

# Trucs et astuces

Il est un certain nombre de logiciels qu'un administrateur systèmes rencontrera au cours du temps. Ceux-ci lui permettront d'acquérir des automatismes et de gagner un temps considérable dans l'accomplissement de ses tâches.

Page aussi disponible sur <https://luc.frama.io/cours-asrall/tips/>.

## Éditer un fichier

Que votre éditeur favori soit *vim*, *nano*, *emacs* ou autre, il faut vous assurer :

- de connaître ses commandes (aller à la ligne X, faire un chercher/remplacer, réindenter)
- de savoir activer la coloration syntaxique. Ça paraît superfétatoire, mais c'est essentiel car cela vous permet de vous repérer plus rapidement dans le fichier voire de détecter des erreurs de syntaxe.

Il est nécessaire de connaître son éditeur pour être efficace.

Pour *vim*, vous pouvez lancer *vimtutor* qui vous fera passer par des exercices pour vous familiariser avec *vim*. Le site [VimCasts](https://vimcasts.org/) regorge de tutoriaux et d'astuces.

**Attention** : quand bien même vous ne choisiriez pas *vim*, il vous faut en connaître les commandes de base. En effet, *emacs* n'est que rarement installé sur un serveur, et *nano* est parfois (souvent ?) trop limité pour travailler vite et bien.

## Les processus

### htop

*htop* permet de lister les processus, rechercher un processus, tuer des processus, trier les processus selon différents critères...

Il affiche également des informations sur le système : occupation mémoire, utilisation des processeurs, charge du système, etc.

Pour n'afficher que les processus de l'utilisateur `foo` :

```
htop -u foo
```

Voir <https://peteris.rocks/blog/htop/> pour comprendre les informations fournies par *htop*.

[Carl Chenet](#) a traduit ces articles en français dans une série d'article disponible sur

<https://carlchenet.com/category/htop-explique/>.

## kill

La commande *kill* permet d'envoyer un signal à un processus. On peut indifféremment utiliser le n° ou le nom d'un signal pour l'utiliser. Ainsi `kill -9 <PID>` est normalement équivalent à `kill -KILL <PID>`.

Pour être bien certain du signal envoyé, il est préférable d'utiliser son nom : tous les signaux n'ont pas un n° attribué de façon certaine.

Voir [https://en.wikipedia.org/wiki/Unix\\_signal#POSIX\\_signals](https://en.wikipedia.org/wiki/Unix_signal#POSIX_signals) pour la liste des signaux POSIX.

## killall

*killall* est le petit frère de *kill*. Il permet d'envoyer des signaux aux processus sans connaître leur PID, juste avec leur nom.

Comme *killall* peut ratisser large, il vaut mieux lui préférer le couple *pgrep* / *pkill*.

## pgrep / pkill

*pgrep* permet de rechercher parmi les processus, *pkill* permet d'envoyer un signal aux processus avec la même syntaxe de recherche que *pgrep*.

Rechercher un processus par son nom :

```
pgrep nom
```

Rechercher un processus par l'intégralité de sa ligne de commande :

```
pgrep -f nom
```

Rechercher un processus par son nom, appartenant à l'utilisateur foo :

```
pgrep -u foo nom
```

Afficher le nom du processus en plus de son PID :

```
pgrep -l nom
```

Afficher la ligne de commande complète en plus de son PID :

```
pgrep -a nom
```

Envoyer le signal SIGTERM aux processus correspondants à la recherche :

```
pkill SIGTERM nom
```

## lsof

*lsof* permet de connaître le ou les processus utilisant une ressource.

Qui utilise `/home/foo` ?

```
lsof /home/foo
```

Qui utilise `/dev/sda` ?

```
lsof /dev/sda
```

Qui utilise le port 80 ?

```
lsof -i :80
```

## Les logs

### multitail

*multitail* permet de surveiller en temps réel les modifications d'un ou plusieurs fichiers à la manière d'un `tail -f` mais est bien plus souple d'usage.

