent par `.java`, utiliser la commande `find`

## Exercice 9 du TP

On dispose d'un fichier csv `login.csv`

```

aduboit;mdp1
bgrange;mdp2
cdurand;mdp3
dgregoire;mdp4
eroi;mdp5

```

**compte type** : Instruction SQL pour créer un compte et une base de données à un étudiant

```

-- compte login
-- mysql --user=login --host=serveurmysql.iut-bm.univ-fcomte.fr --password=motdepasse --database=BDD_login

create database BDD_login;
create user "login" identified by "motdepasse";
GRANT ALL PRIVILEGES ON BDD_login.* To 'login'@'%';

```

Écrire un script `creer_compte_mysql.sh`

ce script doit :

- Si le fichier `creer_compte_mysql.sh` existe déjà afficher une erreur et quitter
- Si le fichier `login.csv` n'existe pas afficher une erreur et quitter
- Faire un boucle qui récupère dans chaque ligne du fichier `csv`, le login (variable `login`) et le mot de passe (variable `motdepasse`)
  - Ajouter dans le fichier `creer_compte_mysql.sh` chaque ligne du **compte type**, remplacer la chaîne de texte `login` par la variable `login` et `motdepasse` par la variable `motdepasse`
- Installer `mysql`
- Exécuter ce script dans `mysql` et tester un compte

# 4 Maîtriser votre système d'exploitation grâce à Bash

## 4.1 Contrôler son environnement

Pour afficher les variables d'environnement : utiliser la commande `env` ou `printenv`



La commande **set** permet d'afficher ses variables et les variables d'environnement.

```
echo $HOME
cd ~/rep
pwd
cd
pwd
HOME=~/rep
cd
pwd
```

## 5 ANNEXES

### 5.1 LES WILDCARDS

Il s'agit d'un caractère ou d'une chaîne de caractères qui vont être utilisés pour prendre un certain nombre de valeurs.

On parle de **Globbing wildcards** lorsqu'ils vont matcher avec une liste de fichiers ou de répertoires

- Ils peuvent être utilisés avec la plupart des commandes comme **ls** ou **rm**.

#### 5.1.1 Liste des wildcards

- Le caractère **\*** : match avec tous les caractères (ou l'absence de caractères) qu'il n'y en ait qu'un seul ou plusieurs.
- Le caractère **?** : match avec un seul caractère.
- Les classes de caractères
  - **[13bc]\*** : match avec tous les fichiers commençant par 1, 2 ou 3
  - **[1-3]\*** : match avec tous les fichiers commençant par 1, 2 ou 3
  - **[a-c]\*** : match avec tous les fichiers commençant par a, b, ou c
- Si on veut effectuer des recherches sur des fichiers qui comportent un caractère de type wildcard, il faut utiliser le caractère d'échappement **\** avant le wildcard.
- **[:alpha:]** : toutes les lettres de l'alphabet (minuscules et majuscules)
- **[:alnum:]** : toutes les lettres de l'alphabet (minuscules et majuscules) ainsi que tous les chiffres (0 à 9)
- **[:digit:]** : tous les chiffres de 0 à 9
- **[:upper:]** : toutes les lettres de l'alphabet en majuscule
- **[:lower:]** : toutes les lettres de l'alphabet en minuscule
- **[:space:]** : tous les caractères d'espacement (espace, tabulation, nouvelle ligne, etc...)

#### 5.1.2 SUID et SGID

**SUID** : « Set owner User ID », est un moyen de transférer des droits aux utilisateurs

**SUID et SGID** : **SUID** est une autorisation spéciale attribuée à un fichier. Ces autorisations permettent d'exécuter le fichier avec les privilèges du propriétaire. Par exemple, si un fichier appartenait à l'utilisateur root et que le bit **SUID** était défini, peu importe qui exécutait le fichier, il s'exécuterait toujours avec les privilèges de l'utilisateur root.

```
1984 cd /tmp
1985 touch file1.sh
1986 ls -la f*
1987 chmod 4644 file1.sh
1988 ls -la f*
1989 chmod 644 file1.sh
1990 ls -la f*
1991 chmod u+s file1.sh
1992 ls -la f*
```

**SGID** : Lorsque le bit Set Group ID est défini, l'exécutable est exécuté avec les droits du groupe. Par exemple, si un fichier appartenait au groupe utilisateur, quel que soit celui qui l'a exécuté, il s'exécuterait toujours avec l'autorité du groupe de l'utilisateur.

```
1994 cd /tmp
1995 touch file2.sh
1996 ls -la f*
1997 chmod 2754 file2.sh
1998 ls -la f*
1999 chmod 754 file2.sh
2000 ls -la f*
```

```
2001  chmod g+s file2.sh
2002  ls -la f
```

[commande **id**]

### 5.1.3 sticky bit

sticky bit : Un sticky bit est un bit d'autorisation sur un fichier ou un répertoire qui permet uniquement au propriétaire du fichier/répertoire ou à l'utilisateur root de supprimer ou de renommer le fichier. Aucun autre utilisateur ne peut supprimer un fichier qu'un autre utilisateur a créé.

```
1974  cd /tmp
1975  touch file1.txt
1976  ls -la f*
1977  chmod o+t file1.txt
1978  ls -la f*
1979  chmod a+x file1.txt
1980  ls -la f*
1981  chmod 1755 file1.txt
1982  ls -la f*
```

<https://www.linuxtricks.fr/wiki/print.php?id=177>

<https://debian-facile.org/doc:programmation:shells:script-bash-etat-de-sorie-et-les-tests>

<https://debian-facile.org/doc:programmation:shells:script-bash-enchainement-de-commandes-et-etat-de-sortie>

<https://www.plesk.com/blog/various/find-files-in-linux-via-command-line/>

[https://lipn.univ-paris13.fr/~cerin/SE/S2SE\\_01\\_LectureFichiersShell2.html](https://lipn.univ-paris13.fr/~cerin/SE/S2SE_01_LectureFichiersShell2.html)

<https://abs.traduc.org/abs-5.3-fr/ch02.html>

<https://www.formatux.fr/formatux-fondamentaux/module-030-utilisateurs/index.html#g%C3%A9n%C3%A9ralit%C3%A9s>

<http://www.linusakesson.net/programming/tty/index.php>