Comment générer une chaîne aléatoire

Table of Contents

Pour générer une chaîne aléatoire (par exemple, mots de passe, noms d'utilisateur, etc.), il est possible d'utiliser les outils dtandards fournis par les systèmes Linux/Unix.

Pour des raisons de sécurité et de confidentialité, il est préférable que les chaînes soient générées hors ligne, plutôt qu'en ligne sur un site Web.)

Utilisation de APG

La commande **APG** inclus par défaut sur certaines distributions Linux, permet de générer aléatoirement des mots de passe.

Par exemple pour générer des mots de passe de taille 5 à 10 dans les sous-ensembles **S**pecial (caractères spéciaux), **N**umeric, **C**apital (majuscules) et **L**ower (minuscules), la commande:

```
apg -MSNCL -m 5 -x 10
```

retourne

```
@OpVeyhym9
3:Glul
3DroomIc?
hed&Os0
NefIj%Ob3
35Quok}
```

utilisation de /dev/urandom

La philosophie Unix de fournir «de nombreux petits outils qui font bien une chose» sert très bien dans ce cas.

- **/dev/urandom** est un flux “d'octets” aléatoires (qui incluent des caractères non imprimables)
- **base64** encode les données d'octets dans [A-Za-z0-9 / +] (qui est entièrement imprimable)

La commande suivante, donne 57 caractères possibles:

- les caractères difficiles à distinguer (I1100) ont été supprimés
- elle est simple à taper et mémorable
- elle utilise des outils système standard - pas de binaires supplémentaires
- elle ne “gaspille” pas beaucoup de caractère aléatoire (utilise 89% des bits aléatoires qu'il reçoit contre ~ 24% pour les solutions directement acheminées vers tr)
- la sortie peut être facilement sélectionnée et collée ou imprimée et retapée avec un taux d'erreur humain minimal
- elle est plus courtes que les solutions codées en hexadécimal (32 et 64 au lieu de 22 et 44) telles que md5sum / sha1sum, etc.
- elle permet de gérer l'entropie:
  - 44 caractères donnent: log2 (5744)> 256,64 bits d'entropie
  - 22 caractères donnent: log2 (5722)> 128,32 bits d'entropie

```
base64 </dev/urandom | tr -d 'O0Il1 + /' | head -c 44; printf '\ n'
```

La commande xxd (convertit une entrée binaire en représentation hexadécimale et inversement) permet de spécifier la longueur de la chaîne désirée avec l'option -l

```
xxd -l16 -ps /dev/urandom
```

Inclusion et exclusion de caractères

On peut utiliser /dev/urandom avec tr pour supprimer les caractères indésirables. Par exemple, pour obtenir uniquement des chiffres et des lettres:

```
tr -dc A-Za-z0-9 </dev/urandom | head -c 13 ; echo ''
```

Sinon, pour inclure plus de caractères de la liste des caractères spéciaux du mot de passe OWASP:

```
tr -dc 'A-Za-z0-9!"#$%&'\''()*+,-./:;<=>?@[\]^_`{|}~' </dev/urandom | head -
c 13  ; echo
```

Pour résoudre les problèmes rencontrés avec le language par défaut, ajouter LC_ALL=C comme ceci:

```
LC_ALL=C tr -dc 'A-Za-z0-9!"#$%&'\''()*+,-./:;<=>?@[\]^_`{|}~' </dev/urandom
| head -c 13 ; echo
```

Production de chaînes longues

Pour obtenir plus 100 octets avec **/dev/urandom** on peut utiliser **dd** pour copier l'entrée en sortie en appliquant les modificateurs donnés en arguments (qui peuvent inclure la taille et le nombre de blocs)

```
base64 -w0 </dev/urandom | dd bs=1k count=1
```

Lorsqu'on a besoin d'un sous-ensemble de caractères, on peut insérer un modificateur dans le tube.
Ex: tr -d '[A-Z / +]' pour se débarrasser des majuscules et + et
- On peut définir le bs (taille de bloc) sur la longueur dont vous avez besoin.
Sous Linux, base64 encapsule à 76 colonnes par défaut et doit être réinitialisé avec -w0 sauf si on le souhaite.

```
tr -dc A-Za-z0-9 </dev/urandom | dd bs=100 count=1 2> /dev/null
```

Le 2> /dev/null à la fin de la commande dd sert à supprimer la sortie “… enregistrements dans / … enregistrements sortants”.

