Introduction à la Common Gateway Interface
Exécution de programmes externes

Frédéric Couchet, mad@bocal.cs.univ-paris8.fr
APRIL, http://www.april.org
v 1.0, 11 avril 1997

Ce document explique les principes généraux des scripts CGI et la manière d'en écrire. Ce document s'adresse aux personnes connaissant déjà HTML et qui désirent rendre dynamiques leurs pages, créer des programmes gérant des formulaires ou encore s'interfacer avec des bases de données.

Contents

1 Introduction **1**

1.1 La définitions d'un script CGI 1

1.2 Pourquoi utiliser les scripts CGI 1

1.3 Prérequis et choix techniques 2

1.4 Installation du serveur Web 2

2 Notions fondamentales **3**

3 Premier programme **4**

3.1 Source du programme 4

4 Saisie et traitement de données **5**

4.1 Récupération des informations 7

4.2 les variables d'environnement 10

5 Debugger un script CGI **10**

6 That's all folks ! **12**

7 Copyright **13**

1 Introduction

1.1 La définitions d'un script CGI

En simplifiant, un script CGI est tout simplement un programme pouvant être exécuté par un serveur HTTP.

1.2 Pourquoi utiliser les scripts CGI

Vous savez créer des pages au format HTML (*Hyper Text Markup Language*) et les publier sur le WEB (*World Wide Web*). Ainsi, vous maitrisez les bases de HTML (voir même tous les détails de HTML 3.2), et

vous êtes alors prêt(e) à passer à l'étape suivante: la création et gestion de pages dynamiques, et l'interfaçage avec des programmes externes.

Vous pourrez alors effectuer des traitements automatiques à partir de vos pages HTML. Un exemple classique nous étant fourni par les moteurs de recherche qui, à partir de mots clés saisis dans un formulaire, vous affichent l'ensemble des pages au format HTML contenant ces mots clés. La recherche étant effectuée par un programme externe.

La solution est l'utilisation de la Common Gateway Interface (CGI). CGI est une interface permettant l'exécution de programmes externes par un serveur HTTP. Plus généralement, CGI est en fait un standard pour l'écriture de passerelles entre des serveurs d'informations tel que HTTP et des programmes externes. L'interface CGI permet de faire communiquer le programme et le serveur HTTP.

1.3 Prérequis et choix techniques

Dans ce document, je suppose connu les bases du langage HTML, le minimum pour pouvoir faire une page. Pour l'écriture des programmes, la connaissance d'un langage de programmation plus ou moins évolué est bien entendu nécessaire (par exemple, C, Perl, Python ...). Nous supposons un serveur Web tournant sur une machine Unix (tous les exemples présentés dans ce document ont été réalisés sur une machine Linux 2.0 et le serveur HTTP Apache 1.2b3).

1.4 Installation du serveur Web

Autant l'écriture de pages au format HTML nécessitait juste un navigateur pour visualiser le résultat, autant l'écriture de scripts CGI nécessite la présence d'un serveur HTTP tournant sur la machine (en effet, n'oublions pas que c'est le serveur qui exécute le programme).

Donc première étape, il nous faut installer un serveur HTTP. Notre choix se porte sur Apache (http://www.apache.org/. Il est puissant, largement le plus utilisé (40% environ des sites) et libre. On récupère la dernière version sur, par exemple, ftp://ftp.ibp.fr/pub/www/apache/dist ou sur le site officiel : ftp://ftp.apache.org/pub/apache/dist. Une fois l'archive récupérée, on désarchive le fichier apache_1.2b8.tar.gz (version 1.2 beta 8 ou une version ultérieure):

```
tar zxvf apache_1.2b8.tar.gz
```

On va dans le répertoire créé, on lit les docs d'installation. En gros, en résumant, cela donne cela :

```
cd apache_1.2b8
cd src
cp Configuration.tmpl Configuration
make
cd ..
mkdir -p /usr/local/etc/httpd
cp -a . /usr/local/etc/httpd
```

Les fichiers de configuration du serveur se trouvent alors dans le répertoire `/usr/local/etc/httpd/conf`. On se déplace dans ce répertoire et on copie tous les fichiers d'exemples sous le nom réel :

```
cd /usr/local/etc/httpd/conf
cp srm.conf-dist srm.conf
cp access.conf-dist access.conf
cp httpd.conf-dist httpd.conf
```

Vous n'avez que peu de modifications à apporter à ces fichiers de configuration. Les commentaires présents devant vous permettre de faire les modifications minimales pour que votre serveur s'exécute correctement. En cas de besoin, lisez la documentation livrée avec le serveur.

