

Utiliser le protocole IRC avec PHP

Par linkboss
et Eser Deniz (SRWieZ)



www.openclassrooms.com

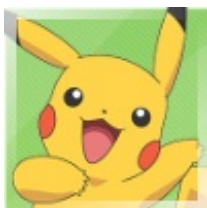
*Licence Creative Commons 2 2.0
Dernière mise à jour le 25/07/2010*

Sommaire

Sommaire	2
Utiliser le protocole IRC avec PHP	3
IRC, kézako ?	3
Construction théorique du programme	4
I. Cahier des charges	4
II. Solutions pour le cahier des charges	4
Pratique — Codage	6
Préparation	6
Le codage	6
Petit récapitulatif	19
Q.C.M.	19
Partager	20



Utiliser le protocole IRC avec PHP



Par

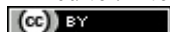
linkboss et



Eser Deniz (SRWieZ)

Mise à jour : 25/07/2010

Difficulté : Intermédiaire



PHP, en plus de permettre de rendre son site Web interactif, permet de faire des programmes (voir le tutoriel sur [PHP-CLI](#) de gnomnain). Mais il permet aussi de dialoguer avec d'autres serveurs ! Vous pouvez donc faire avec PHP beaucoup de choses, comme un indexeur de pages (*via* cURL), un lecteur de news RSS... mais vous pouvez aussi dialoguer avec un serveur IRC facilement ! C'est ce que l'on va voir dans ce tutoriel.

Sommaire du tutoriel :



- [IRC, kékako ?](#)
- [Construction théorique du programme](#)
- [Pratique — Codage](#)
- [Petit récapitulatif](#)
- [Q.C.M.](#)

IRC, kékako ?

IRC, pour *Internet Relay Chat*, est un des premiers (peut-être même **le** premier) protocole de communication instantanée informatique au monde. Il a été créé en 1988 ; c'est un protocole public contrairement à d'autres (MSN, Yahoo! Messenger et d'autres), qui est aussi *relativement* simple si on le compare aux autres protocoles existants. Ici, pas de cryptage (enfin si, mais c'est optionnel et on ne le verra pas ici), pas de trucs compliqués, IRC est un protocole simple.



Cliquez sur les images pour accéder aux sites des clients

De nombreux clients existent pour IRC et dans à peu près n'importe quel langage de programmation supportant l'accès au réseau. Le plus connu est mIRC, mais il en existe plein d'autres, comme notamment XChat, tinyIRC (un client qui tient uniquement dans 150 Ko !), Konversation sous Linux, ChatZilla, qui est un client dans un simple plugin pour Firefox, ou même Irssi, un client en console très performant (voir [le tutoriel](#) à ce sujet). On trouve même des clients qui combinent IRC et d'autres protocoles (MSN, ICQ, Jabber), comme par exemple Miranda IM sous Windows, ou Pidgin sous Linux.

Mais le client que vous devriez voir est [Mibbit](#), un client écrit en PHP et en AJAX (avec un truc en Java derrière quand même).

Pour plus d'informations sur le protocole IRC, je vous conseille d'aller voir le document le décrivant et qui nous servira durant tout le tutoriel (gardez-le ouvert) : la [RFC 1459](#) (ici en français).



Et PHP dans tout ça ?

PHP permet tout simplement de dialoguer avec un serveur IRC, comme n'importe quel client ! Vous pouvez ainsi créer différents programmes en PHP pour interagir avec IRC, comme principalement les *bots*. Mais faites attention, les *bots* IRC (PHP, Python et dans d'autres langages) sont légion sur Internet, donc vous vous ferez peut-être mal voir étant donné que vous (re)développez un truc déjà développé X fois...

On en arrive à notre fil rouge, que je vais utiliser pour vous apprendre à utiliser IRC avec PHP durant ce tutoriel : la création d'un *bot*. Oui, je sais que j'ai dit que c'était mal, mais le problème c'est que développer un autre truc comme par exemple un client, c'est beaucoup plus dur, donc je vous montre comment faire un *bot*. Maintenant, après ce tutoriel, libre à vous de faire un *bot*, ou de faire autre chose.

Construction théorique du programme

Bon, maintenant, qu'est-ce que l'on va faire comme *bot* ? Eh bien, je vous propose un *bot* relativement simple à faire : un ***bot de quiz***. Réaliser ce genre de *bot* est relativement facile, car il ne fait que poser des questions et vérifier les réponses...

I. Cahier des charges

Voici ce que fera notre *bot* :

- poser des questions ;
- comprendre les réponses, et dire qui a bien répondu (en premier) ;
- empêcher le lancement de deux questions en même temps ;
- *auto-rejoin on kick* (si on le vire du canal, on le fait revenir immédiatement).

Voilà, c'est tout. Ce n'est pas beaucoup, mais je juge que c'est déjà bien pour un premier *bot*.

II. Solutions pour le cahier des charges

1. Structuration de la « base de données » de questions

Pour stocker les questions du *bot*, nous allons utiliser un unique fichier pour toutes les questions. Nous n'utiliserons pas de BDD, car c'est relativement lent et gourmand en mémoire ; et pour juste une liste de questions, nous n'allons pas nous embêter à faire du SQL. On structurera le fichier de sorte à pouvoir extraire les questions dans un *array*.

Je vous propose dans ce tutoriel un type de structuration pour vos questions, mais vous pouvez en choisir un autre.

Tout d'abord, dans notre fichier, les questions seront séparées par ligne. Donc à une ligne correspondra un composé question-réponse.