En cas de problème de l'entrée due à la langue par défaut ajouter LC_ALL=C à la commande:

```
LC\_ALL=C tr -dc A-Za-z0-9 </dev/urandom | dd bs=100 count=1 2> /dev/null
```

Utilisation des expressions régulières

Pour génèrer une chaîne aléatoire de 10 caractères on utilise une liste de caractères codée en dur:

```
cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 10 | head -n 1
```

Cette solution n'est pas une pratique de programmation encouragée en raison de la lisibilité, de la compacité et de la propreté du code. De plus, certains des caractères ASCII imprimables spéciaux pourraient être interprétés par bash, sans parler de l'inconvénient le plus évident de sa seule doublure.

Pour générer un mot de passe avec la plus grande entropie possible on peut utiliser les classes de caractères prédéfines dans les expressions régulières

Certaines classes de caractères nommées sont prédéfinies dans des expressions entre crochets, comme suit. Leur interprétation dépend de la locale **LC_CTYPE**; par exemple, `[:alnum:]` désigne la

classe de caractères des nombres et des lettres dans la langue actuelle.

Classe	Signification
[:alnum:]	Caractères alphanumériques: «[:alpha:]» et «[:digit:]»; dans les paramètres régionaux «C» et le codage de caractères ASCII, c'est la même chose que «[0-9A-Za-z]».
[:alpha:]	Caractères alphabétiques: «[:lower:]» et «[:upper:]»; dans les paramètres régionaux «C» et le codage de caractères ASCII, c'est la même chose que «[A-Za-z]».
[:blank:]	Caractères vides: espace et tabulation.
[:Cntrl:]	Caractères de contrôle. En ASCII, ces caractères ont les codes octaux 000 à 037 et 177 (DEL). Dans d'autres jeux de caractères, ce sont les caractères équivalents, le cas échéant.
[:digit:]	Chiffres: 0 1 2 3 4 5 6 7 8 9.
[:graph:]	Caractères graphiques: «[:alnum:]» et «[:punct:]».
[:lower:]	Minuscules; dans les paramètres régionaux «C» et le codage de caractères ASCII, il s'agit de `a b c d e f g h i j k l m n o p q r s t u v w x y z`.
[:print:]	Caractères imprimables: «[:alnum:]», «[:punct:]» et espace.
[:punct:]	Caractères de ponctuation; dans la locale «C» et le codage de caractères ASCII, il s'agit de `! "# $% & '() * +, -. /:; <=>? @ [\] ^ _ {|} ~.`
[:space:]	Caractères d'espacement: dans les paramètres régionaux «C», il s'agit de tabulation, nouvelle ligne, tabulation verticale, saut de page, retour chariot et espace.
[:upper:]	Lettres majuscules: dans les paramètres régionaux «C» et le codage de caractères ASCII, il s'agit de `A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`.
[:xdigit:]	Chiffres hexadécimaux: `0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f`.

La commande suivante génère une chaîne en utilisant tous les caractères imprimables ASCII - de 32 (espace) à 126 (tilde, ~). La longueur du mot de passe peut être contrôlée avec l'indicateur -c de head.

```
< /dev/urandom tr -cd "[:print:]" | head -c 32; echo
```

Pour ne pas inclure l'espace, juste les caractères 33-126, utiliser `[:graph:]`

```
< /dev/urandom tr -cd '[:graph:]'| tr  -d '\\' | head -c 32
```

La commande suivante génère une chaîne en utilisant tous les caractères alphanumériques:

```
cat /dev/urandom | base64 | head -n 1 |tr -dc '[:alnum:]' |cut -c -13
b0R55751vWW9V
```

