Reprenez le contrôle à l'aide de Linux !

30 heures Facile

Mis à jour le 29/06/2021

Exécuter des programmes en arrière-plan

Nous avons commencé à découvrir ce qu'étaient les processus dans le chapitre précédent. Nous savons désormais comment les lister, les trier, les filtrer et enfin comment les tuer.

Ici, je vous propose d'aller plus loin et de découvrir **l'exécution en arrière-plan**. A priori, la console a quelque chose de frustrant : on a l'impression qu'on ne peut lancer qu'un seul programme à la fois par console. Or, c'est tout à fait faux !
... Mais encore faut-il savoir comment faire tourner des programmes en arrière-plan.

Il existe un certain nombre de techniques plus ou moins sophistiquées. Il est recommandé de les connaître car, parfois, on souhaite tout faire au sein d'une seule et même console.

"&" & nohup : lancer un processus en arrière-plan

Lorsque vous vous apprêtez à lancer une opération un peu longue, comme une grosse copie de fichiers par exemple, vous n'avez peut-être pas envie de patienter sagement le temps que la commande s'exécute pour pouvoir faire autre chose en attendant.
Certes, on peut ouvrir une autre console me direz-vous. Il y a des cas cependant où l'on n'a accès qu'à une seule console, ou encore tout simplement pas envie d'en ouvrir une autre (la flemme, vous connaissez ? ;-).

Contrairement aux apparences, plusieurs programmes peuvent tourner en même temps au sein d'une même console. Ce n'est pas parce qu'on ne peut pas afficher plusieurs fenêtres comme

dans un environnement graphique qu'on est bloqué sur un seul programme à la fois ! Encore faut-il connaître les techniques qui permettent de lancer une commande en tâche de fond...

`&` : lancer un processus en arrière-plan

La première technique que je veux vous faire découvrir est très simple : elle consiste à rajouter le petit symbole `&` à la fin de la commande que vous voulez envoyer en arrière-plan.

Le symbole `&` s'appelle le « et commercial » ou encore l'« esperluette ». Il est présent sur la touche 1 d'un clavier AZERTY.

Prenons par exemple la commande `cp` qui permet de copier des fichiers.

Je vous propose de copier un gros fichier vidéo (ce qui prend en général du temps), comme ceci :

```
$ cp video.avi copie_video.avi &
[1] 16504
```

Notez que l'espace avant le `&` à la fin n'est pas obligatoire.

On vous renvoie deux informations.

- `[1]` : c'est le numéro du processus en arrière-plan dans cette console. Comme c'est le premier processus que nous envoyons en arrière-plan, il prend le numéro 1.
- `16504` : c'est le numéro d'identification général du processus (le fameux `PID` dont on a déjà parlé). Cette information vous permet de tuer le processus avec `kill` si nécessaire.

Maintenant, vous ne voyez peut-être rien, mais le processus est bel et bien en train de tourner en « tâche de fond ».

Si vous essayez de faire la même chose avec d'autres commandes, par exemple sur un `find`, vous risquez d'être surpris : les messages renvoyés par la commande s'affichent toujours dans la console ! Vous pouvez certes écrire du texte et lancer d'autres commandes pendant ce temps (essayez), mais c'est un peu frustrant de voir ces messages apparaître dans la console !

Heureusement, vous savez maintenant rediriger la sortie pour ne pas être importunés :

```
$ find / -name "*log" > sortiefind &
[1] 18191
```

Les résultats seront maintenant écrits dans le fichier `sortiefind` au lieu d'être affichés dans la console. De plus, la commande s'exécute en fond et ne nous importune plus.

Notez que pour être sûrs de ne pas être dérangés du tout, vous devrez aussi rediriger les erreurs (par exemple avec `2>&1`), ce qui peut nous donner une jolie commande comme celle-ci :

```
$ find / -name "*log" > sortiefind 2>&1 &
[1] 18231
```

Il reste toutefois un problème : le processus est « attaché » à votre console. Si vous fermez la console sur laquelle vous êtes, le processus sera tué et ne s'exécutera donc pas jusqu'au bout.

