Chapitre 8

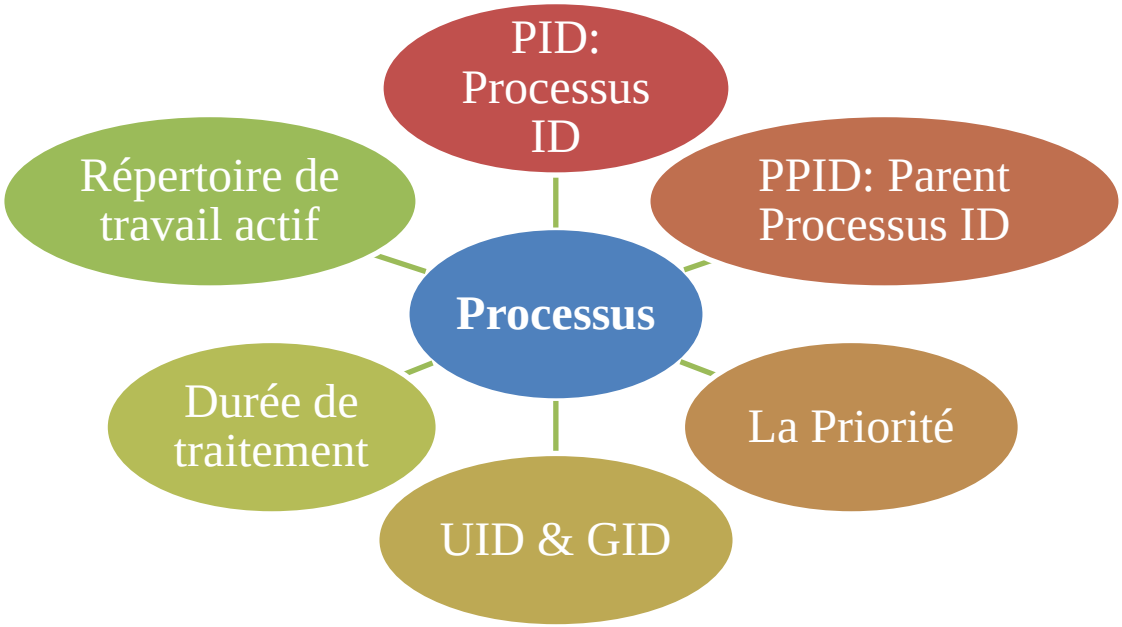
Système d'Exploitation UNIX

La Gestion des Processus

BOUKRI KHALIL

Définition et Environnement

Un **processus** représente à la fois un programme en cours d'exécution et tout son environnement d'exécution (mémoire, état, identifications, propriétaire, père ...).



❖Au démarrage du système un processus nommé **init** est lancé. Il est responsable du lancement des autres processus du système.

❖Un processus peut en lancer d'autres. Ainsi les processus sont groupés en une hiérarchie dont la racine est le processus **init**.

Définition et Environnement

PID (Process ID) : chaque processus Unix est numéroté afin de pouvoir être différencié des autres. Le premier processus lancé par le système est 1 et il s'agit d'un processus appelé généralement *init*

PPID (Parent Process ID) : chaque processus peut lui-même lancer d'autres processus, des processus enfants (child process). Chaque enfant reçoit parmi les informations le PID du processus père qui l'a lancé.

l'UID et le GID de l'utilisateur qui a lancé le processus. C'est nécessaire pour que le système sache si le processus à le droit d'accéder à certaines ressources ou non.

Durée de traitement et priorité : le temps d'exécution écoulé depuis le dernier réveil du processus. Dans un environnement multitâches, le temps d'exécution est partagé entre les divers processus, et tous ne possèdent pas la même priorité.

Définition et Environnement

Répertoire de travail actif : A son lancement, le répertoire courant (celui depuis lequel le processus a été lancé) est transmis au processus. C'est ce répertoire qui servira de base pour les chemins relatifs.

Fichiers ouverts : table des descripteurs des fichiers ouverts. Par défaut au début seuls trois sont présents : 0 1 et 2 (les canaux standards). A chaque ouverture de fichier ou de nouveau canal, la table se remplit. A la fermeture du processus, les descripteurs sont fermés (en principe).

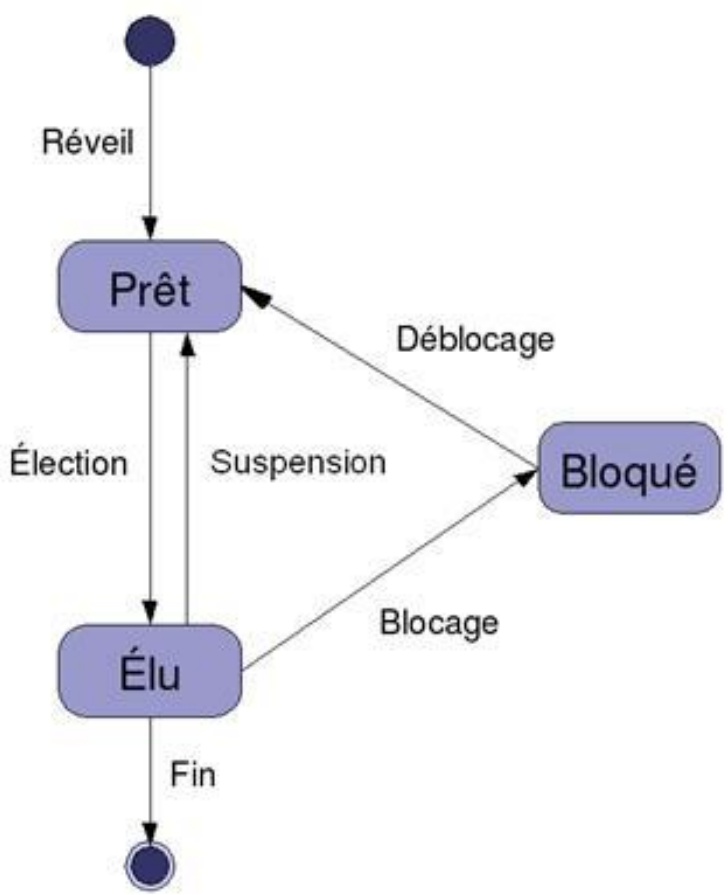
☛ On trouve d'autres informations comme la taille de la mémoire allouée, la date de lancement du processus, le terminal d'attachement, et l'état du processus.