Pour configurer la longueur du mot de passe, remplacer le nombre dans la commande **cut** par la longueur dont vous avez besoin, par exemple 24 caractères

```
cat /dev/urandom | base64 | head -n 1 | tr -dc '[:alnum:]' | cut -c -24
WFBhVHuKbrnRhd5kdWXPzmZG
```

Si on ne veut pas confondre 0 ou O, 1 ou l, on peut filtrer avec un autre **tr**

```

```

```
cat /dev/urandom | base64 | head -n 1| tr -dc '[:alnum:]' | tr -d '0O1l' |
cut -c -24
JuCphwvhrhppD9wVqfpP2beG
```

Pour un mot de passe très sécurisé (où 16 est la longueur pw):

```
cat /dev/urandom | tr -cd [:graph:]|head -c 16
```

Exemple de résultat:

```
jT@s_Nx]gH<wL~W}
```

Ou bien, pour générer plusieurs mots de passe:

```
cat /dev/urandom | tr -cd [:graph:]|fold -w 16|head -6
```

Exemple de résultat:

```
7ck%G7'|f&}_8(]s
<?*]E.CG[NB'gK#A
:tF-WPTOa3!i7S(h
2xy(>=%3=Kb1B$`6
*98Lf{d?Jzb}6q1\
E7uCEAN2Hz]]y|5*
```

Un peu moins sécurisé (jeu de caractères plus petit):

```
cat /dev/urandom | base64 -w 0 | fold -w 16 | head -6
```

Exemple de résultat:

```
rJqYxYZLOTV+A45w
PUKQ+BoBf6yfZw5m
+LRfpsN9CsLRiN8V
yMS6zDpL6NHmG17s
yTUWPG7Rum97schi
cognvjVwaKaAyPRK
```

Utilisation de md5

On peut utiliser l'un des outils md5 qui a précisément cet objectif. Dans le cas de la création d'un mot de passe complètement aléatoire, on peut utiliser le **md5pass**. C'est un outil très simple à utiliser et très utile, car on peut utiliser le “texte normal” avec un “sel” pour créer un bit de saut du même mot de passe que l'on peut récupérer par la suite, ou on peut également obtenir un complètement mot de

passe aléatoire tout le temps. L'utilisation générale est:

```
md5pass [mot-de-passe] [sel]
```

où **mot-de-passe** est un mot choisi qui sera utilisé pour la construction de la chaîne aléatoire et **sel** est le salt d'octets à utiliser. Comme ça:

```
md5pass mot-de-passe

$1$.MUittVW$j.XDTF1QRnxqFdXRUiSLs0
```

Cela créera un mot de passe “une séquence aléatoire” que l'on utilisera. Si on ne définit pas de sel, on ne pourra peut-être pas recréer cette même chaîne par la suite.

Cependant, si on spécifie un sel comme celui-ci:

```
md5pass mot-de-passe 512

$1$512$.0jcLPQ83jgszaPT8xzds0
```

alors on peut créer une séquence que l'on peut récupérer si on utilise le **mot-de-passe** en conjonction avec le même sel (ou salt) s'il a été défini à l'origine.

Utilisation de la variable $RANDOM

Selon le niveau de caractère aléatoire que l'on souhaite, on peut simplement utiliser la variable intégrée $RANDOM de bash (également zsh et ksh, éventuellement d'autres):

```
echo $RANDOM | tr '[0-9]' '[a-z]'
bfeci
echo $RANDOM | tr '[0-9]' '[a-z]'
cijjj
```

Les méthodes lisant directement à partir de /dev/urandom sont beaucoup plus simples, mais par souci de complétion, on peut également utiliser $RANDOM:

```
echo $(for((i=1;i<=13;i++)); do printf '%s' "${RANDOM:0:1}"; done) | tr
'[0-9]' '[a-z]'
```

Important: cette solution ne produira que des chaînes aléatoires en utilisant les 10 premieres lettres de l'alphabet.