Par contre, comme le but de ce document est d'écrire des scripts CGI, il faut configurer le serveur pour qu'il puisse exécuter nos programmes. Dans le fichier `srm.conf`, il faut décommenter la ligne suivante (supprimer le dièse en début de ligne):

```
#ScriptAlias /cgi-bin/ /usr/local/etc/httpd/cgi-bin/
```

Ainsi, si votre machine s'appelle `yoda`, lorsque le navigateur accédera à l'url `http://yoda/cgi-bin/date.cgi` le serveur saura que le fichier `date.cgi` se trouve dans le répertoire `/usr/local/etc/httpd/cgi-bin` et qu'il devra l'exécuter (voir 2).

Vous pouvez également configurer votre serveur pour qu'il puisse exécuter n'importe quel fichier, où qu'il se trouve, pourvu qu'il ait pour extension `.cgi`. Pour cela, décommentez, dans `srm.conf`, la ligne suivante:

```
#AddHandler cgi-script .cgi
```

Une fois la configuration terminée, pour lancer votre serveur il suffit de taper la commande suivante :

```
/usr/local/etc/httpd/src/httpd -f /usr/local/etc/httpd/conf/httpd.conf
```

Faites un `ps`pour vérifier que votre serveur s'exécute normalement. Chaque fois que vous modifiez un fichier de configuration, n'oubliez pas de relancer le serveur.

Si vous désirez que votre serveur soit lancé au démarrage de la machine, rajoutez la ligne de commande dans un des fichiers de démarrage (fichiers `/etc/rc.*`)

2 Notions fondamentales

Lorsque vous tapez un URL (*Uniform Ressource Locator*), par exemple http://www.april.org/april.html, votre navigateur préféré va se connecter au serveur Web indiqué, c'est à dire le programme installé sur la machine distante (dans notre exemple `www.april.org`), et qui écoute sur le port 80. Le serveur va chercher le document en question sur son disque et l'envoie au navigateur, en utilisant le protocole HTTP (*HyperText Transfer Protocol*). Il peut s'agir de n'importe quel type de fichier, un fichier au format HTML, une image, un fichier son etc.

Pour savoir comment et où chercher le document, le serveur possède un ensemble de fichiers de configuration chargés en mémoire à son lancement (qui, dans le cas de notre installation, se trouvent dans le répertoire `/usr/local/etc/httpd/conf`). Dans notre exemple, le serveur ira chercher le fichier `april.html` dans le répertoire `/usr/local/etc/httpd/htdocs/` (répertoire racine du serveur). Un autre exemple d'URL est http://www.bocal.cs.univ-paris8.fr/~drieu/index.html. Le caractère ~ indique que la chaîne qui suit correspond à un nom de login sur la machine du serveur. Ainsi, dans ce cas, le serveur va chercher le document en question dans le répertoire `/home/drieu/public_html` (selon la configuration classique du serveur). Le fichier `index.html` se trouve dans ce répertoire. Les utilisateurs ont ainsi la possibilité de créer un répertoire `public_html` (accessible en lecture et exécution) dans leur compte pour publier sur le Web.

Mais ce type de document est statique, or il est possible d'accéder à de l'information dynamique. Une possibilité est l'utilisation de scripts CGI. En effet, ce programme est exécuté, par le serveur, en temps réel sur la machine distante, au moment où le navigateur fait une requête vers ce programme.

Un exemple assez puissant étant l'accès, via un formulaire HTML, à une base de données. Par l'intermédiaire du formulaire, vous saisissez les nom et prénom d'une personne. A l'aide d'un bouton 'valider' présent sur le

formulaire, le serveur invoque alors un programme qui, à partir des informations saisies dans le formulaire, va interroger le moteur de votre base de données et envoyer en réponse un page au format HTML, contenant les informations concernant cette personne, que votre navigateur va alors afficher.