`nohup` : détacher le processus de la console

L'option `&` , bien qu'assez couramment utilisée, a ce défaut non négligeable : le processus reste attaché à la console, ce qui veut dire que si la console est fermée ou que l'utilisateur se déconnecte, le processus sera automatiquement arrêté.

Si on veut que le processus continue, il faut lancer la commande via `nohup` . Cela s'utilise comme ceci :

```
nohup commande
```

Par exemple, voici ce que ça donne si on lance la copie via un `nohup` :

```
$ nohup cp video.avi copie_video.avi
nohup: ajout à la sortie de `nohup.out'
```

La sortie de la commande est par défaut redirigée vers un fichier `nohup.out` . Aucun message ne risque donc d'apparaître dans la console.

D'autre part, la commande est maintenant immunisée contre la fermeture de la console. Elle continuera de fonctionner quoi qu'il arrive (sauf si on lui envoie un `kill` , bien sûr).

`nohup` est très utile par exemple lorsque vous vous connectez à un serveur. Imaginons que vous voulez lancer un programme (comme un serveur de jeu) : celui-ci s'arrêtera de fonctionner dès que vous vous serez déconnectés de la ligne de commandes du serveur. Vous n'allez pas rester connectés juste pour que le programme continue à fonctionner ! Heureusement, `nohup` vous préserve de ce problème.

Ctrl + Z, jobs, bg & fg : passer un processus en arrière-plan

Voyons maintenant le problème différemment : vous avez lancé la commande sans penser à rajouter un petit `&` à la fin. Malheureusement, la commande prend beaucoup plus de temps à s'exécuter que ce que vous aviez prévu. Êtes-vous condamnés à attendre qu'elle soit terminée pour reprendre la main sur l'invite de commandes ? Bien sûr que non !

Il y a une série de commandes et de raccourcis qu'il vous faut absolument connaître ! Nous allons les étudier un par un dès maintenant.

`Ctrl + Z` : mettre en pause l'exécution du programme

Reprenons le cas de notre grosse copie de fichiers. Cette fois, je suppose que vous l'avez lancée sans le petit symbole `&` :

```
$ cp video.avi video_copie.avi
```

Si vous n'avez pas de gros fichier sous la main pour faire le test, vous pouvez aussi faire un `top` .

Tapez maintenant `Ctrl + Z` pendant l'exécution du programme. Celui-ci va s'arrêter et vous allez immédiatement reprendre la main sur l'invite de commandes.

```
[1]+  Stopped                 top
mateo21@mateo21-desktop:~$
```

Vous noterez que nous avons plusieurs informations : le numéro du processus en arrière-plan (ici `[1]`), son état (`Stopped`) et le nom de la commande qui a lancé ce processus.

Le processus est maintenant dans un état de pause. Il ne s'exécute pas mais reste en mémoire.

`bg` : passer le processus en arrière-plan (*background*)

Maintenant que le processus est en « pause » et qu'on a récupéré l'invite de commandes, tapez :

```
$ bg
[1]+ top &
```

C'est tout, pas besoin de paramètre.

Qu'est-ce que cela fait ? Cela commande la reprise du processus, mais cette fois en arrière-plan. Il continuera à s'exécuter à nouveau, mais en tâche de fond.

En résumé, si vous avez lancé une commande par erreur en avant-plan et que vous voulez récupérer l'invite de commandes, il faudra faire dans l'ordre :

- `Ctrl + Z` : pour mettre en pause le programme et récupérer l'invite de commandes ;
- `bg` : pour que le processus continue à tourner mais en arrière-plan.

`jobs` : connaître les processus qui tournent en arrière-plan

Vous pouvez envoyer autant de processus en arrière-plan que vous voulez au sein d'une même console :

- soit en les lançant directement en arrière-plan avec un `&` à la fin de la commande ;
- soit en utilisant la technique du `Ctrl + Z` suivi de `bg` que vous venez d'apprendre.

Comment savoir maintenant quels sont les processus qui tournent en arrière-plan ? Vous pourriez, certes, recourir à la commande `ps` , mais celle-ci vous donnera tous les processus. C'est un peu trop.

