UnrealIRCD

Installation et configuration d'un serveur IRC

par Olivier Locatelli

Date de publication : 25/04/2011

Dernière mise à jour : 28/05/2011

Les premiers pas pour découvrir les serveurs IRC en tant qu'administrateur

I - Préambule

I-A - Introduction

Nombreux sont les utilisateurs d'IRC qui désirent franchir le pas et installer leur propre serveur, que ce soit pour en apprendre un peu plus sur la technique de gestion ou pour créer leur propre réseau.

Cet article se veut être un guide d'installation pratique et un pense-bête pour y arriver.

I-B - Avertissement et pré-requis

UnrealIRCd peut fonctionner aussi bien sous Windows que sous Linux, nous ne traiterons ici pour ce qui est de l'installation que la version Linux.

Vous devez savoir (et pouvoir) vous connecter à votre serveur par SSH.

⚠ *Toutes les opérations qui suivent doivent être effectuées avec un utilisateur « normal » du serveur, en aucun cas avec l'utilisateur **root** ou un utilisateur ayant des droits particuliers.*

II - Installation

II-A - Obtenir les sources

Pour télécharger UnrealIRCd, il faut vous rendre sur le site officiel et le téléchargement de la dernière version stable (actuellement 3.2.8.1) se lancera seul. Ensuite, uploadez le fichier sur votre serveur puis connectez vous par ssh à votre serveur.

Si vous désirez télécharger directement depuis votre machine, notez le numéro de la dernière version stable et téléchargez la directement en faisant:

```
$ wget http://www.unrealircd.com/downloads/Unreal3.2.8.1.tar.gz
```

II-B - Compiler les sources

Il vous suffit de décompresser les sources et lancer la procédure d'installation:

```
$ tar xvfz Unreal3.2.8.1.tar.gz
$ cd Unreal3.2
$ ./Config
$ make
```

Il vous suffit de répondre aux questions (dans le doute, laissez les valeurs par défaut) et d'aller jusqu'au bout.

II-C - Ajout de modules

Il est possible d'ajouter des modules sur UnrealIRCd.

Pour cela, téléchargez les fichiers sources des modules dans le répertoire Unreal3.2/src/modules/ et compilez les en faisant :

```
$ make custommodule MODULEFILE=module
```

Les modules validés par l'équipe de développement d'UnrealIRCD sont disponibles à http://www.unrealircd.com/modules.php

III - Configuration

La configuration principale d'Unreal 3.2 se fait par un fichier principal obligatoirement appelé unrealircd.conf qui utilise un format basé sur les blocs. Ce fichier peut inclure d'autres fichiers de configuration, pour une meilleure lisibilité et organisation.

Les blocs remplacent ce qui était auparavant les « lines », pour une gestion plus souple et plus lisible.

III-A - Normes d'écriture

III-A-1 - Structure

Les blocs sont structurés de la manière suivante:

```
<bloc> <valeur> {
    <directive1> <valeur>;
    <directive2> <valeur> {
        <sous-directive1> <valeur>;
        <sous-directive2> <valeur>;
    };
};
```

Le fichier de configuration est sensible à la case (aux majuscules / minuscules) donc BLOC ou Bloc n'est pas la même chose que bloc.

Il est possible d'accéder à une directive sans déclarer son bloc au préalable, en utilisant le séparateur « :: ». Par exemple, dans le cas précédent, on peut utiliser *bloc::directive1*. Et l'on pourrait accéder à une sous-directive de la même manière: *bloc::directive2::sous-directive*1

III-A-2 - Commentaires

Trois types de commentaires sont supportés :

- # commentaire sur une seule ligne
- // commentaire sur une seule ligne
- /* Commentaire
  multi-ligne */

III-A-3 - Structure des fichiers

Par habitude et par souci d'entretien, je sépare ma configuration en plusieurs fichiers qui ne suivent pas l'exemple de configuration d'UnrealIRCD. Ceci n'est bien sûr pas imposé, vous pouvez tout à fait utiliser une autre structure, voire mettre toute votre configuration dans un seul fichier.