Etats d’un processus

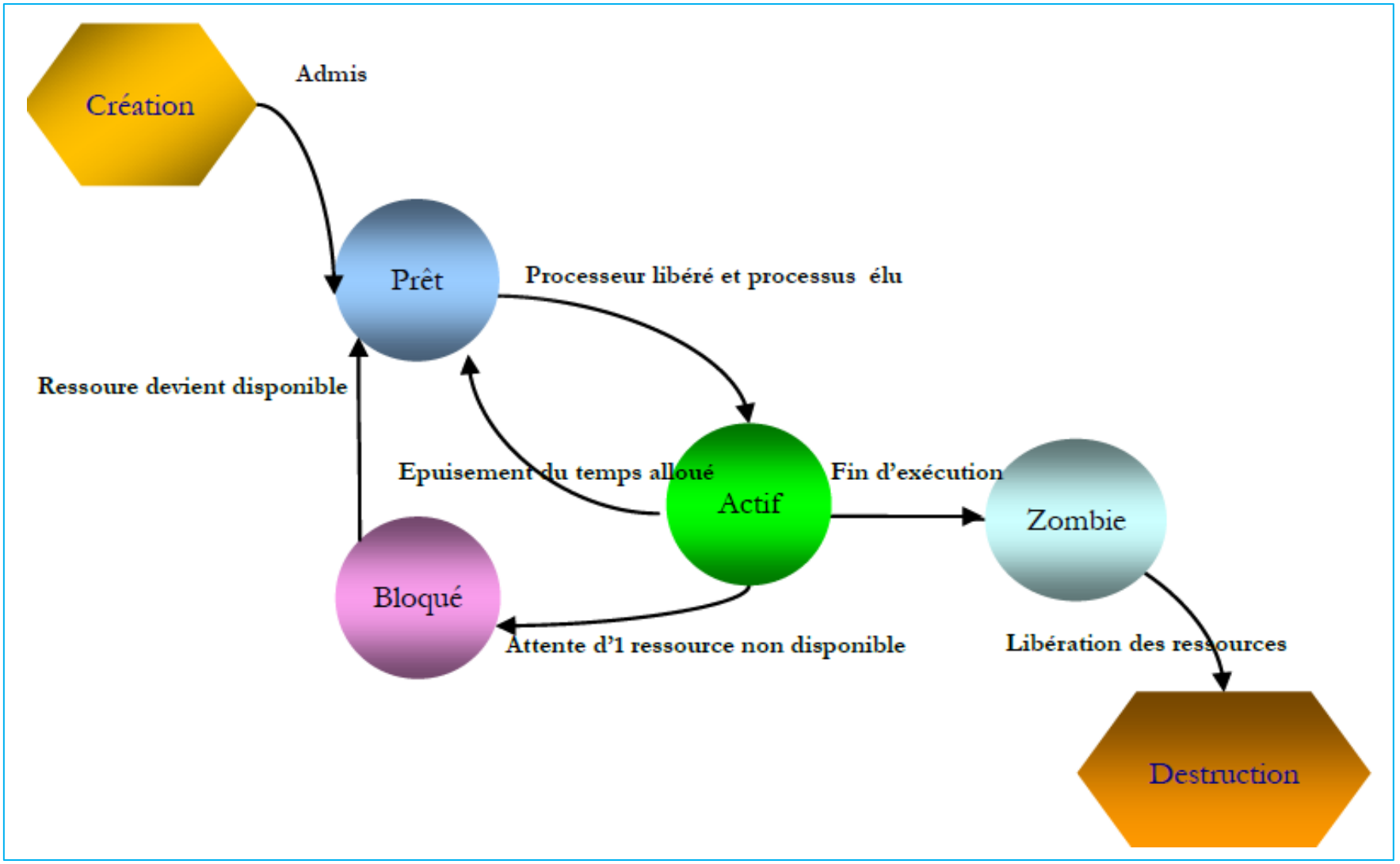
Un processus est une entité dynamique et on peut étudier son état au cours du temps. Typiquement, on recense trois états de processus :

Un processus est soit :

- **prêt** (suspendu par le système d'exploitation)
- **élu** (en exécution)
- **bloqué** (en attente d'un événement quelconque pour poursuivre)



Cycle de vie d'un processus



Modes de lancement de processus

Processus en avant plan (foreground)