Lire plusieurs fichiers :

```
multitail mail.log kern.log
```

Filtrer les lignes affichées d'un fichier selon une regex :

```
multitail -e regex mail.log kern.log
```

Filtrer les lignes affichées de *tous* les fichiers selon une regex :

```
multitail -E regex mail.log kern.log
```

Pour les données depuis l'entrée standard :

```
commande_qui_fait_des_logs | multitail -j
```

Une fois *multitail* lancé, un grand nombre de raccourcis claviers permet de le manipuler :

- **Entrée** : Affiche une ligne rouge avec l'heure et la date sur chaque fenêtre d'affichage de fichier (utile pour se donner un repère avant un test générant des logs)
- **O** (la lettre o en majuscule) : Efface l'affichage de toutes les fenêtres
- **/** : Effectue une recherche dans toutes les fenêtres
- **b** : Permet de revenir en arrière sur une fenêtre
- **F1** : affiche l'aide, avec tous les raccourcis claviers

## goaccess

*goaccess* va analyser en temps réel les logs d'un serveur pour fournir des statistiques.

On pourra alors voir rapidement quelle est l'adresse IP qui se connecte le plus, quelle est la page la plus visitée, etc.

## Veille technologique

Non, passer du temps sur [LinuxFR](#) ou sur le [Journal du hacker](#) n'est pas du temps perdu, quoi qu'on en dise. Il est en effet important d'effectuer une veille technologique régulière afin de découvrir de nouvelles technologies, de nouvelles astuces ou d'être averti de nouvelles failles de sécurité.

Votre meilleur ami pour cette veille sera un lecteur de flux RSS. En effet, un lecteur de flux a cet immense avantage sur les réseaux sociaux d'être asynchrone : partez en vacances deux semaines, revenez, et lisez tout ce que vous avez loupé (essayez un peu de faire cela avec Twitter : impossible). Vous pouvez aussi généralement le configurer pour qu'il vous envoie un résumé par mail de vos flux RSS... parfait quand on le couple à la liste de discussion des autres administrateurs systèmes !

Attention : les réseaux sociaux comme Twitter peuvent aussi être utiles, de par leur propension à propager (très) rapidement l'information. Le revers de la médaille est qu'il faudra bien vérifier la véracité de la-dite information.

# SSH

## Concierge

SSH fonctionne bien de base, mais avoir un fichier de configuration SSH améliore grandement les choses.

Exemple : votre identifiant sur votre machine locale est *rim*, mais *rimd* sur la machine *mavrick.chatons.org*. Pour vous connecter, vous lancez la commande `ssh rimd@mavrick.chatons.org`

Avec un fichier de configuration ssh (`~/.ssh/config`) contenant

```
Host mavrick
    HostName mavrick.chatons.org
    User rimd
```

vous pourrez vous connecter avec un simple `ssh mavrick`,

Avec quelques serveurs, la gestion de ce fichier ne pose pas de problème, mais on s'aperçoit, au fur et à mesure que l'on a plus de serveurs à gérer que cela devient une plaie. C'est là qu'intervient *concierge*.

[concierge](#) permet de gérer son fichier de configuration avec un langage de *template*.

On pourra donc écrire

```
{% for i in ('dorone', 'khais') %}
Host {{i}}
    HostName {{i}}.chatons.org
    User rimd
    IdentitiesOnly yes
    IdentityFile /home/%u/.ssh/id_chatons
{% endfor %}

{% for i in ('gohan', 'diren') %}
Host {{i}}
    HostName {{i}}.perso.org
    User rim
    IdentitiesOnly yes
    IdentityFile /home/%u/.ssh/id_perso
{% endfor %}
```

Ce qui créera des entrées dans le fichier de configuration SSH pour les serveurs *dorone*, *khais*, *gohan* et *diren*.

Voir <https://github.com/9seconds/concierge> pour l'installation de *concierge*.

## Mssh

*mssh*, disponible habituellement dans les dépôts de votre distribution préférée, vous permettra de lancer plusieurs connexions SSH en même temps. La fenêtre contiendra autant de terminaux que de connexions SSH. Les commandes tapées seront envoyées à tous les terminaux en même temps (il est possible de n'envoyer la commande que sur un seul serveur ou de "désactiver" certains serveurs pour que les commandes ne leur soient pas envoyées).