Heureusement, il existe une commande qui liste uniquement les processus qui tournent en fond au sein d'une même console : `jobs` .

```
$ jobs
[1]-  Stopped                 top
[2]+  Stopped                 find / -name "*log*" > sortiefind 2>&1
```

Encore une fois, vous avez le numéro du processus qui tourne en fond (à ne pas confondre avec le `PID`), son état et son nom.

`fg` : reprendre un processus au premier plan (*foreground*)

La commande `fg` renvoie un processus au premier plan.

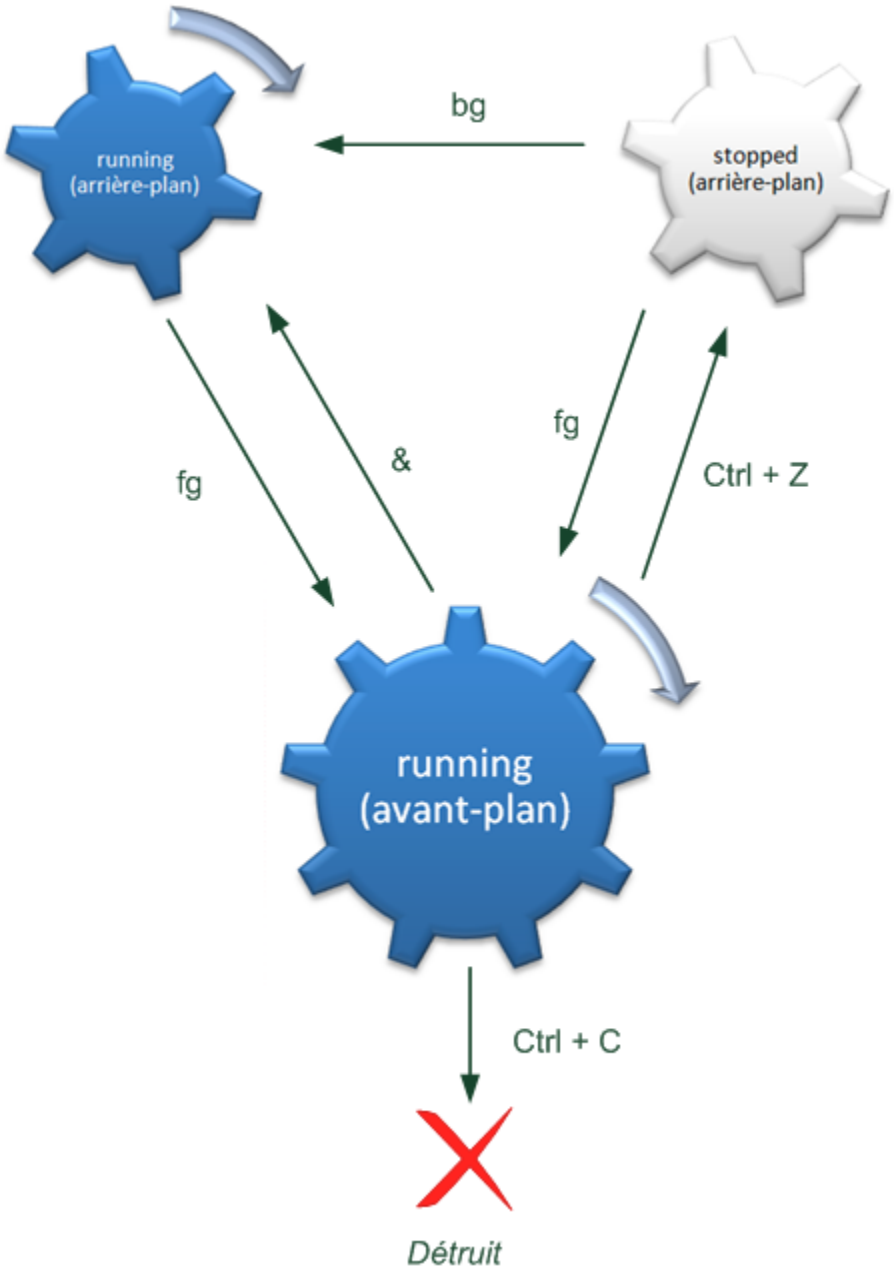
```
$ fg
```

Si vous avez un seul processus listé dans les `jobs` , c'est ce processus qui sera remis au premier plan.
Si, comme moi tout à l'heure, vous avez plusieurs processus en arrière-plan, il faudra préciser lequel vous voulez récupérer. Par exemple, voici comment reprendre le `find` qui était le `job` n° 2 :

```
$ fg %2
```