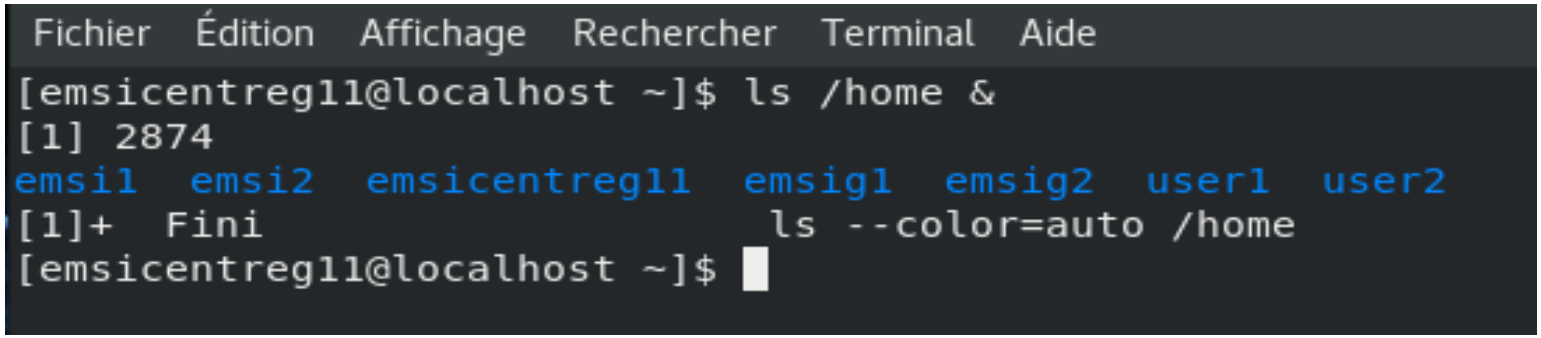
Lorsqu'on lance une commande, le Shell doit attendre, jusqu'à la fin de l'exécution de la commande avant l'exécution d'une deuxième commande, pour retrouver le prompt de nouveau. On appelle ce processus, un processus en avant plan.

Processus en arrière plan (background)

Si on ajoute **&** derrière la commande, ceci permet de lancer un processus en arrière plan.
Le Shell n'attend plus la fin de son exécution. On dit qu'il lance la commande en arrière plan. Lorsqu'un utilisateur lance un processus en arrière plan, le Shell affiche entre crochets le numéro de tâche (job) et le PID du processus. Le premier processus lancé aura un numéro de tâche = 1.

Modes de lancement de processus

Exemple:



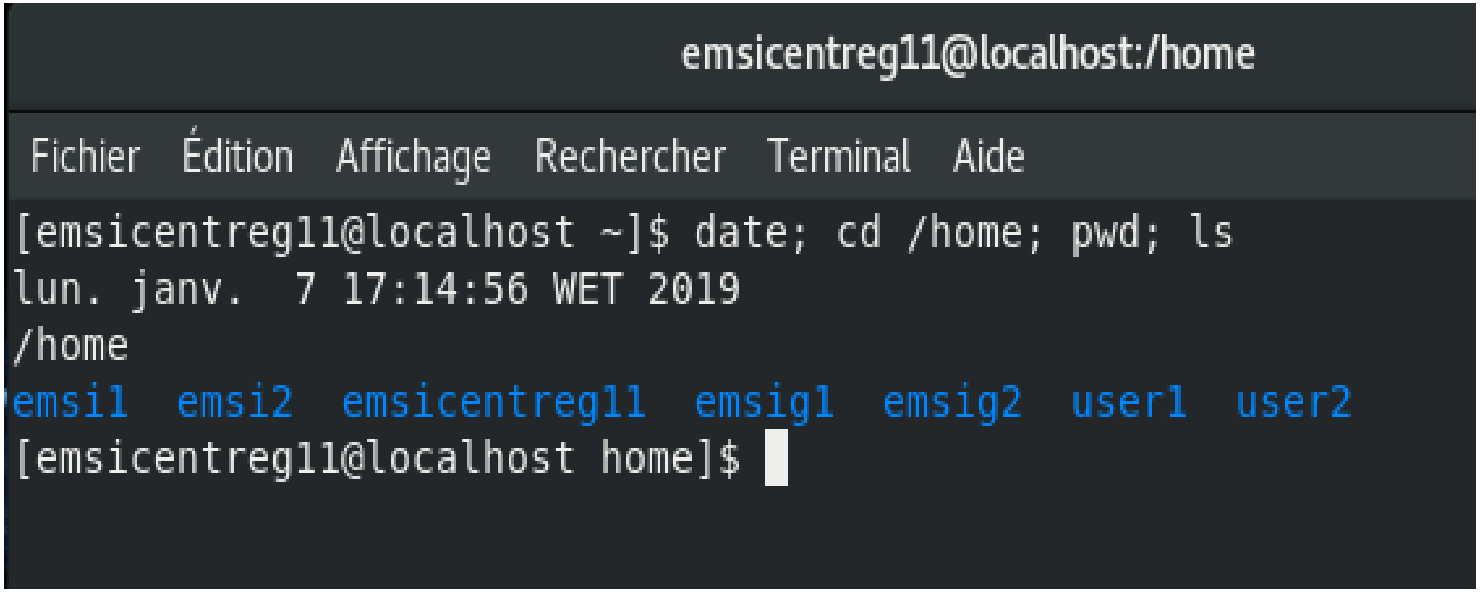
La commande: Wait

Lorsqu'un processus est en tâche de fond, il est possible de le récupérer et d'attendre la fin de la dernière commande lancée en arrière-plan avec la commande :

$wait [pid]

Enchaînements séquentiels de processus

Les commandes peuvent être enchaînées séquentiellement en utilisant un point virgule.
Citons par exemple la ligne de commande suivante :



Commandes Shell de manipulation de Processus

La commande: PS

PS

Syntaxe: **ps** [Options]

☛ La commande ps (process) liste les processus de l'utilisateur.