Mais ne nous attaquons pas tout de suite à des requêtes SQL, et commençons par beaucoup plus simple. Pour montrer le dynamisme de notre page, nous allons faire un programme qui affiche l'heure courante dans une page HTML (exemple d'école, car beaucoup plus facilement réalisable avec les `Server Side Includes`).

3 Premier programme

Pour écrire un script CGI, nous pouvons utiliser n'importe quel langage de programmation. Nous devons simplement pouvoir lire sur l'entrée standard, écrire sur la sortie standard et accéder aux variables d'environnement.

Nous allons écrire notre premier programme que nous appelerons `date.cgi`. Nous devons penser aux permissions de ce fichier. Le programme doit être exécutable par tous, et lisible par tous dans le cas de script shell, perl ... En effet, n'oublions pas que c'est le serveur HTTP qui va exécuter notre programme. Or, le serveur tourne dans la majeure partie des cas sous l'utilisateur `nobody`, et donc il n'a pas de droits particuliers (retenons bien que le serveur ne s'exécute pas sous l'utilisateur qui a écrit le script).

Nous installerons notre progamme dans le répertoire `/usr/local/etc/httpd/cgi-bin`, qui est le répertoire par défaut des scripts CGI (encore faut-il que l'administrateur du système nous en laisse le droit). En effet, n'importe qui n'a pas le droit de faire des scripts CGI. L'administrateur peut limiter la possibilité de mettre des scripts CGI que dans un seul répertoire. Il contrôlera ainsi beaucoup plus facilement ces scripts. Car, il faut le savoir, les scripts CGI sont une bonne source de trous de sécurité.

Ainsi, le serveur sait que tous les fichiers se trouvant dans ce répertoire devront être exécutés et non pas envoyés directement au navigateur, et ce seront les seuls dans ce cas. C'est le programme qui devra envoyer les informations au navigateur, par l'intermédiaire de la sortie standard (d'où l'obligation de pouvoir écrire sur la sortie standard). Donc, notre programme doit produire sur sa sortie standard quelque chose de compréhensible par le navigateur. Ainsi, lorsque votre navigateur accédera à l'URL correspondant à ce fichier (par exemple: http://www.april.org/cgi-bin/date.cgi), le serveur exécutera le programme `date.cgi` qui produira un fichier au format HTML contenant la date courante. Le navigateur affichera alors le résultat.

3.1 Source du programme

Notre premier programme est écrit sous la forme d'un script shell. Voici le source de date.cgi:

```
#!/bin/sh

tmp=‘/bin/date‘

cat << EndFile
Content-type: text/html

<HTML><HEAD><TITLE>Script Cgi</TITLE></HEAD>
<BODY>

<CENTER>

<H1>La date courante sur le serveur est</H1>
```

```
$tmp

</CENTER>

</BODY>
</HTML>

EndFile
```

On fait un `chmod 555 date.cgi` et on copie le fichier dans le répertoire `/usr/local/etc/httpd/cgi-bin`. Ensuite, on peut utiliser notre navigateur pour voir le résultat. On accède à l'URL http://www.april.org/cgi-bin/date.cgi et on obtient le résultat suivant:

```
La date courante sur le serveur est
 Sun Mar 30 11:48:54 GMT+0100 1997
```

Au niveau du source, on affecte à la variable `tmp` le résultat de la commande `/bin/date` (`$tmp` permet ensuite d'avoir accès au contenu de la variable `tmp`). Ensuite, il faut juste savoir que la commande `cat << EndFile` permet d'afficher sur la sortie standard tout ce qui suit jusqu'à ce que l'on rencontre une ligne formée uniquement par `EndFile` (la chaîne suivant immédiatement <<). En fait, tout ce qui se trouve entre `<< EndFile` et `EndFile` est utilisé comme entrée standard de la commande cat (la chaine EndFile est choisie arbitrairement).[1]

Ce qui est important de noter, c'est la première ligne écrite sur la sortie standard (`Content-type: text/html`), c'est à ce niveau que l'on spécifie le type de données que l'on envoie au navigateur. Cette ligne fait en fait partie du header HTTP. Ce header est indispensable pour que le navigateur puisse interpréter correctement les données reçues (les autres parties du header seront ajoutées par le serveur HTTP). Le type "text/html" est le type MIME standard pour un document HTML. Si notre programme devait générer une image au format Gif (par exemple pour implémenter un compteur graphique), nous aurions utilisé le type "image/gif". Le header doit obligatoirement être terminé par une ligne vide (comme le header de tous les protocoles). Cette ligne sépare l'en-tête de la réponse (c'est à dire les informations au sujet de la réponse) du contenu de la réponse.