Ensuite, la question sera séparée de la réponse par une chaîne de caractères définie au préalable. Pour ce *bot*, je vais choisir cette chaîne de caractères :

Code : Autre

```
<|#()#|>
```

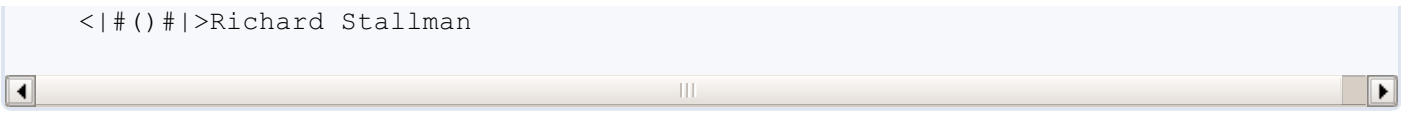


Bien sûr, cette chaîne de caractères ne devra se trouver ni dans la question, ni dans la réponse.

Voici donc un exemple de fichier de questions que vous pouvez utiliser avec ce *bot* :

Code : Autre

```
Quel est le nom du premier langage de programmation encore utilisé actuellement ?  
<|#()#|>FORTRAN  
Que veut dire PHP ?<|#()#|>PHP Hypertext Preprocessor  
Quelle est la propriété CSS pour définir la bordure d'un bloc ?<|#()#|>border  
Quand a été créé le Site du Zéro ?<|#()#|>1999  
Quelle est la Réponse à la Grande Question de la Vie, de l'Univers et du Reste ?  
<|#()#|>42  
Qui a créé Linux ?<|#()#|>Linus Torvalds  
Qui est à l'initiative du projet de système d'exploitation libre GNU ?
```



Pour mon fichier de questions, j'aurais pu utiliser une structure XML, mais c'est assez dur de naviguer dans les nœuds XML et surtout trop compliqué pour le petit *bot* que l'on va faire. C'est déjà assez compliqué avec le protocole IRC, on ne va pas rajouter en plus le XML (mais si vous voulez le faire, libre à vous).

2. Structuration globale du programme

Maintenant, on passe à la construction (théorique) globale du programme, on mettra les différentes étapes de fonctionnement du *bot* (regroupées en catégories).

Lecture des questions sur le fichier

Ici, l'étape est plus que simple et sera effectuée **avant** la connexion au serveur IRC :

1. on ouvre le fichier des questions et on le lit ;
2. on extrait chaque couple question/réponse ;
3. on sépare la question et la réponse.

Comme vous le voyez, rien de plus simple. Si vous n'y arrivez pas, retournez consulter le [tutoriel sur la manipulation des fichiers](#), de M@teo21.

Connexion au serveur IRC

Alors pour IRC, la connexion se passe en plusieurs étapes :

1. on ouvre la connexion au serveur (au hasard, irc.epiknet.org sur le port 6667) ;
2. on indique le pseudonyme (*nick*) du *bot*, avec la commande NICK ;
3. on indique le « vrai nom » du bot (*username*) via la commande USER ;
4. le serveur nous envoie une requête de PING ;
5. on rejoint le canal où le *bot* doit être (au hasard, #sdz-quiz).

Cette démarche n'est pas propre à PHP, mais au protocole IRC. Si vous essayez de vous connecter directement au serveur IRC (avec Telnet par exemple), vous verrez que vous devez exécuter ces mêmes étapes.

D'ailleurs, en réalité il y a une étape « 4,5 » qui correspond à la réception du MOTD, mais ici le MOTD ne nous intéresse pas, donc on ne le met pas dans la description théorique.

Lorsque le *bot* est au repos (pas de question posée)

1. On voit s'il y a un nouveau message.
 - a. Si oui, on regarde si c'est une demande de question.
 1. S'il y en a une, on passe en « mode question ».
 2. Sinon, on ne fait rien.
 - b. Si non, on passe.
2. On regarde si le serveur nous a envoyé un PING.
 - a. Si oui, on renvoie un PONG.
 - b. Si non, on ne fait rien.
3. On regarde si on a été *kické* (éjecté) du canal.
 - a. Si oui, on retourne dans le salon.
 - b. Si non, on ne fait rien.

Ça a l'air long, mais ne vous inquiétez pas, ça se résume par peu de lignes de code.

Quand on a demandé une question (« mode question »)

1. On regarde s'il y a un message.
 - a. Si oui, on vérifie si c'est la réponse à la question.
 1. Si oui, on affiche « Bravo ! » ou un message du genre.
 2. Si non, on ne fait rien.
 - b. Si non, alors on ne fait rien.

Pour le reste du mode de fonctionnement, on le fait à l'image du mode au repos (à partir du 2.).

Voilà pour le mode de fonctionnement, maintenant on passe à la création, où tout le mode pratique de la connexion à IRC avec PHP va vous être expliqué !

Pratique — Codage

Maintenant, on va (enfin) coder le *bot* en lui-même. Au départ, on va coder chaque partie séparément, puis nous rassemblerons le tout pour former le *bot* final.

Préparation

Tout d'abord, nous allons nous préparer pour créer le *bot*. Je vous conseille de créer un dossier rien que pour le *bot*, même si à la fin, il n'y aura que deux fichiers. Pourquoi ? Tout simplement pour éviter de se perdre dans une masse de fichiers.

Maintenant, créez un fichier appelé *bot.php* (le *bot*), et un fichier *questions.txt* (les questions), dans lequel vous mettrez vos questions (si vous n'avez pas d'idée pour tester, prenez le fichier d'exemple que j'ai mis plus haut).

Voilà, on est prêts à commencer !