Options:

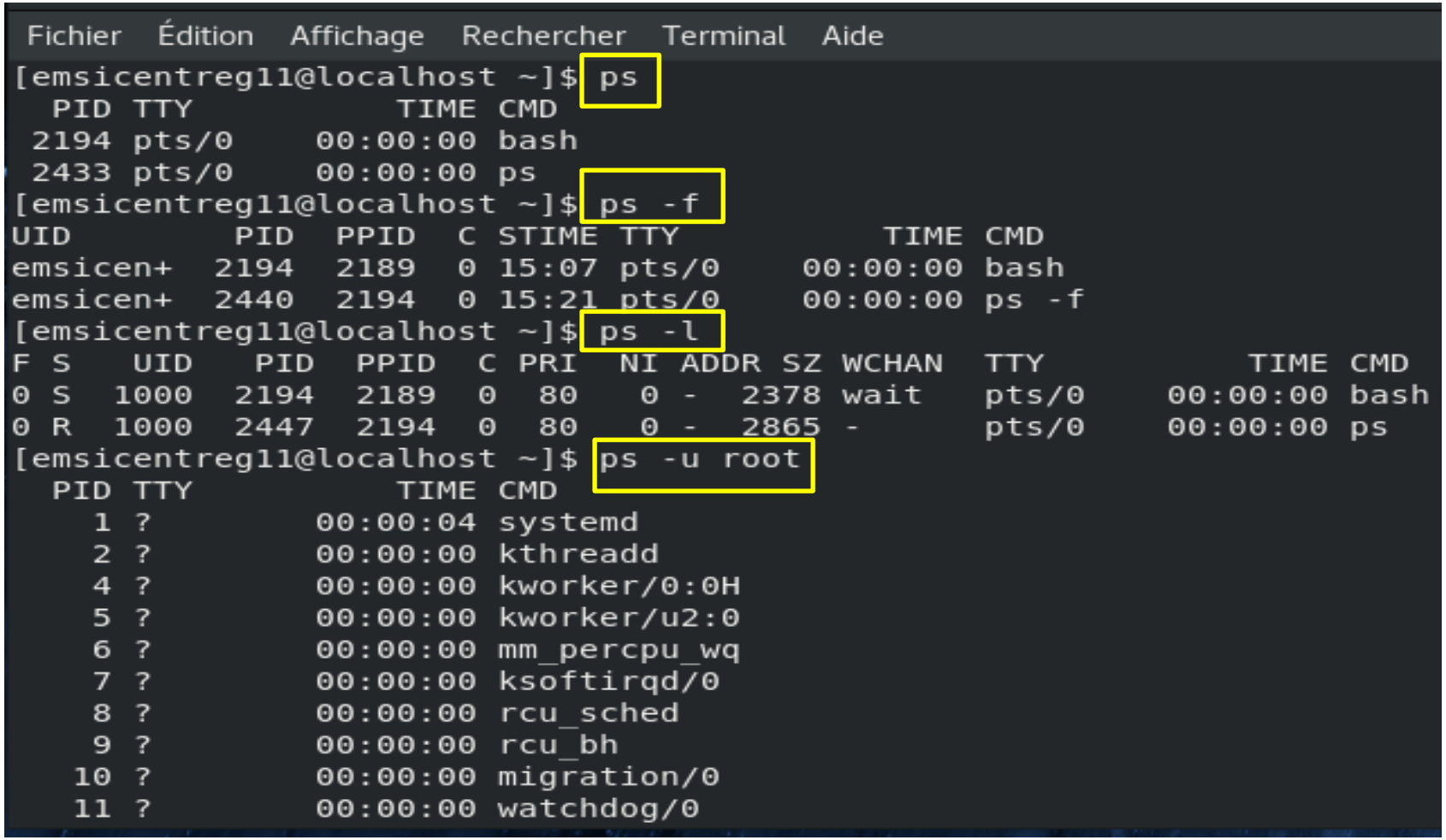
-e : affiche tous les processus en cours d'exécution sur un ordinateur.

–l et **–f**: affichent des informations détaillées.

-u: permet d'afficher les processus lancés par un utilisateur particulier.

La commande: PS

Exemples Commande PS:



La commande: PS

Les Informations présentées par ps -f

- **UID:** nom de l'utilisateur qui a lancé le processus.
- **PID:** correspond au numéro du processus.
- **PPID:** correspond au numéro du processus parent.
- **C:** au facteur de priorité : plus la valeur est grande, plus le processus est prioritaire
- **STIME:** correspond à l'heure de lancement du processus
- **TTY**: correspond au nom du terminal
- **TIME:** correspond à la durée de traitement du processus
- **CMD:** correspond au nom du processus.

La commande: Kill

Kill

Syntaxe: **kill [-l] -Num_signal PID1 [PID2...]**

☛ Envoyer un signal à un processus, et ce signal est identifié par un numéro.

☛Contrairement à ce que son nom semble indiquer, le rôle de cette commande n'est pas forcément de détruire ou de terminer un processus, mais d'envoyer des signaux aux processus.

La commande: Kill

Remarques:

❖ Le signal est l'un des moyens de communication entre les processus. Lorsqu'on envoie un signal à un processus, il doit l'intercepter et réagir en fonction de celui-ci. Certains signaux peuvent être ignorés, d'autres non.

❖ Un processus peut recevoir plusieurs signaux engendrés par la commande **kill**, ou bien par l'utilisateur en frappant sur des touches du clavier ou causés par des erreurs matérielles ou logicielles.

❖L'utilisateur peut utiliser des touches pour tuer un processus en avant plan en cliquant sur **CTRL^C** ou le suspendre en cliquant **CTRL^Z**.