Ce qu'il faut retenir à ce niveau, c'est qu'un script CGI doit produire sur sa sortie standard quelque chose de compréhensible par le navigateur, donc, dans la majorité des cas, une sortie au format HTML. La seule obligation est la présence, sur la première ligne, du header `Content-type`. Les autres parties du header seront ajoutées par le serveur HTTP. C'est pourquoi ce type de script est appelé *Parsed Header Script*. Il existe un autre type de script CGI, appelé *Non Parsed Header Script*. Pour ce type de script, le serveur ne rajoute rien, et c'est donc le script lui_même qui a la charge de fournir un header complet, conforme au protocole HTTP.

Voilà, nous avons écris notre premier script CGI. On y accède directement par le navigateur.

Faites le test: http://www.april.org/cgi-bin/date.cgi.

4 Saisie et traitement de données

L'exemple précédent était relativement simple, et illustrait l'appel direct d'un script CGI. Maintenant nous allons aborder un problème un peu plus complexe et intéressant. L'exercice consiste à pouvoir saisir des

[1] Ce type de script shell où les données sont directement incluses dans le script s'appellent des documents *in-line* ou des *here documents*.

données dans un formulaire et, grâce à un script CGI, effectuer un traitement quelconque sur ces données (ce qui est le mode de fonctionnement des moteurs de recherche).

Ceci nécessite deux étapes :

- la création d'un document HTML contenant un formulaire, ce qui permettra à l'utilisateur de saisir des données.
- l'écriture d'un programme traitant ces données.

Pour pouvoir saisir des données dans une page HTML, nous allons utiliser la balise FORM. Cette balise permet de réaliser un questionnaire. Les réponses saisies par l'utilisateur sont codées par le navigateur et transmises au serveur HTTP. Utilisons l'exemple suivant (fichier `form.html`) :

```
<HTML><HEAD><TITLE>Formulaire simple</TITLE></HEAD>
<BODY>
<H2>R&eacute;pondez aux questions suivantes</H2>
<FORM ACTION="http://www.april.org/cgi-bin/treat.pl" METHOD=GET>
Prénom : <INPUT TYPE="text" NAME=prenom SIZE=20><BR>
Nom : <INPUT TYPE="text" NAME=nom SIZE=20><BR>
Age : <SELECT NAME=age>
        <OPTION>- de 18 ans
        <OPTION>19 &agrave; 40 ans
        <OPTION>41 &agrave; 60 ans
        <OPTION>+ de 60 ans
</SELECT><BR>
<INPUT TYPE=submit VALUE="Envoyer"> <INPUT TYPE=reset VALUE="Remettre
&agrave; z&eacute;ro">
</FORM>
</BODY>
</HTML>
```

L'utilisateur pourra ainsi entrer son prénom, nom (champ de type texte) et sélectionner l'une des quatres valeurs possibles pour l'âge (champ de type SELECT).

Le champ d'un formulaire est identifié par l'attribut NAME. Les données sont transmises au serveur sous la forme de paires : name=value, codées au format URL et séparées par le symbole &. Le format de codage URL est le suivant :

- les caractères non ASCII (ceux dont le code est supérieur à 128) sont remplacés par la chaîne de caractères **%xx** où **xx** représente le code ASCII du caractère au format hexadécimal.
- les caractères réservés sont également remplacés par leur valeur hexadécimale.
- le caractère espace est remplacé par le caractère +.

il est absolument nécessaire de connaître ce codage pour écrire des scripts CGI. En effet, le programme doit savoir décoder ces chaînes. Il est également possible, en connaissant ce codage, d'appeler directement un script sans passer par la page HTML appelant le script.