Le codage

Alors, comme je l'ai dit plus haut, on va le faire par étapes, puis à la fin rassembler le tout. Je vous conseille d'avoir gardé ouverte la RFC 1459 (le lien est en haut du tuto), car je vais y faire pas mal de références.



Note : ici, je ne vais pas faire de connexion avec un mot de passe. Ce sera à vous de le faire plus tard, si vous en avez besoin (ce n'est pas très difficile de toute façon).

Citation : Informations utiles pour le bot

Serveur : irc.epiknet.org port 6667
Nom du bot : QuizBot
Canal : #quiz-sdz
Nom d'utilisateur (*username*) : Quiz

Le nom d'utilisateur ne nous sera pas très utile.

1. Construction de la liste des questions-réponses

Ici, rien de plus simple, on récupère la totalité du fichier dans une variable, via `file_get_contents()`, puis on sépare questions et réponses, et éventuellement on retire le dernier élément si celui-ci est vide (c'est relatif aux fichiers, certains éditeurs insèrent une ligne en sauvegardant) :

Code : PHP

```
<?php
$file = file_get_contents('questions.txt'); // On lit le fichier des
questions dans une chaîne
$questions = explode("\n", $file); // On sépare chaque ligne (\n est
le caractère signifiant une nouvelle ligne)

// On sépare la question et la réponse
for($i=0;isset($questions[$i]);$i++)
    $questions[$i] = explode('<|#()#|>', $questions[$i]);
// Si le dernier élément est vide (bug relatif aux fichiers), on le
supprime
if(!$questions[count($questions)-1])
    unset($questions[count($questions)-1]);
?>
```

Ici, après le traitement, on pourra trouver les questions dans l'array bi-dimensionnel \$questions : la question n° X sera à \$questions[X][0] et la réponse correspondante \$questions[X][1] .

2. Connexion

Tout d'abord, pour se connecter au serveur IRC, on doit ouvrir la connexion au serveur, et pour cela, rien ne vaut une bonne vieille commande `fsockopen()` . Bien sûr, on vérifie que l'on est bien connectés au serveur, sinon cela ne marchera pas :

Code : PHP

```
<?php
set_time_limit(0);
$socket = fsockopen('irc.epiknet.org', '6667');

// Vérification de la bonne connexion :
if(!$socket)
{
    // Si on n'a pas réussi, on affiche un message d'erreur et on
    quitte.
    echo 'Impossible de se connecter';
    exit;
}
?>
```



Ici, le `set_time_limit(0)` permet d'indiquer à PHP que l'on souhaite que le script tourne indéfiniment (en effet, si le bot tourne 30 secondes et s'arrête, ce n'est pas très utile).

Ensuite, on passe aux étapes 2. et 3. : le renseignement du pseudonyme (*nickname*) et du nom d'utilisateur. Si l'on lit la RFC 1459, le nom d'utilisateur n'est pas pris en compte lorsque l'on veut se connecter, mais il faut quand même le renseigner.

Il reste encore un problème : comment envoie-t-on des données au serveur ? Eh bien, c'est plus que simple : lorsque l'on arrive à se connecter, la commande `fsockopen()` nous renvoie un pointeur de fichier ! Il suffit donc d'effectuer des fonctions `fputs()` pour envoyer des données sur le serveur, et des fonctions `fgets()` pour lire les données du serveur.

Par contre, toujours selon la RFC 1459, il faut envoyer une certaine chaîne de caractères pour indiquer la fin d'un message : c'est la chaîne composée des caractères CR et LF, représentés en PHP par `\r\n`, utilisables uniquement entre guillemets :

Code : PHP

```
<?php
// On renseigne l'USER : ici, je mets un peu n'importe quoi, vu que
le serveur ne prend en compte que le premier argument (mais qu'il a
```

```

besoin de 4 arguments).
fputs($socket, "USER QuizBot QuizBot Quiz Quiz\r\n");
// On donne le NICK.
fputs($socket, "NICK QuizBot\r\n");
?>

```

Maintenant, on arrive aux étapes 4. et 5., où on doit attendre que le serveur nous envoie un PING et y répondre par un PONG, puis attendre la fin du MOTD. Pour ce faire, nous allons utiliser une boucle.

Pour le PING, la RFC nous dit que le serveur nous envoie une ligne du style « PING: 215613202 » à laquelle nous devons répondre « PONG: 215613202 ».

Ensuite, le MOTD ne nous intéressant pas des masses, nous allons simplement l'ignorer.

Code : PHP

```

<?php
$continuer = 1; // On initialise une variable permettant de savoir
si on doit continuer la boucle.
while($continuer) // Boucle principale.
{

    $donnees = fgets($socket, 1024); // Le 1024 permet de limiter la
quantité de caractères à recevoir du serveur.
    $retour = explode(':', $donnees); // On sépare les différentes
données.
    // On regarde si c'est un PING, et, le cas échéant, on envoie
notre PONG :
    if(rtrim($retour[0]) == 'PING')
    {
        fputs($socket, 'PONG :'.$retour[1]);
        $continuer = 0;
    }
    if($donnees)
        echo $donnees;
}
?>

```

Normalement, on est correctement connectés au serveur, il ne reste plus qu'à joindre le canal :

Code : PHP

```

<?php
fputs($socket, "JOIN #quiz-sdz\r\n"); // On rejoint le canal #quiz-
sdz.

```

Voilà, nous sommes maintenant connectés au canal du quiz, on va pouvoir faire le programme principal !