La commande: Kill

```
kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

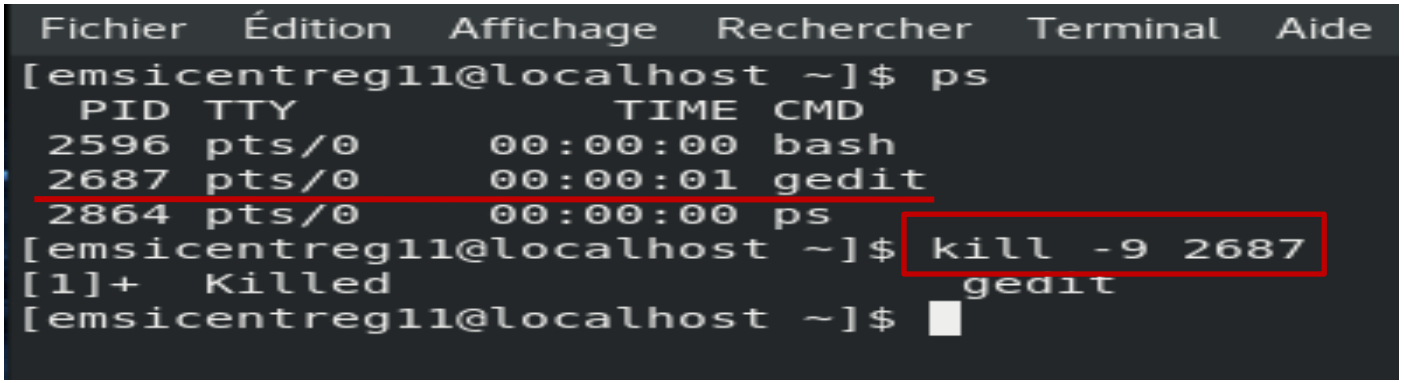
Les signaux sont numérotés et nommés, la liste des signaux peut être visualisée en appliquant l'option **–l** à la commande **kill**.

Pour tuer un processus, on doit connaître son PID, et écrire la commande suivante :

kill -9 <PID>

La commande: Kill

Exemple Commande Kill:



❖Ci-dessous les numéros, les noms et la description de quelques signaux que peuvent recevoir un processus :

Numéro	Nom	Description
9	KILL	Un signal qui va tuer à coup sûr le processus qui le reçoit.
15	TERM	(Terminate) Envoyé à un processus pour le terminer de façon élégante si possible.
19	STOP	Ce signal permet de suspendre un processus.

La commande: jobs

jobs

Syntaxe: **jobs [-l]**

☛ La commande jobs permet d'afficher la liste des tâches du Shell courant (suspendus ou s'exécutant en tâche de fond) avec leurs numéros de tâches, les PIDs et les états des processus.

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[emsicentreg11@localhost ~]$ find / -type f -name "k*" > resultat 2> erreurs &
[2] 3050
[emsicentreg11@localhost ~]$ ps
  PID TTY          TIME CMD
 2596 pts/0    00:00:00 bash
 2905 pts/0    00:00:00 gedit
 3050 pts/0    00:00:00 find
 3059 pts/0    00:00:00 ps
[emsicentreg11@localhost ~]$ jobs
[1]-  En cours d'exécution   gedit &
[2]+  En cours d'exécution   find / -type f -name "k*" > resultat 2> erreurs &
[emsicentreg11@localhost ~]$
```

La commande: jobs

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[emsicentreg11@localhost ~]$ gedit &
[1] 2284
[emsicentreg11@localhost ~]$ find / -type f -name "e*" > resultat 2> erreurs &
[2] 2355
[emsicentreg11@localhost ~]$ ps
  PID TTY          TIME CMD
 2211 pts/0    00:00:00 bash
 2284 pts/0    00:00:04 gedit
 2355 pts/0    00:00:00 find
 2363 pts/0    00:00:00 ps
[emsicentreg11@localhost ~]$ kill -19 2355

