# Maintenir plusieurs machines Unix identiques

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 31 octobre 2007. Dernière mise à jour le 16 janvier 2008

https://www.bortzmeyer.org/maintenir-plusieurs-machines-identiques.html

---

Avec la disponibilité d'Unix pour des machines de bas prix comme les PC, l'ingénieur système a souvent besoin de maintenir N machines Unix identiques ou presque. Si N est de l'ordre de deux ou trois, cela peut encore se faire à la main, mais si N vaut vingt ou cent, cela n'est plus réaliste et il faut automatiser. Je présente ici la solution simpliste mais suffisante que j'utilise pour des salles temporaires, par exemple pour des formations comme les FFTI <http://www.afnic.fr/afnic/international/college/ffti>.

Le problème est connu depuis longtemps, même s'il n'avait pas été anticipé au début d'Unix, lorsqu'il y avait un seul ordinateur pour toute une université. Pour traiter ce problème, plusieurs solutions ont été proposées et la plus connue est certainement un programme très perfectionné, cfengine. cfengine permet de définir des **classes** de machine et de synchroniser les fichiers de configuration pour chaque classe. Il est très riche et je n'ai jamais eu le courage de l'apprendre. (Un logiciel du même genre est C3 <http://www.csm.ornl.gov/torc/C3/C3relatedresearch.shtml>, que me recommande Olivier Ricou.)

Ma solution est plus bête et elle repose sur des petits scripts shell simples qui copient des fichiers depuis un **maître** vers des **esclaves** (ou bien qui exécutent des commandes sur tous les esclaves). On configure donc à la main une machine et on pousse ensuite sa configuraion vers toutes les autres.

Tous ces scripts lisent leur configuration dans le fichier `config-pc` dont voici un contenu typique (ici, les quinze machines ont uniquement des adresses IPv6) :

```
start=131
end=145
prefix=2001:660:F108:1::
```

Les scripts dépendent de la commande Unix `seq` pour générer la liste des machines (notez que les différents scripts n'utilisent pas seq de la même manière, pour montrer la variété des possibilités) à partir d'un préfixe, d'un point de départ et d'un point d'arrivée. Cela nécessite donc que les adresses IP soient consécutives (pour une salle de TP, c'est une supposition raisonnable).

Ils utilisent ssh et, pour ne pas taper le mot de passe à chaque fois, il faut, avant tout, configurer les autorisations appropriées sur chaque machine (dans mon cas, en créant le fichier `root/.ssh/authorized_keys` dont le contenu est ma clé publique, que je trouve dans `/.ssh/id_dsa.pub`). Attention aux éventuels problèmes de sécurité que cela peut poser, comme par exemple le fait qu'il faut autoriser root en SSH (directive `PermitRootLogin` dans `sshd.conf`). Pour une salle de cours temporaire, ce n'est pas trop grave.

Voici l'utilisation du script `copy-each-pc` pour copier le fichier (ici `/tmp/example`) vers le répertoire indiqué (ici `/var/tmp`) :

```
% ./copy-each-pc /tmp/example /var/tmp
```

et le code du script :

```
#!/bin/sh

. ./config-pc

file=$1
destination=$2

if [ -z "$destination" ]; then
   echo "Usage: $0 file destination"
   exit 1
fi

for host in $(seq -f ${prefix}%g $start $end); do
   echo ""
   echo "Copying \"$file\" to $host:$destination..."
   scp $file root@\[$host\]:$destination
done
```

Notez les crochets autour de l'adresse IP pour `scp`, le deux-points étant utilisé également pour séparer l'adresse du répertoire (HTTP a le même problème donc un URL IPv6 ressemble à `http://[2001:660:f108` cf. RFC 2732 [1]). Ces crochets étant significatifs pour le shell, on doit faire un échappement avec la barre inverse.

Et voici l'utilisation du script `for-each-pc`, ici pour exécuter la commande `date` sur tous les PC :

```
% ./for-each-pc "date"
```

et le code du script est :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc2732.txt`

```
#!/bin/sh

. ./config-pc

command=$1

if [ -z "$command" ]; then
   echo "Usage: $0 command"
   exit 1
fi

for host in $(seq -f ${prefix}%g $start $end); do
   echo ""
   echo "Executing \"$command\" on $host..."
   ssh root@$host $command
done
```

Prenons maintenant un script un peu plus compliqué, le script `v6-only`, qui copie sur toutes les machines un fichier `interfaces` avec l'adresse IPv6 de la machine (oui, DHCP aurait été plus simple, mais moins sûr et il rend les machines dépendantes du serveur DHCP). Ce script est un peu plus compliqué car il copie un fichier dont le contenu dépend de la machine. Il appelle donc `sed` pour faire un remplacement dans le fichier maître, avant de copier le fichier créé par sed vers la machine :

```
#!/bin/sh

. ./config-pc

for i in `seq $start $end`; do
    sed "s/TOKEN/$i/" etc/interfaces-v6only > /tmp/interfaces
    echo "Copying to $prefix$i..."
    scp /tmp/interfaces root@\[$prefix$i\]:/etc/network/interfaces
    ssh root@$prefix$i reboot
done
```

Pour des scripts analogues, Ollivier Robert me suggère Dancer's Shell <http://www.netfort.gr.jp/~dancer/software/dsh.html.en> qui, en outre, permet le parallélisme des requêtes SSH et le contrôle de leur nombre. Olivier Perret propose un script qu'il a écrit, cdist (en ligne sur `https://www.bortzmeyer.org/files/cdist.sh`), qui se configure via un fichier de configuration, script dont il précise qu'il est livré sans aucune garantie. Michel Casabona suggère une version parallèle (mon script est purement séquentiel) de ssh, pssh <http://www.theether.org/pssh/>.