3. Le programme principal

Tout d'abord, on va devoir mettre une variable indiquant si une question a été posée ou non. J'ai choisi le nom `$questionPosee`, mais vous pouvez utiliser autre chose. Quand le mode question sera activé, la variable contiendra le numéro de la question (on pourra donc accéder à la question par `$questions[$questionPosee]`), et quand le mode question sera désactivé, elle vaudra -1 (comme le numéro des questions commence à 0 (*array* oblige), on met -1 sinon on risque des bugs) :

Code : PHP


```
<?php
$questionPosee = -1;
```

Ensuite, nous devons faire tenir notre programme dans une boucle infinie, sinon il s'ouvrira et se refermera instantanément ! (Enfin presque, car il se connectera à IRC entre-temps.)

Code : PHP

```
<?php
// Boucle principale du programme.
while(1)
{
    // Ici, nous mettrons nos commandes.
}
```

Maintenant, il faut lire sur le *socket* les messages envoyés par le serveur (ou pas), avec notre commande `fgets()` , que l'on place dans la boucle. Ensuite, on vérifie si le serveur nous a envoyé quelque chose, donc on met une condition d'existence. Si le serveur nous a envoyé quelque chose, on le traite.

Il faut savoir qu'une ligne de commande IRC ressemble à ceci :

:linkboss!~linkboss@svn44-1-78-228-160-160.fbx.proxad.net PRIVMSG#quiz-sdz :!question. On verra plus tard comment traiter cette ligne, mais pour l'instant il s'agit de la recevoir. Nous utiliserons donc `fgets()` :

Code : PHP

```
<?php
$donnees = fgets($socket,1024);
```

Maintenant, nous avons lu sur le serveur les données qu'il faut. Mais il faut savoir que le serveur peut ne rien envoyer, et contrairement aux *sockets* serveur (en PHP tout du moins, et avec les paramètres par défaut), la lecture n'arrête pas le script ! Donc, avant de procéder à tout traitement, il faut d'abord vérifier que le serveur a bien envoyé quelque chose :

Code : PHP

```
<?php
if($donnees)
{
    echo $donnees;
    // Ici on traite ce que nous a envoyé le serveur.
}
```

Ici le `echo $donnees;` permet juste d'afficher les messages du serveur, histoire de voir ce qu'il nous envoie (pratique pour le *debug* et pour savoir si tout a bien marché).

Maintenant, avant de continuer, on va régler un petit problème inhérent aux programmes en console : la consommation du processeur. En effet, si on laisse le programme comme cela, il consommera la totalité du processeur disponible, ralentissant tout votre ordinateur. Pour remédier à ce problème on place avant la fin de la boucle principale un petit code qui fait « dormir » le programme un petit peu de temps. Bien sûr, ça n'a aucune influence sur le temps de réaction, car il faut s'arrêter le programme moins d'une milliseconde (rappel : pour que l'on discerne deux événements différents, au niveau de l'œil, il faut qu'entre ces deux événements se passent environ 41 ms, donc on ne ressentira rien, mais on soulagera le processeur).

Pour arriver à faire cela, nous allons recourir à une petite fonction à placer en fin de boucle :

Code : PHP

```
<?php
usleep(100);
```

Cette fonction fait dormir le programme pendant le nombre de **microsecondes** renseigné en paramètre. Notez que ce sont des **microsecondes**, donc avec `usleep(100)`, le programme dormira non pas 100 ms (soit 0,1 s), mais 100 µs (soit 0,1 ms !), donc le programme sera très peu ralenti même si votre processeur verra grandement la différence (on passe de 100 % du processeur à 3 % du processeur !).

Nous sommes maintenant prêts à traiter les données. Je vous rappelle donc le code source courant, commenté de nouveau pour une meilleure compréhension :

Code : PHP

```
<?php
$questionPosee = -1; // Initialisation de la question posée à -1.

// Boucle principale du programme.
while(1)
{
    $donnees = fgets($socket,1024); // On lit les données du serveur.
    if($donnees) // Si le serveur nous a envoyé quelque chose.
    {
        echo $donnees;
        // Ici on traite ce que nous a envoyé le serveur.
    }
    usleep(100); // On fait « dormir » le programme afin d'économiser
    l'utilisation du processeur.
}
```

4. Traitement des données

Pour traiter les données, nous allons utiliser des `explode()`. Je sais qu'il y a d'autres manières de traiter un signal (RegEx, fonctions de traitement) mais je préfère les *explode* car il donnent un accès rapide aux différentes informations. Nous allons donc scinder les données par espace, vu que pour le protocole IRC, les différentes infos sont séparées par des espaces, exceptés les textes contenant des espaces qui sont placés en fin de commande, après une **double point** « : ».

Nous allons tout d'abord scinder par espace, puis par un double point « : ».

Code : PHP

```
<?php
$commande = explode(' ', $donnees);
$message = explode(':', $donnees);
```

Maintenant, on va tester les différentes commandes pouvant être envoyées par le serveur.

Rappel : la commande originale `:linkboss!~linkboss@svn44-1-78-228-160-160.fbx.proxad.net PRIVMSG#sdz-quiz:!question est` maintenant scindée en deux *arrays* :

Code : Autre

```
Array
(
    [0] => :linkboss!~linkboss@svn44-1-78-228-160-
160.fbx.proxad.net
    [1] => PRIVMSG
    [2] => #quiz-sdz
```

```

        [3] => :!question
    )

    Array
    (
        [0] =>
        [1] => linkboss!~linkboss@svn44-1-78-228-160-
160.fbx.proxad.net PRIVMSG #quiz-sdz
        [2] => !question
    )

```

Mais avant d'aller plus loin, on va d'abord devoir traiter le cas du PING. En effet, selon la RFC, le serveur nous envoie régulièrement une commande PING afin de vérifier si on ne s'est pas déconnecté en route. Il faudra donc renvoyer un PONG si l'on ne veut pas que notre serveur IRC nous déconnecte comme un rustre. Par contre cette commande diffère un peu des autres, il n'y a pas de :linkboss!~link... car c'est le serveur qui l'envoie et donc PING se retrouvera dans `$commande[0]`.