Le script CGI est identifié par l'attribut ACTION de la balise FORM, cet attribut contenant l'URL du programme externe. Dans notre exemple, le script est le fichier `treat.pl` qui se trouve sur le serveur `www.april.org` (dans le répertoire `/usr/local/etc/httpd/cgi-bin/` comme nous l'avons vu précédemment). L'attribut METHOD de la balise FORM spécifie le mode de transfert des données vers le serveur. On peut en distinguer deux : GET et POST.

Dans la méthode GET le navigateur concatène à l'URL précisée par l'attribut ACTION, le symbole **?** et la chaîne contenant les données saisies par l'utilisateur. Et lorsque l'utilisateur clique sur le bouton "Envoyer" on accède à cet URL.

Dans la méthode POST la chaine contenant les données saisies par l'utilisateur est insérée dans le corps de la requête HTTP.

Si dans notre formulaire, on saisit pour le prénom "Marcel", pour le nom "Dugenou" et pour l'âge on sélectionne "41 à 60 ans", la chaîne transmise sera alors la suivante :

```
prenom=Marcel&nom=Gnou&age=41+%E0+60+ans
```

Le `%E0` correspond au caractère `à`. Alors, dans le cas de la méthode GET on accédera à l'URL suivante:

```
http://www.april.org/cgi-bin/treat.pl?prenom=Marcel&nom=Gnou&age=41+%E0+60+ans
```

Ce qui montre, par ailleurs, que l'on peut donc directement accéder à un script CGI, en lui passant ses paramètres de cette façon. Ceci peut être très dangereux si on n'écrit pas ses scripts avec beaucoup de précautions.

Dans le cas de la méthode POST on accédera à l'URL suivante:

```
http://www.april.org/cgi-bin/treat.pl
```

et la chaîne `prenom=Marcel&nom=Gnou&age=41+%E0+60+ans` est insérée dans le corps de la requête, et donc le programme ira récupérer cette chaîne sur son entrée standard.

4.1 Récupération des informations

Nous avons écris le formulaire, on a vu comment transmettre les informations au script (GET ou POST), maintenant on va écrire le programme.

Précisons tout de suite le rôle du programme externe :

- d'abord récupérez les données saisies par l'intermédiaire du formulaire.
- traiter ces données
- enfin, fournir sur sa sortie standard le résultat qui sera alors transmis au client (le navigateur).

La première chose qui nous intéresse est d'extraire l'information envoyée par le navigateur à l'aide du formulaire. La méthode dépend en fait de celle choisie dans le formulaire.

Dans le cas de la méthode GET, l'information est contenue dans la variable d'environnement `QUERY_STRING`, qui a pour longueur la valeur de la variable `CONTENT_LENGTH`. Ainsi dans notre exemple précédent, la variable contiendra la chaîne `prenom=Marcel&nom=Gnou&age=41+%E0+60+ans`.

Dans le cas de la méthode POST, le programme récupère les informations sur son entrée standard.

Pour mettre en pratique ces principes, nous allons juste écrire un programme qui récupère les informations, les décode et qui écrit sur sa sortie standard une page HTML contenant les données décodées (que le navigateur affichera alors). La partie traitement des données n'est pas abordée ici car cela dépend de ce que doit faire votre programme, et il n'y a rien à dire de spécifique par rapport au sujet.

Nous allons utiliser deux langages différents : Perl et le shell.

En général, la phase de récupération des données est toujours la même, ce qui permet d'écrire des fonctions d'extraction réutilisables.