*mssh* est très utile pour effectuer des tâches simultanément.

On lance *mssh* ainsi : `mssh gohan diren`

## Confort visuel

### redshift

[redshift](#), lui aussi généralement dans les dépôts, ajuste la température de votre écran en fonction de l'heure. L'idée est de rougir graduellement l'écran afin d'éviter la fatigue visuelle due à la lumière bleue de votre écran.

## Confort dans le terminal

### bash-completion

Activer l'utilisation d'une complétion avancée des commandes se fait dans Debian en décommentant les lignes suivantes du fichier */etc/bash.bashrc* :

```
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
```

Cela permettra, par exemple, de compléter les options d'un logiciel, le nom d'un paquet à installer, etc. Sans cela, vous n'aurez que la complétion du logiciel que vous voulez utiliser et des chemins de votre système de fichiers.

## tree

*tree* affichera l'ensemble d'une arborescence... sous forme arborescente. Ce qui permet de parcourir un dossier très vite.

```
% tree foo
foo
├── bar
│   └── baz.txt

1 directory, 1 file
```

## Sécurité

### mkpasswd.pl

Fourni par le paquet *libstring-mkpasswd-perl* dans Debian, *mkpassword.pl* permet de générer des mots de passe aléatoires, éventuellement en forçant quelques paramètres.

```
% mkpasswd.pl -l 20 -s 4
kta*vvN:g7bxM/se8a-b
```

- `-l 20` : 20 caractères
- `-s 4` : avec 4 caractères spéciaux (ponctuation, pourcent, etc)

## Manipulation de données

### Sort

Le vénérable `sort` permet de trier des données.

```
sort < fichier.txt
```

Parmi les options intéressantes :

- `-u` supprime les doublons
- `-n` fait un tri numérique, c'est-à-dire que `2` vient avant `10`, là une machine dira que `10` vient avant `2` car le premier caractère `1` vient avant `2`
- `-h` fait un tri « humain » des nombres, c'est-à-dire qu'il va lire `2K` comme `2000`, et le triera après `1G`
- `-r` trie en sens inverse
- `-k` trie selon une clé. Par exemple, avec `foo bar`, `bar` est la clé n°2
- `-t` spécifie le séparateur des clés. Par défaut, la séparation est définie par une transition entre caractère vide (espace, tabulation, etc.) en non-vide

## Jq

Le logiciel `jq` permet de manipuler du [JSON](#) directement depuis la ligne de commande.

J'avoue, je l'utilise surtout pour récupérer juste l'info qu'il me faut, je modifie très rarement du JSON avec (mais on peut !)

```
jq '.job.status' < foo.json
```

## Yq

Le logiciel `yq` est au [YAML](#) ce que `jq` est au JSON. D'ailleurs, `yq` se sert de `jq` en interne.

```
yq '.job.status' < foo.yml
```

Note : `yq` n'aime pas les clés avec un `-`. Il faut alors utiliser une autre syntaxe que la classique `.key` :

```
yq '.job.["foo-bar"]' < foo.yml
```

## Divers

### ncdu

[ncdu](#) va regarder la taille du répertoire ciblé (celui où on se trouve par défaut) et afficher les fichiers/dossiers contenus, triés par taille. Très utile pour trouver ce qui bouffe de l'espace disque.

Petit bonus : pour avoir un *ncdu* sur un *bucket* S3, on peut utiliser le logiciel [rclone](#) ainsi :

```
rclone ncd� remote:path
```

## watch

*watch* permet de lancer une commande à intervalle régulier. Après une modification DNS, *watch dig chatons.org* pourra par exemple vous permettre de surveiller la prise en compte de cette modification sur votre résolveur.

## truncate

*truncate* permet de réduire ou étendre la taille d'un fichier à la taille indiquée.

```
truncate -s 1M fichier_trop_gros
```

## split

*split* permet de découper un fichier en plusieurs parties.

## wall

*wall* permet d'envoyer un message à tous les utilisateurs connectés à la machine.

---

Révision #6

Créé 23 janvier 2020 17:11:00 par Luc

Mis à jour 30 octobre 2024 12:09:38 par Luc