Donc nous devons rajouter une condition dans notre boucle :

Code : PHP

```

<?php
if($commande[0] == 'PING')
{
    fputs($socket, "PONG " . $commande[1] . "\r\n");
}

```

Maintenant, le *bot* est vraiment prêt à traiter les commandes sans risquer de se faire déconnecter intempestivement. Je sais que selon la partie précédente, on traitait le PING après les questions, mais je préfère faire le gros en dernier, donc la gestion des questions-réponses, on va faire ça en dernier.

Nous allons commencer par vérifier si un opérateur ne nous a pas *kickés* (expulsés) du canal, auquel cas on le rejoint. Pour cela, il suffit de guetter une commande KICK et de vérifier si l'argument du KICK est notre pseudo ou pas. Si ça l'est, c'est qu'on a été kické, et donc on fait une commande JOIN. En réalité, ça ressemble à ça :

Code : PHP

```

<?php
if($commande[1] == 'KICK')
{
    if($commande[3] == 'QuizBot')
    {
        fputs($socket, "JOIN #quiz-sdz\r\n");
    }
}

```

Maintenant, nous pouvons traiter les messages.

Pour commencer, les messages sont envoyés avec la commande PRIVMSG. Il faut donc détecter qu'un client envoie un PRIVMSG. Nous allons donc utiliser le code suivant :

Code : PHP

```

<?php
if($commande[1] == 'PRIVMSG')
{
    // Ici, on traite le message.
}
?>

```

Maintenant, traitons les commandes. Avec ce *bot*, les commandes, pour qu'elles soient reconnues, seront préfixées par le caractère « ! ». C'est le caractère le plus utilisé par les *bots*, donc je ne vais pas « déroger à la règle », même s'il n'y a pas vraiment de règle. Donc tout d'abord, nous allons tester si le premier caractère du message (contenu dans `$message[2]`) est un « ! ».

Code : PHP

```
<?php
if($message[2][0] == '!')
{
    // Maintenant on traite les commandes.
}
```

J'ai mis `$message[2][0]` pour me référer au premier caractère du message. Ensuite, nous allons traiter les commandes. En fait, ce ne sont pas les commandes qu'on va tester, c'est **la** commande. En effet, il n'y a qu'une commande dans le bot : « !question », qui pose une question. Ensuite, on vérifie si une question est posée, et si oui, on ne fait rien (après, si vous le voulez, vous pourrez mettre un message d'erreur). Après, si aucune question n'est posée, on choisit une question au hasard, dans la liste des questions (via la fonction `rand()`) puis on la pose (et l'on oublie pas de régler la variable `$questionPosee`).

Code : PHP

```
<?php
if(trim($message[2]) == '!question')
{
    if($questionPosee == -1)
    {
        $questionPosee = rand(0, count($questions)-1);
        fputs($socket, "PRIVMSG #quiz-sdz :Question :
".$questions[$questionPosee][0]."\r\n");
    }
}
```

Maintenant, nous allons faire la partie concernant la vérification de la réponse. Mais attention, cette réponse doit être vérifiée uniquement si une question est posée. En conséquence, nous devons ajouter une condition avant toute chose :

Code : PHP

```
<?php
if($questionPosee != -1)
{
    // On vérifie la réponse.
}
```

Ensuite, on doit vérifier si la réponse donnée est la bonne, mais en faisant attention à **ne pas** prendre en compte les majuscules et les accents. Pour les majuscules, rien de plus simple, une bête fonction `strtolower()` qui rendra toute la chaîne entrée par le client en minuscule. Mais pour les accents, il n'existe malheureusement pas de fonction toute faite qui nous élimine tous les accents d'une chaîne. Bon, je suis gentil, je vous donne la fonction toute faite, car nous n'avons pas vraiment à nous concentrer sur ça (mais libre à vous de décortiquer le code). Voici donc le code :

Code : PHP

```
<?php
function normaliser($string)
{
    $a = 'âäåèëêîïüç';
```

```

    $b = 'aaaaaaeiüuc';
    $string = utf8_decode($string);
    $string = strtr($string, utf8_decode($a), $b);
    $string = strtolower($string);
    return utf8_encode($string);
}

```



Remarquez que j'ai inclus le `strtolower()` dans cette fonction. La combinaison des deux (retrait des accents et mise en minuscules) s'appelle la **normalisation**. Il vous suffira de copier-coller cette fonction en début de script, puis de mettre `normaliser($variable)` pour que la variable perde ses majuscules et ses accents. 😊 Notez aussi que cette fonction supporte l'UTF-8.



Dans cette fonction, je n'ai mis que les accents principaux. Pour une version plus complète, voyez sur la page de documentation PHP de la fonction `strtr()` sur l'une des notes d'utilisateurs.