Voyons le code source de `treat.pl` :

```
#!/usr/bin/perl

# les donnees sont envoyees par methode GET
# donc on recupere les donnees dans la variable
# d'environnement QUERY_STRING
$buffer=$ENV{"QUERY_STRING"};

# on split la chaine de donnees en des paires name=value
local(@champs) = split(/&/, $buffer);
local($donnees) = "";

# affichage du debut du code HTML
printf STDOUT "Content-type: text/html\n\n";
printf STDOUT "<HTML><HEAD>";
printf STDOUT "<TITLE>Reponse au questionnaire</TITLE>";
printf STDOUT "</HEAD>";
printf STDOUT "<BODY BGCOLOR=\"#ffffff\">";

printf STDOUT "<H1>R&eacute;sultat du traitement de votre questionnaire</H1>";
printf STDOUT "<H2>Chaine de donn&eacute;es reçue par le programme</H2>";
printf STDOUT "QUERY_STRING <STRONG>%s</STRONG>",$buffer;
printf STDOUT "<H2>Liste des informations d&eacute;cod&eacute;es</H2>";
printf STDOUT "<UL>";
printf STDOUT "<BL>";

# recuperation et mise en forme des donnees
# on parcourt la liste des paires name=value
foreach $i (0 .. $#champs) {
    # On convertit les plus en espaces
    $champs[$i] =~ s/\+/ /g;

    # On separe chaque champ en une cle et sa valeur
    ($key, $val) = split(/=/,$champs[$i],2);

    # On convertit les %XX de leur valeur hexadecimale en alphanumerique
    $key =~ s/%(..)/pack("c",hex($1))/ge;
    $val =~ s/%(..)/pack("c",hex($1))/ge;

    # on affiche le resultat
    printf STDOUT "<LI><STRONG>%s:</STRONG>%s\n",$key,$val;
}

printf STDOUT "</BL>";
printf STDOUT "</UL>";

printf STDOUT "</BODY>";
printf STDOUT "</HTML>";
```

Etant donné que la méthode choisie est GET, on récupère les données dans la variable `QUERY_STRING`, par la ligne: `$buffer=$ENV{"QUERY_STRING"};`. Si nous avions, au niveau du formulaire, choisi la méthode POST, nous aurions récupéré les données par la ligne: `read(STDIN,$buffer,$ENV{"CONTENT_LENGTH"});`. La variable `CONTENT_LENGTH` contenant la longueur de `QUERY_STRING`.

Pour bien comprendre l'exemple, il faut évidemment connaître Perl. Les commentaires mis dans le programme l'explicitant un peu.

Nous pouvons écrire le programme avec d'autres langages, voici le code source de `treat.sh` écrit en shell.

```
#!/bin/sh
#

if [ "$REQUEST_METHOD" = "POST" ]; then
        read QUERY_STRING
fi

# on split la chaine de donnees en des paires name=value
OPTS=`echo $QUERY_STRING | sed 's/&/ /g'`

echo "Content-type: text/html"
echo ""
echo "<HTML><HEAD>"
echo "<TITLE>Reponse au questionnaire</TITLE>"
echo "</HEAD>"
echo "<BODY BGCOLOR=\"#ffffff\">"

echo "<H1>R&eacute;sultat du traitement de votre questionnaire</H1>"
echo "<H2>Chaine de donn&eacute;es reçue par le programme</H2>"
echo "QUERY_STRING <STRONG>"$QUERY_STRING
echo "</STRONG>"
echo "<H2>Liste des informations d&eacute;cod&eacute;es</H2>"

# recuperation et mise en forme des donnees
# on parcourt la liste des paires name=value
echo "<UL>"
echo "<BL>"
for opt in $OPTS
do
  NAME=`echo $opt | sed 's/=/ /g' | awk '{print $1}'`
  VALUE=`echo $opt | sed 's/=/ /g' | awk '{print $2}' |  sed 's,%,\\\x,g' | sed 's/+/ /g'`
  printf "<LI><STRONG>$NAME:</STRONG>$VALUE"
done

echo "</BL>"
echo "</UL>"
echo "</BODY></HTML>"
```

Il suffit de remplacer `treat.pl` par `treat.sh` dans la balise FORM du formulaire.

L'algorithme de décodage utilisé est le suivant (la chaîne à traîter étant dans notre exemple `prenom=Marcel&nom=Gnou&age=41+%E0+60+ans` :

- récupération des paires name=value (qui sont séparées par le caractère `&`). En shell, cela se fait par `OPTS=`echo $QUERY_STRING | sed 's/&/ /g'` (on remplace & par un espace)` et en perl on utilise un tableau : `local(@donnees) = split(/&/, $buffer);`
- séparation de chaque paire en `name` et `value`
- pour chaque champ `value`, convertion des `+` en espaces, et tous les `%xx` en leur valeur alphanumérique.

Les deux méthodes GET et POST différencient le moyen de communication entre le serveur et le programme. La méthode GET est limitée par la taille maximale acceptée par le serveur au niveau d'un URL, la méthode POST n'étant pas restreinte par cette limite.