Utilisation de shuf

La commande shuf génère des permutations aléatoires entre les lignes d'entrée et la sortie standard. Si on lui donne un fichier ou une série de fichiers, il mélangera les lignes et écrira le résultat sur la sortie standard.

shuf fait partie de linux coreutils et est largement disponibleInspiré par Pablo Repetto, je me suis retrouvé avec cette solution facile à retenir:

```
shuf -zer -n20 {A..Z} {a..z} {0..9}
```

- **-z** évite la sortie multi-lignes
- **-e** fait écho au résultat
- **-r** permet à n'importe quel caractère d'apparaître plusieurs fois
- **-n20** chaîne aléatoire d'une longueur de 20 caractères
- **{A..Z} {a..z} {0..9}** classes de caractères autorisées

Le problème ici est “-z” car il “n'évite pas la sortie multi-lignes” mais ajoute un \0 au lieu d'une nouvelle ligne à chaque chaîne (visible en ajoutant `| od -xc`). on peut supprimer les caractères zéro avec tr: `shuf -zer -n20 {A..Z} {a..z} {0..9} | tr -d '\0'`

Les deux commandes suivantes génèrent respectivement des mots de passe et des phrases de passe aléatoires:

```
shuf --random-source=/dev/urandom --repeat --head-count=20
file_with_characters | tr --delete '\n'

shuf --random-source=/dev/urandom --repeat --head-count=7 file_with_words |
tr '\n' ' '
```

- Le générateur de mot de passe nécessite un fichier `file_with_characters` contenant tous les caractères que l'on souhaite utiliser pour le mot de passe, un caractère par ligne et exactement une fois chacun. Le fichier ne doit pas contenir de lignes vides et les lignes doivent être terminées par une nouvelle ligne.
- Le générateur de phrase de passe nécessite un fichier `file_with_words` contenant tous les mots que la phrase de passe utilise, un mot par ligne et exactement une fois chacun. Le fichier ne doit pas contenir de lignes vides et les lignes doivent être terminées par une nouvelle ligne.
- L'option **–head-count** spécifie la longueur du mot de passe - en caractères - ou de la phrase de passe - en mots.

Utilisation de la commande openssl

```
openssl rand -base64 12
```

ou

```
openssl rand -hex 12
```

rand -hex limitera la sortie à seulement 16 caractères, plutôt que les 90+ du clavier. **base64** est meilleur car il contient 64 caractères, mais ce n'est pas aléatoire (par exemple, il existe des motifs de remplissage prévisibles, et peut-être que certains caractères apparaissent plus souvent que d'autres

Utilisation de pwgen

pwgen génère des mots de passe aléatoires, sans signification mais prononcables. Ces mots de passe contiennent soit uniquement des lettres minuscules, soit des majuscules et des minuscules mélangées, soit des chiffres ajoutés. Les lettres majuscules et les chiffres sont placés de manière à faciliter la mémorisation de leur position lors de la mémorisation du mot uniquement.

Par exemple pour générer 7 mots de passe de longueur 13:

```
pwgen 13 7
Eu7Teadiphaec giepahl3Oyaiy iecoo9Aetaib4 phaiChae6Eivi athoo3igee8Co
Iphu4ufeDeelo aesoYi2lie9he
```

Pour améliorer l'entropie on peut utiliser l'argument -s (c'est-à-dire générer des mots de passe plus sûrs, complètement aléatoires mais difficiles à retenir):

```
pwgen -s 13 7
eAfycrPlM4cYv 4MRXmZmyIVNBp D8y71iqjG7Zq7 FQRHcserl4R8O yRCUtPtV3dsqV
0vJpp2h0OrgF1 QTp7MKtJyTrjz
```

Utilisation de gpw

Ce package génère des mots de passe prononçables. Il utilise les statistiques des combinaisons de trois lettres (trigraphes) tirées de tous les dictionnaires que l'on alimente.

Par exemple pour générer 7 mots de passe (noms d'utilisateur) de longueur 13:

```
gpw 7 13
sreepoidahsas
risadiestinge
ntodynesssine
décourager
natinglumperm
riasigentspir
enderiferback
```