Résumé des états possibles des processus

Je pense qu'un schéma s'impose maintenant. Dans la figure suivante, je résume tout ce que nous avons vu jusqu'ici, à l'exception de `nohup` qui est une commande un peu à part.



Expliquons un peu ce schéma !

Par défaut, un processus est lancé dans l'état `running` à l'avant-plan. On peut l'arrêter avec la combinaison `Ctrl + C` , auquel cas il sera détruit.

Mais on peut aussi l'envoyer en arrière-plan. Si on l'exécute dès le départ avec un `&` , il sera à l'état `running` à l'arrière-plan. Si on choisit de faire `Ctrl + Z` , il passera à l'état `Stopped` à l'arrière-plan. Il faudra taper `bg` pour le faire passer à nouveau à l'état `running` en arrière-plan.

Enfin, la commande `fg` renvoie un processus de l'arrière-plan vers l'avant-plan.

Prenez cinq minutes pour bien analyser ce schéma et vérifier que vous avez compris l'essentiel de ce chapitre, c'est vraiment important. Il résume à peu près tout ce qu'il faut savoir. Il manque seulement `nohup` que j'ai mis à part comme je vous l'ai dit.

screen : plusieurs consoles en une

Il nous reste à découvrir une commande un peu particulière que j'ai volontairement réservée pour la fin : `screen` .
Pourquoi ai-je attendu avant d'en parler ? Tout simplement parce que, contrairement à ce que nous avons vu jusqu'ici, ce n'est pas une commande « standard » qui est installée par défaut sur toutes les distributions Linux. Parfois, vous n'aurez pas accès à `screen` (parce que vous n'êtes pas root sur la machine) et il faudra vous débrouiller avec les commandes que l'on vient de voir.

Si toutefois vous êtes les maîtres de la machine (ce qui est votre cas si vous avez installé Linux chez vous), je peux vous recommander d'installer le programme `screen` .

```
$ sudo apt-get install screen
```

De quoi s'agit-il ?
`screen` est un multiplicateur de terminal. Derrière ce nom un peu pompeux qui peut faire peur – je le reconnais – se cache en fait un programme capable de gérer plusieurs consoles au sein d'une seule, un peu comme si chaque console était une fenêtre !

Imaginez que `screen` est un programme qui permet entre autres de faire une **mise en veille prolongée** de votre console, tout comme vous le faites peut-être avec votre ordinateur portable qui se retrouve exactement dans l'état où vous l'avez laissé en l'éteignant.
Concrètement, j'ai souvent tendance à utiliser `screen` sur un serveur. Cela me permet par exemple de lancer un serveur de jeu dans une console `screen` , de quitter le serveur, puis de revenir l'administrer plus tard au besoin en récupérant la console dans l'état où je l'ai laissée.

Lorsque vous avez installé `screen` , essayez-le en tapant tout simplement :

```
$ screen
```

Un message s'affiche, précisant tout d'abord que le programme est un logiciel libre. Il indique ensuite l'adresse e-mail de l'auteur à laquelle on peut envoyer, je cite « des t-shirts, de l'argent, de

la bière et des pizzas ». Bon... passons. :-)

```
Screen version 4.00.03 (FAU) 23-Oct-06

Copyright (c) 1993-2002 Juergen Weigert, Michael Schroeder
Copyright (c) 1987 Oliver Laumann

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
this program (see the file COPYING); if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.

Send bugreports, fixes, enhancements, t-shirts, money, beer & pizza to
screen@uni-erlangen.de

                         [Press Space or Return to end.]
```

Tapez `Entrée` ou `Espace` pour passer ce message.