[2]+  Stoppé                 find / -type f -name "e*" > resultat 2> erreurs
[emsicentreg11@localhost ~]$ jobs
[1]-  En cours d'exécution   gedit &
[2]+  Stoppé                 find / -type f -name "e*" > resultat 2> erreurs
[emsicentreg11@localhost ~]$ jobs -l
[1]-  2284 En cours d'exécution   gedit &
[2]+  2355 Stopped (signal)        find / -type f -name "e*" > resultat 2> erreu
rs
[emsicentreg11@localhost ~]$
```

Les commandes: fg et bg

fg

Syntaxe: **fg %n**

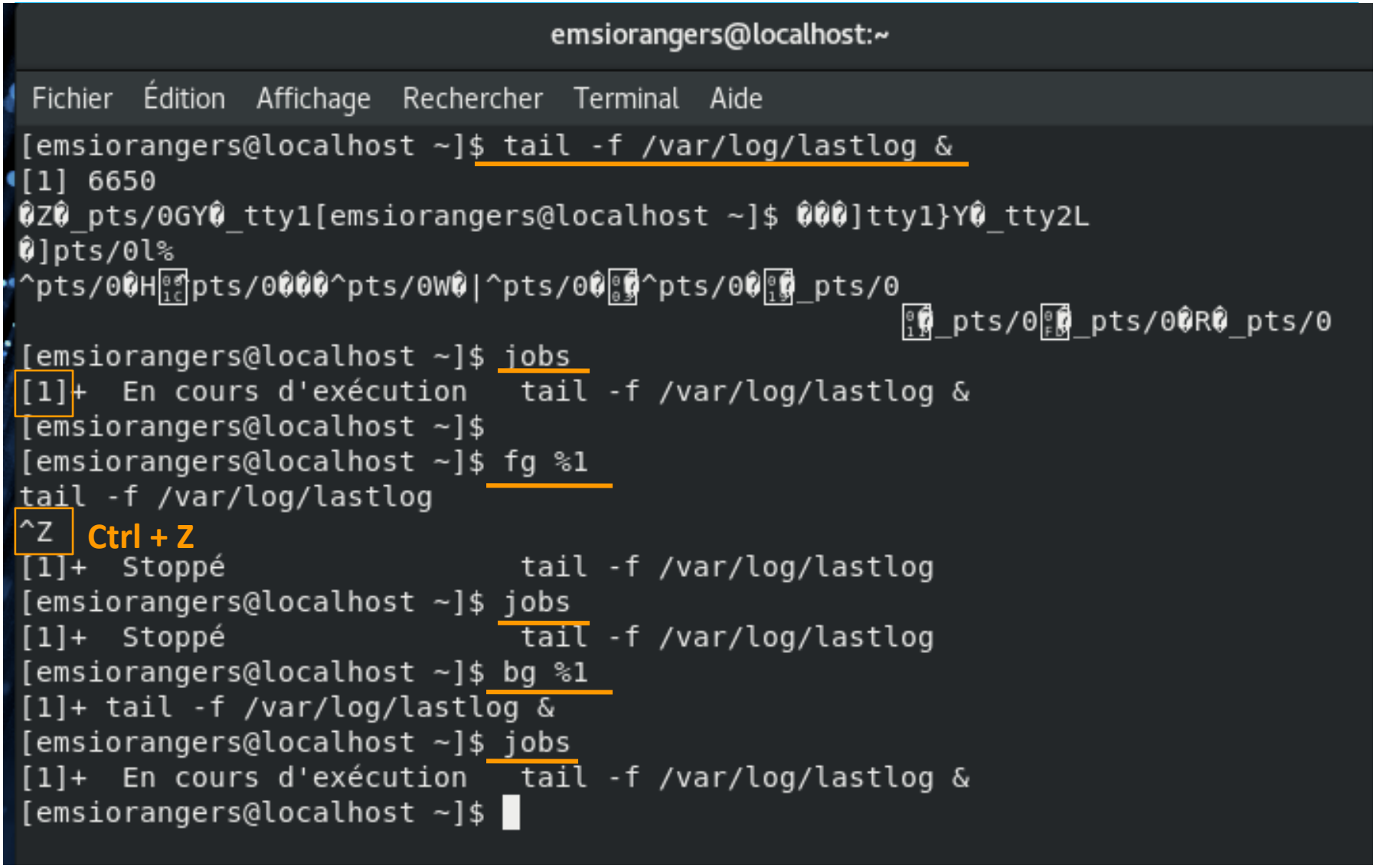
☛ Relancer l'exécution d'un processus en arrière plan en un processus en avant plan.
☛ **n** désigne le numéro de tâche.

bg

Syntaxe: **bg %n**

☛ Relancer l'exécution d'un processus suspendu en processus en arrière plan.
☛ **n** désigne le numéro de tâche.

Les commandes: fg et bg



La commande: top

top

Syntaxe: **top**

☛ La commande **top** vous permet d'afficher des informations en continu sur l'activité du système. Elle permet surtout de suivre les ressources que les processus utilisent (quantité de RAM, pourcentage de CPU, la durée de ce processus depuis son démarrage).

☛ Pour quitter l'utilitaire top, il suffit d'appuyer sur la touche "**q**".

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
top - 13:03:14 up 20 min,  1 user,  load average: 5,95, 6,35, 3,86
Tasks: 207 total,   2 running, 166 sleeping,   0 stopped,   0 zombie
%Cpu(s): 43,8 us, 14,9 sy, 17,8 ni,  0,0 id, 16,8 wa,  3,4 hi,  3,4 si,  0,0 st
KiB Mem :  1023764 total,    52600 free,   593652 used,   377512 buff/cache
KiB Swap:  2097148 total,  2033148 free,    64000 used.   398280 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 1578 emsicen+  20   0 1038228  96300  45260 S 24,9  9,4   0:39.81 gnome-shell
 1875 emsicen+  39  19  244980  36336  17240 R 22,3  3,5   0:51.74 tracker-ext+
 1901 emsicen+  20   0  131808  29848  11292 S 18,4  2,9   1:09.98 tracker-sto+
 2426 emsicen+  20   0   32300   5768   5308 S  4,9  0,6   0:16.22 gvfsd-metad+
 1458 emsicen+  20   0   16584   5212   3856 S  3,0  0,5   0:10.45 dbus-daemon
```