Les fichiers de configuration complémentaires sont chargés grâce à l'instruction « include ». Si le découpage est bien effectué, l'inclusion peut être faite à n'importe quel niveau du fichier de configuration. Il y a tout de même des

placements privilégiés qui permettent de suivre une certaine logique, comme inclure le fichier *help.conf* dès le début, juste après avoir chargé les modules.

III-B - Fichier principal : unrealircd.conf

III-B-1 - Chargement des modules

Les modules sont la première chose à charger dans la configuration, car ils peuvent influer sur les blocs utilisés.

Le chargement se fait avec l'instruction « loadmodule ». Certains modules sont obligatoires pour le bon fonctionnement du serveur et sont par défaut dans le fichier exemple.

Les modules disponibles sont habituellement dans le répertoire src/modules/ et sont soit des fichiers .so (sous linux) soit des .dll (sous windows).

```
loadmodule "src/modules/commands.so";
loadmodule "src/modules/cloak.so";
loadmodule "src/modules/m_nocodes.so";
include "help.conf"
```

Vous constatez, comme précisé plus haut, que le fichier *help.conf* est appelé juste après le chargement des modules.

III-B-2 - Définition du serveur ? M:Line

Ce bloc **(obligatoire)** donne le nom, la description et l'identifiant numérique du serveur.

- me::name est le nom qui sera vu par les utilisateurs lorsqu'ils se connecteront sur le serveur,
- me::info est la description du serveur, à titre informatif,
- me::numeric est l'identifiant numérique du serveur sur le réseau. Il doit être unique sur le réseau et être compris entre 0 et 254

Exemple

```
me {
   name "irc1.developpez.net";
   info "Serveur principal de l'IRC developpez.net";
   numeric 1;
};
```

III-B-3 - Définition de l'administrateur ? A:Line

Ce bloc **(obligatoire)** permet de donner les informations sur l'administrateur du serveur. Il peut contenir autant de lignes que vous le voulez, mais on y retrouve habituellement le nom, le pseudo et le moyen de contacter l'administrateur. Ces informations sont disponibles pour les utilisateurs lorsqu'ils utilisent la commande */admin* sur IRC.

Exemple

```
admin {
   "Olivier Locatelli";
   "olocatelli@monadresse.net";
};
```

III-B-4 - Définition des classes ? Y:Line

Ce ou ces blocs **(obligatoires)** vont vous permettre d'associer une connexion à un type particulier d'usage (classes d'utilisateurs) afin de les réutiliser dans les différents blocs de restriction et/ou d'autorisation.

On retrouve habituellement au moins 2 blocs, celui des clients (utilisateurs « normaux ») et celui des serveurs. Certains ajoutent parfois un bloc pour les administrateurs afin de les différencier des utilisateurs et pouvoir facilement leur donner plus de privilèges.

Les noms des blocs sont très importants, car ils servent de référence dans d'autres blocs.

Les blocs contiennent 4 directives:

- pingfreq est la fréquence en secondes à laquelle votre serveur va envoyer des requêtes PING sur la connexion, une bonne valeur se situe entre 90 et 180,
- maxclients est le nombre maximal de connexions simultanées de ce type acceptées par le serveur,
- sendq est la taille du tampon d'envoi du serveur, il vaut mieux une valeur élevée pour la classe des connexions serveur, et une plus faible pour celle des clients. Pour ma part, je choisis un ratio de 10,
- recvq est la taille du tampon de réception, elle permet de contrôler le flood. Cette directive n'est à utiliser **que pour les classes « client »** et une bonne valeur se situe entre 3000 et 8000 (valeur par défaut),
- connfreq est utilisé **uniquement pour les classes « serveur »** et sert à définir la fréquence de tentative de connexion à un serveur du réseau lorsque celui-ci est déclaré en *autoconnect*.

Exemple

```
class clients {
   pingfreq 90;
   maxclients 200;
   sendq 10000;
   recvq 8000;
};
class servers {
   pingfreq 90;
   maxclients 10;
   sendq 100000;
   connfreq 60;
};
```

III-B-5 - Définition des autorisations ? I:Lines

Ces blocs **(obligatoires)** permettent de définir les autorisations et restrictions générales par type de connexion (voir chapitre précédent) ainsi que selon certains critères de connexion comme l'IP.