À première vue, il ne se passe rien de bien extraordinaire : on retrouve une console vide. Mais mine de rien, nous nous trouvons dans une console « émulée », non pas dans la « vraie » console où nous étions tout à l'heure. Vous pouvez en sortir en tapant `Ctrl + D` ou `exit` , comme si vous quittiez une console normalement.
Vous retrouverez alors votre console habituelle où vous avez lancé `screen` :

```
mateo21@mateo21-desktop:~$ screen
[screen is terminating]
```

Bon, maintenant que vous savez sortir de `screen` , retournez-y. :-)

Il faut savoir que sous `screen` , tout se fait à partir de combinaisons de touches de la forme suivante : `Ctrl + a + autre touche` . En fait, vous devez taper `Ctrl + a` , relâcher ces touches (lever les mains du clavier) et ensuite appuyer sur une autre touche.

`Ctrl + a` puis `?` : afficher l'aide

Essayez de taper `Ctrl + a` , puis tapez `?` . L'aide devrait alors s'afficher :

```
Screen key bindings, page 1 of 2.

                    Command key:  ^A   Literal ^A:  a

  break       ^B b         license     ,            removebuf   =
  clear       C            lockscreen  ^X x         reset       Z
  colon       :            log         H            screen      ^C c
  copy        ^[ [         login       L            select      '
  detach      ^D d         meta        a            silence     _
  digraph     ^V           monitor     M            split       S
  displays    *            next        ^@ ^N sp n   suspend     ^Z z
  dumptermcap .            number      N            time        ^T t
  fit         F            only        Q            title       A
  flow        ^F f         other       ^A           vbell       ^G
  focus       ^I           pow_break   B            version     v
  hardcopy    h            pow_detach  D            width       W
  help        ?            prev        ^H ^P p ^?   windows     ^W w
  history     { }          quit        \            wrap        ^R r
  info        i            readbuf     <            writebuf    >
  kill        K k          redisplay   ^L l         xoff        ^S s
  lastmsg     ^M m         remove      X            xon         ^Q q

                 [Press Space for next page; Return to end.]
```

Il y a deux pages de commandes. Avec `Espace` vous allez à la page suivante ; avec `Entrée` , vous refermez l'aide.

Comment lire cette page d'aide ? Par exemple, si vous voulez connaître la version du programme (milieu de la troisième colonne), il faudra taper `Ctrl + a` suivi de `v` (la lettre minuscule). Toutes les touches que vous voyez là doivent impérativement être précédées d'un `Ctrl + a` .
Notez par ailleurs que l'accent circonflexe `^` signifie ici `Ctrl` .

Les principales commandes de `screen`

Je ne connais pas toutes ces commandes, mais je vais vous en présenter les principales, celles qui selon moi peuvent vous être utiles.

- `Ctrl + a` puis `c` : créer une nouvelle « fenêtre ».
- `Ctrl + a` puis `w` : afficher la liste des « fenêtres » actuellement ouvertes. En bas de l'écran vous verrez par exemple apparaître : `0-$ bash  1*$ bash` . Cela signifie que vous avez deux fenêtres ouvertes, l'une numérotée 0, l'autre 1. Celle sur laquelle vous vous trouvez

actuellement contient une étoile `*` (on se trouve donc ici dans la fenêtre n° 1).

- `Ctrl + a` puis `A` : renommer la fenêtre actuelle. Ce nom apparaît lorsque vous affichez la liste des fenêtres avec `Ctrl + a` puis `w` .
- `Ctrl + a` puis `n` : passer à la fenêtre suivante (*next*).
- `Ctrl + a` puis `p` : passer à la fenêtre précédente (*previous*).
- `Ctrl + a` puis `Ctrl + a` : revenir à la dernière fenêtre utilisée.
- `Ctrl + a` puis un chiffre de 0 à 9 : passer à la fenêtre n° X.
- `Ctrl + a` puis `"` : choisir la fenêtre dans laquelle on veut aller.
- `Ctrl + a` puis `k` : fermer la fenêtre actuelle (*kill*).