Maintenant, nous pouvons vérifier la réponse entrée par le client, et si elle est bonne, afficher un message du genre « Bravo machin ! » puis remettre la variable `$questionPosee` à -1. Nous obtenons donc le code suivant :

Code : PHP

```

<?php
if(normaliser(trim($message[2])) ==
normaliser(trim($questions[$questionPosee][1])))
{
    $pseudo = explode('!', $commande[0]); // On prend le pseudo
    de la personne qui a entré la bonne réponse.
    $pseudo = substr($pseudo[0], 1); // On enlève le double
    point au début du pseudo.
    fputs($socket, "PRIVMSG #quiz-sdz :Bravo $pseudo ! La réponse
    était bien : ".$questions[$questionPosee][1]. " !");
    $questionPosee = -1; // On réinitialise la question.
}

```

J'ai rajouté deux lignes de codes juste après la condition, il s'agit du traitement de la ligne pour obtenir le pseudo du client ayant entré la bonne réponse. Si vous lisez les rendus d'*explode* du message (un peu plus haut dans cette sous-partie) vous verrez facilement comment on obtient le pseudo seul (et en plus vous avez la solution 😊).

Voilà, c'est tout ! Le *bot* est fini !

Bon, je récapitule le code source en commentant ce que je n'ai pas commenté.

Code : PHP

```

<?php

// Fonction normalisant le test, c'est-à-dire retirant les
majuscules et les accents.
function normaliser($string)
{
    $a = 'äääëèêëîïüüç';
    $b = 'aaaaaaeiüuc';
    $string = utf8_decode($string);
    $string = strtr($string, utf8_decode($a), $b);
    $string = strtolower($string);
    return utf8_encode($string);
}

set_time_limit(0); // On met la durée maximale du script à
l'infini.

```

```

$file = file_get_contents('questions.txt'); // On lit le fichier des
questions dans une chaîne.
$questions = explode("\n",$file); // On sépare chaque ligne (\n est
le caractère signifiant une nouvelle ligne).

// On sépare la question et la réponse.
for($i=0;isset($questions[$i]);$i++)
    $questions[$i] = explode('<|#()#|>',$questions[$i]);
// On retire le dernier élément si celui-ci est vide (réglage d'un
bug relatif aux fichiers).
if($questions[count($questions)-1])
    unset($questions[count($questions)-1]);

$socket = fsockopen('irc.epiknet.org','6667'); // On ouvre la
connexion au serveur en tant que pointeur de fichier.

// Vérification de la bonne connexion :
if(!$socket)
{
    // Si on n'a pas réussi, on affiche un message d'erreur et on
quitte.
    echo 'Impossible de se connecter';
    exit;
}

// On renseigne l'USER : ici, je mets un peu n'importe quoi, vu que
le serveur ne prend en compte que le premier argument (mais qu'il a
besoin de 4 arguments).
fputs($socket,"USER QuizBot QuizBot Quiz Quiz\r\n");
// On donne le NICK :
fputs($socket,"NICK QuizBot\r\n");

$continuer = 1; // On initialise une variable permettant de savoir
si l'on doit continuer la boucle.
while($continuer) // Boucle principale.
{
    $donnees = fgets($socket, 1024); // Le 1024 permet de limiter la
quantité de caractères à recevoir du serveur.
    $retour = explode(':', $donnees); // On sépare les différentes
données.
    // On regarde si c'est un PING, et, le cas échéant, on envoie
notre PONG.
    if(rtrim($retour[0]) == 'PING')
    {
        fputs($socket,'PONG :'.$retour[1]);
        $continuer = 0;
    }
    if($donnees) // Si le serveur a envoyé des données, on les
affiche.
        echo $donnees;
}

fputs($socket,"JOIN #quiz-sdz\r\n"); // On rejoint le canal #quiz-
sdz.

$questionPosee = -1; // Initialisation de la question posée à -1.

// Boucle principale du programme :
while(1)
{
    $donnees = fgets($socket,1024); // On lit les données du serveur.
    if($donnees) // Si le serveur nous a envoyé quelque chose.
    {
        echo $donnees;
        $commande = explode(' ', $donnees);
        $message = explode(':', $donnees);
        if($commande[0] == 'PING') // Si c'est un PING, on renvoie un
PONG.
        {

```

```

    fputs($socket, "PONG ".$commande[1]."\r\n");
}
if($commande[1] == 'KICK') // S'il y a une expulsion du canal.
{
    if($commande[3] == 'QuizBot') // Si c'est le bot qui est expulsé, on rejoint le canal
    {
        fputs($socket, "JOIN #quiz-sdz\r\n");
    }
}
if($commande[1] == 'PRIVMSG') // Si c'est un message.
{
    if($message[2][0] == '!') // Si c'est une commande.
    {
        if(trim($message[2]) == '!question') // Si on demande une question.
        {
            if($questionPosee == -1) // Si aucune question n'est posée, on en choisit une au hasard.
            {
                $questionPosee = rand(0, count($questions)-1);
                fputs($socket, "PRIVMSG #quiz-sdz :Question : ".$questions[$questionPosee][0]."\r\n"); // On envoie la question courante.
            }
        }
        if($questionPosee != -1) // Si une question est posée (ici on analyse tous les messages).
        {
            if(normaliser(trim($message[2])) == normaliser(trim($questions[$questionPosee][1]))) // Si la réponse est bonne.
            {
                $pseudo = explode('!', $commande[0]); // On prend le pseudo de la personne qui a entré la bonne réponse.
                $pseudo = substr($pseudo[0], 1); // On enlève le double point au début du pseudo.
                fputs($socket, "PRIVMSG #quiz-sdz :Bravo $pseudo ! La réponse était bien : ".$questions[$questionPosee][1]."\r\n");
                $questionPosee = -1; // On réinitialise la question posée.
            }
        }
    }
}
usleep(100); // On fait « dormir » le programme afin d'économiser l'utilisation du processeur.
}
// Ça y est, c'est fini. :D

```

Maintenant, vous pouvez crâner avec « votre » *bot*, et vous amuser à poser des questions.