On trouve habituellement un premier bloc pour tous les clients, donnant des restrictions générales (le nombre de connexions simultanées autorisées), puis des blocs créant des exceptions.

Les blocs peuvent contenir les paramètres suivants:

- ip : le masque IP de la connexion (jokers autorisé) sous la forme utilisateur@IP,
- host : le masque de l'hôte de la connexion (jokers autorisés) sous la forme **utilisateur@hote**,
- password : un mot de passe de connexion **(optionnel)**,
- class : la classe de connexion à qui s'applique la règle,
- maxperip : le nombre de connexions simultanées autorisées depuis l'adresse IP **(optionnel mais recommandé)**,
- ipv6-clone-mask : la sensibilité aux clones lors de l'utilisation d'une adresse IPv6. La valeur par défaut est 64, ce qui signifie que deux clients dont l'adresse IPv6 ne diffèrent que par leurs derniers bits sont considérés

comme des clones. Si cette valeur est passée à 128 (maximum), chaque adresse IPv6 est considérée unique. Ce réglage est **optionnel**,
- redirect-server : si le serveur a atteint sa capacité maximale pour la classe (clients::maxclients par exemple), cette instruction force les clients compatibles à se connecter sur le serveur de redirection (**optionnel**),
- redirect-port : si un serveur de redirection est défini, il est possible de définir le port de ce serveur ici. Par défaut, il s'agit du port 6667 (**optionnel**),
- : il est possible de mettre 4 options à un bloc d'autorisation:
  - useip : utilise l'adresse de la connexion plutôt que l'hôte littéral,
  - noident : ne fait pas de requête « ident » mais utilise le « username » fourni par le client,
  - ssl : seules les connexions SSL sont autorisées,
  - nopasscont : continue les vérifications si le mot de passe n'est pas concordant. Ceci est utile lorsque vous avez plusieurs classes utilisant des connexions avec mot de passe, la connexion n'est pas interdite tant que toutes les vérifications n'ont pas échoué.

Exemple

```
// Premier bloc : 3 connexions autorisées pour chaque IP par défaut
allow {
   ip *;
   host *;
   class clients;
   maxperip 3;
};
// Deuxième bloc : 10 connexions autorisées pour la plage 192.168.0.*
// indépendamment de leur hôte, uniquement en SSL
allow {
   ip 192.168.0.*;
   hostname NONAME;
   class clients;
   maxperip 10;
   options {
      useip;
      ssl;
};
```

Notez que les directives ip et host fonctionnent avec un « OU ». Si vous voulez restreindre à une IP (ou une plage d'IPs), il vous faut mettre une valeur host qui ne sera jamais valide, et inversement pour forcer sur un host.

III-B-6 - Ports d'écoute ? P:Line

Comme tout serveur, UnrealIRCd a besoin de ports sur lesquels les clients se connecteront.

Les définitions peuvent se font par IP et port par port ou par plage de ports, et peuvent contenir des options:

- clientsonly : seules les connexions définies de type « client » peuvent utiliser cette IP et ce port,
- serversonly : seules les connexions définies de type « server » peuvent utiliser cette IP et ce port,
- ssl : ce port est réservé aux connexions SSL,
- java : ce port est compatible avec les applets Java.

Exemple

```
listen 127.0.0.1:7000 {
   options {
      serversonly;
   };
};
listen 192.168.0.*:6660 {
   options {
      clientsonly;
      ssl;
   };
```

Exemple
```
};
listen *:6665-6670;
```

Dans l'exemple précédent, nous venons donc de définir 3 classes d'écoute, permettant les connexions suivantes:

- depuis le serveur lui-même, d'autres serveurs (services, serveurs de statistiques, ?) peuvent se connecter sur le port 7000, et seulement des serveurs,
- depuis le réseau local, des clients utilisant une connexion SSL peuvent se connecter sur le port 6660,
- n'importe qui, client ou serveur, peut se connecter sur les ports 6665 à 6670.