Testez le formulaire et son script sur http://www.april.org/form.html.

4.2 les variables d'environnement

Les scripts CGI peuvent accéder aux variables d'environnement. Selon les systèmes, on peut accéder à un nombre plus ou moins important de variables d'environnement.

Certaines variables sont relatives au serveur HTTP (par exemple `HTTP_HOST` contient l'adresse IP de la machine hébergeant le serveur HTTP). D'autres sont relatives à la connexion client-serveur (par exemple `REMOTE_ADDR` contient l'adresse IP de la machine cliente effectuant la requête. Enfin, certaines sont relatives à la requête (par exemple `QUERY_STRING` contient la chaîne contenant les informations de la requête).

Voici deux programmes permettant l'accès à l'ensemble des variables d'environnement, le premier en C, le deuxième en Perl :

Code de `env.c` :

```
#include <stdio.h>

main(int argc, char **argv, char **env)
{
    int i = 0;

    printf("Content-Type: text/html\n\n");
    printf("<HTML><HEAD><TITLE>Variables</TITLE></HEAD><BODY>\n");

    while(*env){
        printf("%s <BR>\n",*env++);

    }

    printf("</BODY></HTML>\n");
}
```

Code de `env.pl` :

```
#!/usr/bin/perl

print "Content-Type: text/html\n\n";
print "<HTML><HEAD><TITLE>Variables</TITLE></HEAD><BODY>\n";

while(($name,$value) = each %ENV){
    print "$name = $value <BR>\n";
}

print "</BODY></HTML>\n";
```

Il suffit alors de compiler `env.c` et de copier l'exécutable résultant dans le répertoire des cgi-bin du serveur. Pour le script Perl, il suffit de le copier dans le répertoire, en n'oubliant pas de le rendre lisible et exécutable par tous.

Cliquez sur les liens suivants pour voir le résultat : http://www.april.org/cgi-bin/env et http://www.april.org/cgi-bin/env.pl

5 Debugger un script CGI

Vous venez de terminer la lecture de ce document, d'installer avec succès votre serveur HTTP, d'écrire un script CGI en vous basant sur les exemples fournis. Vous lancez votre navigateur pour faire un test.

D'ailleurs, à ce propos, utilisez bien `Ouvrir URL` pour accéder au script, et non pas `Ouvrir Fichier`. Si votre machine s'appelle `yoda`, et que vous avez placé le script `date.cgi` dans le répertoire des scripts CGI, vous devez ouvrir l'URL `http://yoda/cgi-bin/date.cgi`.

Et là, que voyez-vous apparaître à la place de la sortie attendue de votre programme ? Le navigateur vous affiche une erreur du genre `Internal server error`. Affreux.

Nous voilà donc lancé dans l'enfer du debug d'un script CGI. Disons tout de suite que le debug d'un script CGI n'est pas tout à fait le même problème que le debug d'un programme normal.

En effet, la plupart des programmes que vous avez écrits jusqu'à maintenant étaient lancés directement à partir de la ligne de commande. Ce n'est pas le cas des scripts CGI. Ils sont exécutés par une autre programme, le serveur HTTP, et souvent sur une machine différente de celle sur laquelle ils ont été écrits.

Si au lieu de la sortie attendue de votre programme, vous voyez s'afficher le code source du programme c'est que le serveur HTTP n'est pas configuré pour pouvoir exécuter des scripts CGI. Contacter l'administrateur du serveur, ou si c'est votre propre serveur relisez la partie 1.4 pour voir si vous l'avez correctement configuré.

Le secret du debug est donc de bien comprendre que **c'est le serveur HTTP qui exécute le programme**.

Gardons cela en tête, et commençons le debug de notre programme.

La première chose à vérifier est que le programme soit exécutable par tous dans le cas d'un programme compilé, et lisible et exécutable par tous dans le cas d'un script. Ceci parce que le serveur s'exécute sous un nom d'utilisateur différent du votre. Donc une fois votre programme installé dans le répertoire adéquat n'oubliez pas de faire un petit `chmod 555 date.cgi`, par exemple.