`screen` est sensible à la casse pour les commandes ! Faites donc bien la différence entre « c » et « C » par exemple.

Il nous reste deux options très intéressantes de `screen` à découvrir et qui méritent une attention particulière : `split` et `detach` .

`Ctrl + a` puis `S` : découper `screen` en plusieurs parties (*split*)

`Ctrl + a` puis `S` coupe l'écran en deux pour afficher deux consoles à la fois (*split*). Il est possible de répéter l'opération plusieurs fois pour couper en trois, quatre, ou plus (dans la mesure du possible, parce qu'après les consoles sont toutes petites).

Voici, en figure suivante, ce que vous voyez après avoir *splitté* l'écran une fois.

L'écran est bien découpé en deux, mais la « fenêtre » du bas est vide. Il n'y a même pas d'invite de commandes.

Pour passer d'une fenêtre à une autre, faites `Ctrl + a` puis `Tab` .
Une fois le curseur placé dans la fenêtre du bas, vous pouvez soit créer une nouvelle fenêtre (`Ctrl + a` puis `c`) soit appeler une autre fenêtre que vous avez déjà ouverte (avec `Ctrl + a` puis un chiffre, par exemple).

Vous pourrez, comme dans la figure suivante, afficher par exemple `top` pendant que vous faites des opérations sur la fenêtre du dessus.

```
mateo21@mateo21-desktop:~$ ls
bin              Desktop             ies4linux-latest.tar.gz  tuto
copie_video.avi  Examples            Images                   video.avi
cours_unix.txt   ies4linux-2.99.0.1  sortiefind               video_copie.avi
mateo21@mateo21-desktop:~$ rm copie_video.avi
mateo21@mateo21-desktop:~$
```

```
 0 bash
top - 21:34:26 up  3:39,  3 users,  load average: 0.15, 0.18, 0.12
Tasks:  99 total,   1 running,  97 sleeping,   1 stopped,   0 zombie
Cpu(s):  0.7%us,  1.0%sy,  0.0%ni, 98.0%id,  0.0%wa,  0.3%hi,  0.0%si,  0.0%st
Mem:    515984k total,   480984k used,    35000k free,   111412k buffers
Swap:   240932k total,    33820k used,   207112k free,   197040k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5910 root      15   0  2444  984  784 S  0.7  0.2   0:17.45 vmware-guestd
 5680 root      15   0 41216  16m 6440 S  0.3  3.2   1:16.42 Xorg
20582 mateo21   15   0  2320 1144  880 R  0.3  0.2   0:00.05 top
    1 root      20   0  2912 1844  524 S  0.0  0.4   0:00.97 init
 1 bash
```

La classe de geek, quoi. ;-)

Ah, et pour fermer une fenêtre que vous avez *splittée*, il faudra taper `Ctrl + a` puis `X` . Voilà, vous savez l'essentiel !

`Ctrl + a` puis `d` : détacher `screen`

`Ctrl + a` puis `d` détache `screen` et vous permet de retrouver l'invite de commandes « normale » sans arrêter `screen` . C'est peut-être une des fonctionnalités les plus utiles que nous devons approfondir, et cela nous ramène d'ailleurs à l'exécution de programmes en arrière-plan dont nous avons parlé au début du chapitre.

Concrètement, si vous détachez `screen` , vous retrouvez l'invite de commandes classique :

```
mateo21@mateo21-desktop:~$ screen
[detached]
mateo21@mateo21-desktop:~$
```

L'information `[detached]` apparaît pour signaler que `screen` tourne toujours et qu'il est détaché de la console actuelle. Il continuera donc à tourner quoi qu'il arrive, même si vous fermez la console dans laquelle vous vous trouvez.

> Ah, alors c'est comme `nohup` finalement, non ?

En effet, `screen` se comporte comme un `nohup` . La différence est qu'une session `screen` vous permet d'ouvrir plusieurs fenêtres de console à la fois, contrairement à `nohup` qui ne peut lancer qu'un programme à la fois.

Vous pouvez donc partir, quitter la console et revenir récupérer votre session `screen` plus tard. Il faudra simplement taper :

```
$ screen -r
```

... pour retrouver votre session `screen` dans l'état où vous l'avez laissée.