La commande: time

time

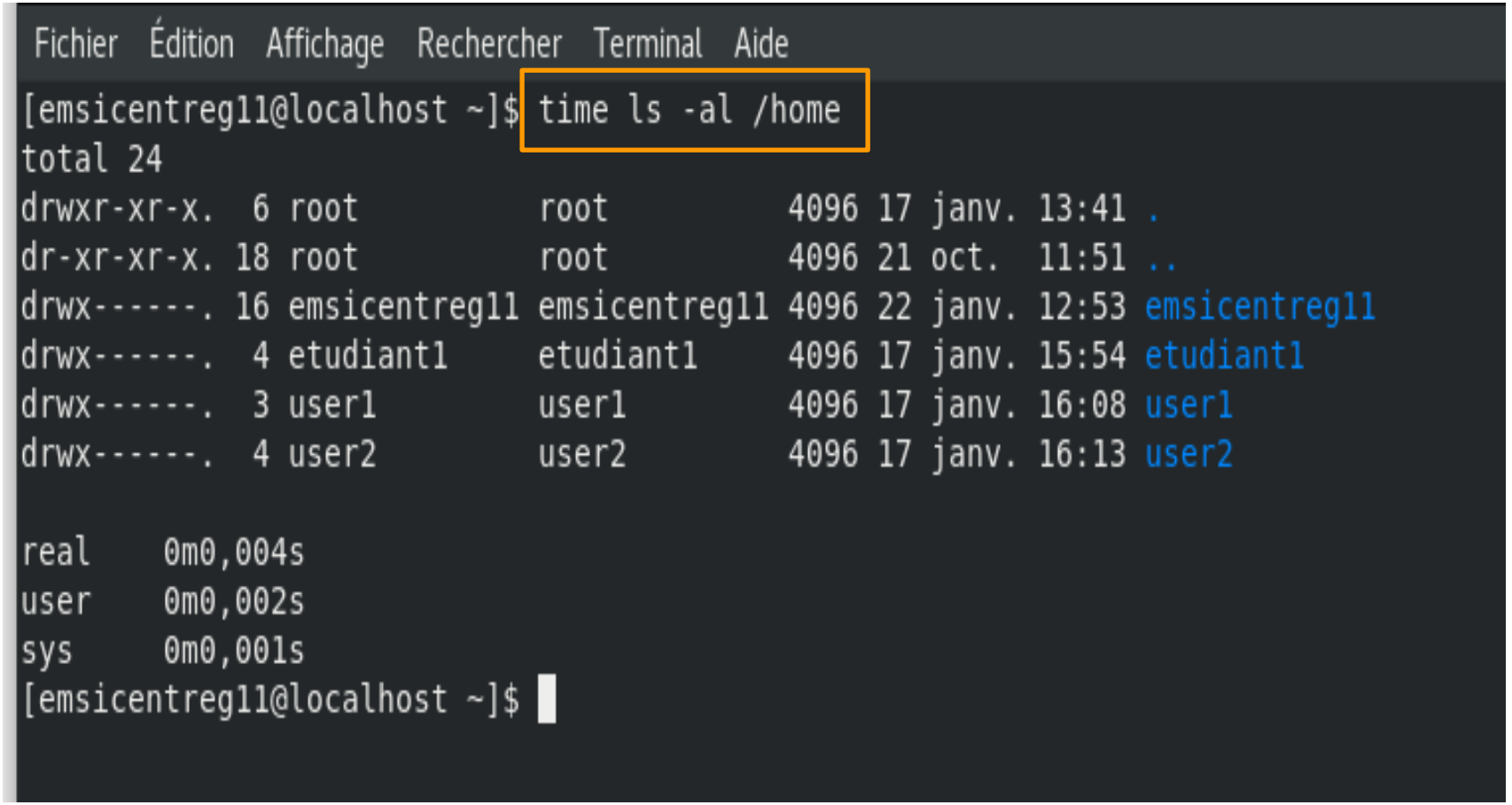
Syntaxe: **time**

☛ La commande time mesure les durées d'exécution d'une commande, idéale pour connaître les temps de traitement, et retourne trois valeurs :

- *real* : durée totale d'exécution de la commande
- *user* : durée du temps CPU nécessaire pour exécuter le programme
- *system* : durée du temps CPU nécessaire pour exécuter les commandes liées à l'OS (appels système au sein d'un programme).

☛ Si le résultat est inférieur à 10, la machine dispose de bonnes performances, au-delà de 20 la charge de la machine est trop lourde (performances basses).

La commande: time



Regroupement des processus

Il existe plusieurs méthodes pour enchaîner des commandes sur une même ligne :

Exécution sous condition d'erreur :

cmd1 || *cmd2* || *cmd3* || ... || *cmdN*

Si cmd1 ne se termine pas correctement, alors cmd2 est exécuté, et ainsi de suite. Sinon, si cmd1 s'exécute correctement les autres commandes ne seront pas exécutées.

Exécution sous condition de réussite :

cmd1 && cmd2 && cmd3 && ... && cmdN

Si cmd1 se termine correctement, alors cmd2 est exécuté, et ainsi de suite. Sinon, si cmd1 est exécutée avec erreur les autres commandes ne seront pas exécutées.

Regroupement des processus

Exemples:

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[emsicentreg11@localhost /]$ cd && ls && pwd
Bureau     erreurs  Modèles  Public  resultat         test1  TPRevision
Documents  Images   Musique  rep1    Téléchargements  test2  Vidéos
/home/emsicentreg11
[emsicentreg11@localhost ~]$
[emsicentreg11@localhost ~]$
[emsicentreg11@localhost ~]$ cd rep2 || mkdir rep2
bash: cd: rep2: No such file or directory
[emsicentreg11@localhost ~]$ ls
Bureau     erreurs  Modèles  Public  rep2      Téléchargements  test2       Vidéos
Documents  Images   Musique  rep1    resultat  test1           TPRevision
[emsicentreg11@localhost ~]$
```

Configuration de tâches automatisées

Le Service « CRON »

- **Cron** est un service ou démon (crond) permettant d'exécuter des tâches à des intervalles de temps réguliers. Il est présent sur tout système de type Unix et distributions GNU/linux.
- La configuration de **cron** se fait via des fichiers au format **crontab**, il existe un fichier par utilisateur.
- L'utilisateur pourra ajouter une action, en éditant son fichier crontab via la commande **crontab -e**.

crontab

crontab -l : permet de lister toutes les actions crontab de votre utilisateur .

crontab -u user -l : permet à l'administrateur de lister toutes les actions crontab de l'utilisateur user.

crontab -r : permet de supprimer votre fichier crontab .

crontab -e : permet d'éditer votre fichier crontab. Cela ouvrira l'éditeur de texte standard vi.

La commande crontab

Syntaxe des fichiers au format crontab:

Il existe 7 champs paramétrables par un nombre, une chaîne de caractères, séparés par un espace :

minute(1) heure(2) jour(mois)(3) mois(4) jour(semaine)(5) utilisateur(6) commande(7)

- (1) valeur comprise entre 0 et 59.
- (2) valeur comprise entre 0 et 23.
- (3) valeur comprise entre 1 et 31.
- (4) valeur comprise entre 1 et 12.
- (5) valeur comprise entre 0 et 7 (Dimanche étant le 0 ou le 7) ou alors les abréviations correspondant aux jours de la semaine en Anglais: sun, mon, tue, wed, thur, fri, sat.
- (6) uniquement pour les fichiers **crontab** du système dans /etc/cron.d/ .
- (7) commande à effectuer.

La commande crontab

Syntaxe des fichiers au format crontab:

Il existe aussi des caractères spéciaux :

***** : s'il est utilisé dans l'un des 5 premiers champs, indique que la commande doit être effectuée tout le temps ;

/ : permet de spécifier une répétition ;

- : permet de définir une plage ;

, : permet de définir plusieurs valeurs.

Exemples:

- Vider un répertoire tmp dans notre home toutes les heures:

```
* */1 * * *  rm -rf ~/tmp
```

- Lancer le script sauvegarde.sh régulièrement le mercredi et le vendredi à 4h00 du matin

```
0 4 * * 3,5 sauvegarde.sh
```

La commande crontab

Exemple:

emsiorangers@localhost:~

Fichier Édition Affichage Rechercher Terminal Aide

```
# le matin à 11:20 écrire "Bonjour Emsi 3IIR" dans le fichier F3IIR
 20  11  *   *   *  echo "Bonjour Emsi 3IIR" > /home/emsiorangers/RepCron/F3IIR
~
~
```

```
[emsiorangers@localhost ~]$ crontab -e
crontab: installing new crontab
[emsiorangers@localhost ~]$ clear
```

```
[emsiorangers@localhost ~]$ ls RepCron/
[emsiorangers@localhost ~]$ ls RepCron/
F3IIR
[emsiorangers@localhost ~]$ cat RepCron/F3IIR
Bonjour Emsi 3IIR
[emsiorangers@localhost ~]$
```

La commande at

- La commande **at** prévoit l'exécution d'une commande à un moment ultérieur.
- Elle prend l'horaire et la date prévus en paramètres sur sa ligne de commande, et la commande à exécuter sur son entrée standard.
- La commande sera exécutée comme si elle avait été saisie dans un interpréteur de commandes.
- **at** conserve d'ailleurs l'environnement courant afin de pouvoir travailler exactement dans les mêmes conditions que celles de la planification.
- L'horaire est indiqué en suivant les conventions habituelles : **16:12** représente 16 h 12. La date peut être précisée au format **JJ.MM.AA** (27.07.15 représentant ainsi 27 juillet 2015) ou **AAAA-MM-JJ**.
- En son absence, la commande sera exécutée dès que l'horloge atteindra l'heure signalée (le jour même ou le lendemain). On peut encore écrire explicitement today (aujourd'hui) ou tomorrow (demain).

La commande at

- Une autre syntaxe permet d'exprimer une durée d'attente : at now + nombre période.
- La période peut valoir minutes, hours (heures), days (jours) ou weeks (semaines). Le nombre indique simplement le nombre de ces unités qui doivent s'écouler avant exécution de la commande.

```
[emsiorangers@localhost ~]$ at 14:21
warning: commands will be executed using /bin/sh
at> mkdir repTestAt
at> echo "Bonjour 3IIR" > repTestAt/FileTestAt
at> <EOT>  Ctrl + D
job 5 at Mon Dec 14 14:21:00 2020
[emsiorangers@localhost ~]$ ls repTestAt
ls: impossible d'accéder à 'repTestAt': No such file or directory
[emsiorangers@localhost ~]$ ls repTestAt
FileTestAt
[emsiorangers@localhost ~]$ cat repTestAt/FileTestAt
Bonjour 3IIR
[emsiorangers@localhost ~]$
```

Les Alias

- Sur Linux, les **alias** sont des raccourcis de commandes jugées trop longues par l'utilisateur.
- En effet, le terminal est très pratique mais les commandes sont parfois lourdes et il devient facile de se tromper.
- Un **alias** permet également de gagner du temps en créant une commande courte pour une séquence que l'on tape fréquemment.

Méthode:

- Pour créer un alias en éditant le fichier **.bashrc**, vous devez avoir les droits nécessaires.
- Tout à la fin de votre fichier vous pouvez inscrire vos alias.
- Vous devez les écrire de la manière suivante :

alias nom_de_votre_alias='commande_a_executer'

Exemple : *alias installe='aptitude install'*

Les Alias

Exemple:

```
  GNU nano 2.9.8                                  .bashrc


# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
# Uncomment the following line if you don't like systemctl's auto-paging
# export SYSTEMD_PAGER=

# User specific aliases and functions
# .bashrc
# .bashrc
alias bonjour='echo "Bonjour 3IIR" ; echo "Bonne chance!"'
```

```
[emsiorangers@localhost ~]$ nano .bashrc
[emsiorangers@localhost ~]$ bonjour
Bonjour 3IIR
Bonne chance!
[emsiorangers@localhost ~]$
```