Si cela ne marche toujours pas, il faut essayer d'exécuter le programme sur la ligne de commande pour voir ce qu'il affiche en sortie. En effet, quand votre programme est exécuté par le serveur HTTP, le résultat de sa sortie est envoyé au navigateur, et donc vous ne voyez en fait que ce qu'en fait le navigateur. Votre programme peut donc s'exécuter correctement sur le serveur, afficher quelque chose sur sa sortie standard, mais cette sortie est peut-être incohérente pour le navigateur, qui vous affiche alors un message d'erreur.

Exécutez donc votre programme sur la ligne de commande, qu'affiche t'il en sortie ? Comme nous l'avons déjà vu précédemment, la première ligne affichée doit être le champ `Content-Type` du header HTTP, suivi par une ligne blanche. Si ce n'est pas le cas, corrigez-le. Une erreur fréquente est, en langage C, l'instruction suivante : `printf("Content-Type: text/html\n");`. Cette instruction est syntaxiquement correcte, mais elle n'affiche pas de ligne blanche après l'affichage de la chaîne. Le `\n` ne fait aller qu'à la ligne. Il faut deux `newline`. L'instruction correcte est la suivante : `printf("Content-Type: text/html\n\n");`. Ceci est une des erreurs les plus fréquement commises au début de la programmation de scripts CGI. Le reste de l'affichage, après le header et la ligne blanche, doit être cohérent avec le header. Donc, dans ce cas de figure, une sortie au format HTML. Par exemple:

```
Content-type: text/html

<HTML><HEAD>
<TITLE>Sortie Correcte</TITLE>
</HEAD>
<BODY>

<H1>Ceci est une sortie correcte</H1>

</BODY></HTML>
```

Ce test peut être suffisant si votre programme ne traite aucune donnée en entrée. Mais si votre programme traite des données, par exemple par l'intermédiaire d'un formulaire (voir le deuxième exemple), il faut tester le programme dans les conditions les plus proches du réel, et donc lui passer des arguments en entrée.

Si le formulaire utilise la méthode GET, votre programme récupère les données par l'intermédiaire de la variable d'environnement `QUERY_STRING`. On peut alors facilement affecter une valeur à cette variable sous le shell. Par exemple, en bash :

```
export QUERY_STRING="prenom=Marcel&nom=Gnou&age=41+%E0+60+ans"
```

Il faut affecter une valeur la plus proche possible du réel (donc éventuellement avec les `%xx` correspondant aux caractères accentués et réservés). On peut alors exécuter le programme sur la ligne de commande et étudier sa sortie.

Enfin, si votre script doit être installé sur une autre machine que celle sous laquelle vous l'avez écrit, pensez à vérifier les chemins (`path`) des programmes que vous utilisez. Dans l'exemple de `date.cgi`, le programme peut planter (et donc votre navigateur vous affiche un message d'erreur), si par exemple le programme `date` ne se trouve pas dans le répertoire `/bin`. Votre programme, donnant une sortie correcte sur votre machine, exécuté sur l'autre machine, affichera en sortie :

```
./date.cgi: /bin/date: command not found
Content-type: text/html

<HTML><HEAD><TITLE>Script Cgi</TITLE><HEAD>
<BODY>

<CENTER>

<H1>La date courante sur le serveur est</H1>


</CENTER>

</BODY>
</HTML>
```

Ce qui, vous en conviendrez, n'est pas une sortie correcte. Le problème peut se poser également avec les scripts écrits en Perl. En effet, dans nos exemples, on supposait que Perl se trouvait dans le répertoire `/usr/bin/` (première ligne de nos scripts : `#!/usr/bin/perl`). Si sur la machine sur laquelle doit s'exécuter notre programme perl ne se trouve pas dans ce répertoire nous aurions commme sortie :

```
./env.pl: No such file or directory
```

Ce qui n'est pas une sortie attendue par un navigateur.

6 That's all folks !

Ce document est juste une introduction aux script CGI, il ne se veut pas exhaustif.

Si vous avez remarqué des erreurs, si vous voulez apporter des corrections ou des rajouts, si vous avez des questions, n'hésitez pas à m'envoyer un petit mot à mad@bocal.cs.univ-paris8.fr.

Si vous désirez soutenir ou participer avec nous au mouvement du *logiciel libre*, n'hésitez pas à nous rejoindre sur APRIL, http://www.april.org.

7 Copyright