Pour ceux que ça intéresse, voici le code d'un autre *bot* de quiz un peu plus évolué, gérant les classements, ainsi que les quiz à plusieurs questions. Globalement, le principe de fonctionnement reste le même, mais il y a quelques variables en plus. Je vous laisse la corvée le plaisir de l'étudier tout seul.

Secret ([cliquez pour afficher](#))

Code : PHP

```

<?php
set_time_limit(0);
$server='irc.quakenet.org';
$port='6667';
$name='QuizdeBot';
$user='QuizdeBot';
$chan = '#LeelaBot';
$operators = array();

```

```

$voice = array();
$users_online = array();
$admins = array('linkboss');

$socket = fsockopen( $server , $port , $errno, $errstr, 1); //
Connexion au serveur.

if (!$socket) exit(); // Si la connexion n'a pas eu lieu, on
arrête le script (exit()).
fputs($socket , "USER $name $chan $user .\r\n" );

fputs($socket , "NICK $name\r\n" ); // Pseudo du bot.

stream_set_timeout($socket, 0);

$continuer = 1;

/*****/
while($continuer) // Boucle pour la connexion.
{

    $donnees = fgets($socket, 1024);
    $retour = explode(':', $donnees);
    if(rtrim($retour[0]) == 'PING')
        fputs($socket, 'PONG :'.$retour[1]);
    if($donnees)
        echo $donnees;

    if(preg_match('#:(.+):End Of /MOTD Command.#i', $donnees))
        $continuer = 0;
}
fputs($socket , "JOIN $chan\r\n" );

$file = file_get_contents('questions.txt'); // On lit le fichier
des questions dans une chaîne.
$questions = explode("\n", $file); // On sépare chaque ligne (\n
est le caractère signifiant une nouvelle ligne).

// On sépare la question et la réponse :
for($i=0;isset($questions[$i]);$i++)
    $questions[$i] = explode(' \ ', $questions[$i]);

shuffle($questions);
$i = 0;
$newquestions = array();
foreach($questions as $question)
{
    $newquestions[$i] = $question;
    $i++;
}
$questions = $newquestions;
array_pop($questions);

print_r($questions);
$continuer = 1;
$quiz = 0;
$questionEnCours = -1;
while($continuer)
{
    $donnees = fgets($socket, 1024);
    if($donnees)
    {
        $array = explode(':', $donnees);
        $msg=$array[2];
        $pseudo= explode('!', $array[1]);
        $pseudo = $pseudo[0];
        $infos = explode(' ', $array[1]);
        $chan = $infos[2];
    }
}

```



```

$cmd = explode(' ', $array[2]);
if(rtrim($array[0]) == 'PING')
{
    fputs($socket, 'PONG :'. $array[1]);
    echo $donnees;
}
elseif(rtrim($infos[1]) == 'PRIVMSG')
{
    print_r($cmd);
    if(rtrim($cmd[0]) == '.quiz')
    {
        if($quiz == 0)
        {
            $quiz = 1;
            $questionEnCours = 0;
            $questionPosee = 0;
            $highscore = array();
            $questionTime = time()+15;
            fputs($socket, "PRIVMSG $chan :Un quiz a été lancé ! Tentez de
répondre aux questions pour être le meilleur !\r\n");
            fputs($socket, "PRIVMSG $chan :Vous disposez de 30 secondes
pour répondre. Au total 100 questions\r\n");
            fputs($socket, "PRIVMSG $chan :Utilisez un espace pour séparer
les milliers des nombres. Accents obligatoires.\r\n");
            fputs($socket, "PRIVMSG $chan :Départ dans 15
secondes...\r\n");
        }
        else
            fputs($socket, "NOTICE $pseudo :Un quiz est déjà lancé -
_\r\n");
    }

    if(rtrim($cmd[0]) == '.question')
    {
        if($questionPosee == 1)
        {
            fputs($socket, "NOTICE $pseudo :Question $questionNombre :
".$questions[$questionEnCours][0]."\r\n");
        }
        else
        {
            fputs($socket, "NOTICE $pseudo :Il n'y a aucune question
posée...\r\n");
        }
    }

    if(rtrim($cmd[0]) == '.classement')
    {
        if($quiz == 1)
        {
            $classement = '';
            arsort($highscore);
            $nicks = array_keys($highscore);
            $i = 0;
            for($i = 0; $i < count($highscore); $i++)
            {
                $place = $i+1;
                $classement .= "$place. ".$nicks[$i].
(".$highscore[$nicks[$i]].") ";
            }
            fputs($socket, "NOTICE $pseudo :Classement :
$classement\r\n");
        }
        else
        {
            fputs($socket, "NOTICE $pseudo :Il n'y a aucun quiz de
lancé...\r\n");
        }
    }

    if(rtrim($cmd[0]) == '.stopquiz' && in_array($pseudo, $admins))