Notez qu'il est possible de faire tourner plusieurs sessions `screen` en fond à la fois. Dans ce cas, `screen -r` ne sera pas suffisant car on vous demandera de préciser quelle session vous voulez récupérer :

```
$ screen -r
There are several suitable screens on:
        20930.pts-0.mateo21-desktop     (Detached)
        19713.pts-0.mateo21-desktop     (Detached)
Type "screen [-d] -r [pid.]tty.host" to resume one of them.
```

Pour récupérer la session n° 20930, tapez simplement :

```
$ screen -r 20930
```

À noter aussi que `screen -ls` affiche la liste des screens actuellement ouverts :

```
$ screen -ls
There are screens on:
        20930.pts-0.mateo21-desktop     (Detached)
        19713.pts-0.mateo21-desktop     (Detached)
2 Sockets in /var/run/screen/S-mateo21.
```

Certaines personnes ont pris l'habitude de tout faire sur `screen` , notamment sur les serveurs. Il m'est arrivé de laisser tourner une session `screen` pendant plusieurs mois grâce à la possibilité de détachement que nous venons de découvrir.

Un fichier personnalisé de configuration de `screen`

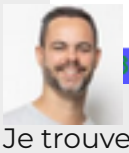
Sans rentrer dans le détail car ce serait bien trop long, sachez qu'il est possible de personnaliser `screen` avec un fichier de configuration, comme la plupart des autres programmes sous Linux d'ailleurs.

Ce fichier s'appelle `.screenrc` et doit être placé dans votre home (`/home/mateo21` par exemple). Vous pouvez vous amuser à lire la doc à ce sujet, mais vous pouvez aussi utiliser le même fichier `.screenrc` que j'ai l'habitude d'utiliser (ce fichier de configuration n'est pas de moi, merci donc à son auteur, « bennyben »).

Une fois placé dans votre home, exécutez `screen` . Vous devriez noter quelques différences, comme vous le montre la figure suivante.

```
mateo21@mateo21-desktop:~$ ls
bin             Examples                    Images      video.avi
cours_unix.txt  ies4linux-2.99.0.1          sortiefind  video_copie.avi
Desktop         ies4linux-latest.tar.gz     tuto
mateo21@mateo21-desktop:~$
```

Le professeur



Mathieu Nebra

Entrepreneur à plein temps, auteur à plein temps et co-fondateur d'OpenClassrooms :o)

Je trouve cette configuration plus pratique car on a toujours en bas l'heure, le nom de la machine sur laquelle on se trouve, la charge ainsi que la liste des fenêtres ouvertes. Après, libre à vous d'utiliser la configuration par défaut ou celle-là : dans tous les cas, les commandes restent les mêmes. ;-)

Découvrez aussi ce cours en...

Livre PDF

En bref

- Il est possible d'envoyer des programmes en arrière-plan dans la console afin de garder la main pour lancer de nouvelles commandes.
- Pour lancer un processus en arrière-plan, on peut ajouter le symbole `&` à la fin de la commande. En revanche, lorsque vous fermez la console, le processus est arrêté. Si vous

voulez qu'il continue, utilisez plutôt la commande `nohup` .

- Si vous avez lancé une commande normalement (en avant-plan) mais que celle-ci s'éternise, vous pouvez utiliser le raccourci `Ctrl + Z` pour la mettre en pause et récupérer la main. Si vous lancez la commande `bg` ensuite, elle reprendra son exécution en arrière-plan. Vous pourrez la récupérer au premier plan avec `fg` à tout moment.
- `screen` est un programme puissant que vous pouvez installer avec `apt-get` (il n'est pas présent par défaut). Il permet d'ouvrir plusieurs consoles virtuelles au sein d'une seule et même console, et donc d'exécuter facilement plusieurs processus en parallèle.

J'AI TERMINÉ CE CHAPITRE ET JE PASSE AU SUIVANT

SURVEILLER L'ACTIVITÉ DU SYSTÈME
EXÉCUTER UN PROGRAMME À UNE HEURE DIFFÉRÉE