```

```

    {
        fputs($socket,"PRIVMSG $chan :Le quiz est terminé ! Classement
: \r\n");
        $classement = '';
        arsort($highscore);
        $nicks = array_keys($highscore);
        print_r($nicks);
        print_r($highscore);
        $i = 0;
        for($i = 0;$i < count($highscore);$i++)
        {
            $place = $i+1;
            $classement .= "$place. ".$nicks[$i].
("$highscore[$nicks[$i]].") ";
        }
        fputs($socket,"PRIVMSG $chan :$classement\r\n");
        $quiz = 0;
    }

    if(rtrim($cmd[0]) == '.help')
    {
        fputs($socket,"NOTICE $pseudo :Liste des commandes : .quiz
(pour lancer un quiz), .classement (pour voir le classement
actuel), .question (pour réafficher la question courante), .help
(affiche cette aide).\r\n");
    }
    elseif($questionPosee == 1 && time() <= $questionTime &&
strtolower(rtrim(join(' ', $cmd))) ==
strtolower($questions[$questionEnCours][1]))
    {
        fputs($socket,"PRIVMSG $chan :Bravo $pseudo ! La réponse était
bien : ".$questions[$questionEnCours][1]."\r\n");
        $questionPosee = 0;
        if($highscore[$pseudo])
            $highscore[$pseudo]++;
        else
            $highscore[$pseudo] = 1;
        print_r($highscore);
        $questionTime = time()+15;
        $questionEnCours++;
        if(count($questions) > $questionEnCours || $questionEnCours >
99)
            fputs($socket,"PRIVMSG $chan :Prochaine question dans 15
secondes...\r\n");
    }
}
}
if($quiz == 1)
{
    if($questionPosee == 0 && time() >= $questionTime)
    {
        $questionNombre = $questionEnCours+1;
        fputs($socket,"PRIVMSG $chan :Question $questionNombre :
".$questions[$questionEnCours][0]."\r\n");
        $questionPosee = 1;
        $questionTime = time()+30;
    }

    if($questionEnCours >= count($questions) || $questionEnCours
> 99)
    {
        fputs($socket,"PRIVMSG $chan :Le quiz est terminé !
Classement : \r\n");
        $classement = '';
        arsort($highscore);
        $nicks = array_keys($highscore);
        print_r($nicks);
        print_r($highscore);
        $i = 0;
        for($i = 0;$i < count($highscore);$i++)

```

```

        {
            $place = $i+1;
            $classement .= "$place. ".$nicks[$i]. "
($".$highscore[$nicks[$i]].") ";
        }
        fputs($socket, "PRIVMSG $chan :$classement\r\n");
        $quiz = 0;
    }
    if($questionPosee == 1 && time() > $questionTime)
    {
        fputs($socket, "PRIVMSG $chan :Le temps est écoulé ! la bonne
réponse était : ".$questions[$questionEnCours][1]."\r\n");
        $questionTime = time()+15;
        $questionPosee = 0;
        $questionEnCours++;
        $unanswered++;
        if(count($questions) > $questionEnCours)
            fputs($socket, "PRIVMSG $chan :Prochaine question dans 15
secondes...\r\n");
    }
}
usleep(100);
}

```

Petit récapitulatif

Petit récapitulatif de tout ce qui a été vu sur le protocole IRC dans ce chapitre :

- pour ouvrir une connexion avec un serveur IRC, on utilise (la plupart du temps) `fsockopen()` sur le port 6667 d'un serveur IRC ;
- pour mettre son pseudo, il suffit d'utiliser la commande `NICK <pseudo>` ;
- pour régler votre *user*, on utilise la commande `USER <user> <user> <user> <user>` (mettre 4 fois le même paramètre marche, mais uniquement le premier est pris en compte) ;
- quand le serveur nous envoie un `PING <quelque_chose>`, lui répondre par `PONG <la_chose_envoyee>` ;
- pour envoyer un message, utiliser la commande `PRIVMSG <canal/pseudo> :<message>` ;
- **ne pas oublier** de mettre `\r\n` à la fin des commandes.

Et puis quelques trucs en plus que nous n'avons pas vus dans ce tutoriel :

- pour envoyer un message en privé à quelqu'un, vous pouvez utiliser (et privilégier) la commande `NOTICE <pseudo> :<message>` au lieu du `PRIVMSG` ;
- pour quitter un canal, vous pouvez utiliser la commande `PART <canal> : <raison>`. Notez que la raison est optionnelle (si vous n'en mettez pas, ne mettez pas de double point) ;
- pour vous déconnecter d'un serveur, utilisez la commande `QUIT` ;
- pour joindre plusieurs canaux, utilisez la commande `JOIN <canal>` autant de fois que c'est nécessaire ;
- pour changer de pseudo, il suffit d'utiliser la commande `NICK` une seconde fois ;
- pour obtenir la liste des clients dans un canal, utilisez la commande `NAMES <canal>` ;
- pour expulser quelqu'un d'un canal, utilisez la commande `KICK <canal> <pseudo> : <raison>`. Ici aussi, la raison est facultative (n'oubliez pas d'enlever le double point).
- pour tous les autres détails relatifs à IRC, référez-vous à la RFC 1459 et à la RFC 2812 (Google est votre ami).

Q.C.M.

Le premier QCM de ce cours vous est offert en libre accès.

Pour accéder aux suivants

[Connectez-vous](#) [Inscrivez-vous](#)

Le protocole IRC est décrit dans les RFC

- ○ 1459 et 2812.

- ☐ 259 et 1869.
- ☐ 1252 et 3102.

Pour envoyer un message sur IRC, la commande est :

- ☐ PRIVMSG <message> <pseudo/canal>
- ☐ PRIVMSG <pseudo/canal> <message>
- ☐ LINKS <pseudo/canal> <message>
- ☐ LINKS <message> <pseudo/canal>

Pour se connecter à un canal IRC, la commande est :

- ☐ CHANNEL <canal>
- ☐ CONNECT <canal>
- ☐ JOIN <canal>

Correction !

[Statistiques de réponses au QCM](#)

Voilà, maintenant, vous savez utiliser IRC avec PHP. Les applications possibles sont nombreuses (*bots*, clients, serveurs...), alors programmez bien. 😊

Partager



Ce tutoriel a été corrigé par les [zCorrecteurs](#).