Techniques de Programmation Internet

Année Spéciale Informatique

ENSIMAG 2003-2004

James L. Crowley

Séance 5 3 mars 2004

Plan :

CGI : Common Gateway Interface.....2
Formulaires HTML 3
Exemple d'un Script pour un html dynamique.....4
Execution d'un script par une ficher HTML.....5
PERL 6
Exemple d'un script CGI appelé par href.....8
Exemple d'un script CGI appelé par un formulaire.....9
Passage des Paramètres aux SCRIPTs : GET et POST10
Méthode GET ou POST ?.....10
Passage des Paramètres avec GET.....11
Environnement du serveur 12
Exemple d'accès aux variables d'environnement 13
Codage des Paramètres.....14
Extraction des paramètres 15
Passage des Paramètres avec POST.....18
Accommodation de GET et POST.....19

CGI : Common Gateway Interface

Formulaires HTML

Rappel : Les formulaires sont des documents dynamiques.

Balise FORM

Syntaxe : <FORM ACTION ="url " METHOD=" méthode ">

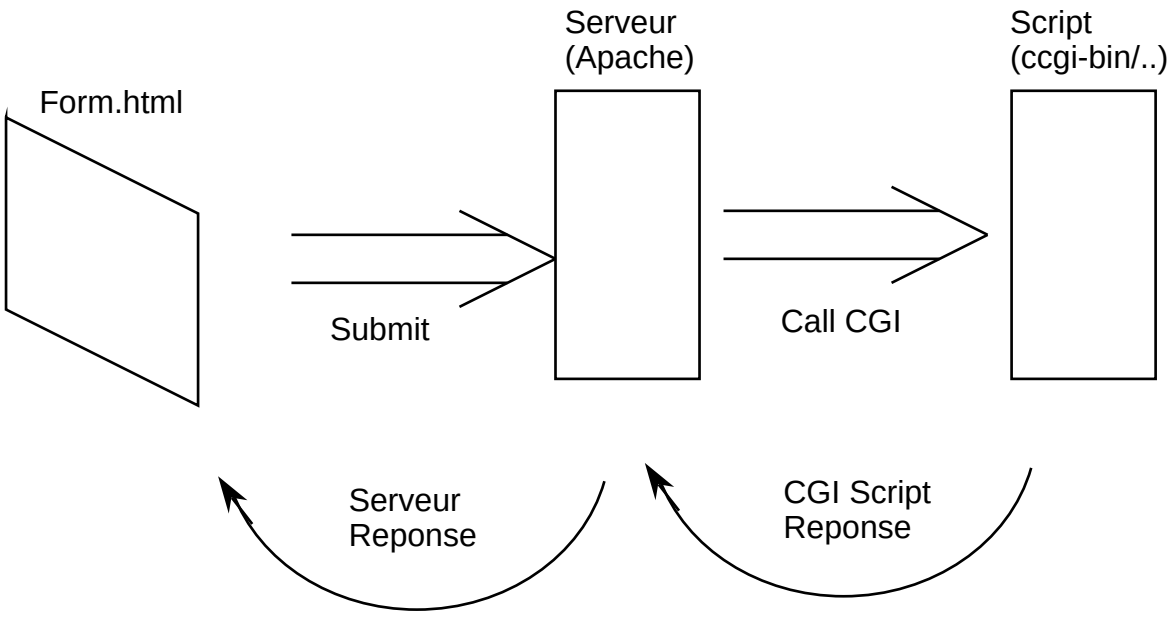
url identifie le programme utilisé pour traiter le formulaire.
méthode définit la méthode à employer pour transmettre au serveur l'information recueillie dans les champs du formulaire.

GET les données sont ajoutées à l'URL.
POST les données sont envoyées dans le corps du message.

Le formulaires sont interpretés par au travers le CGI : Common Gateway Interface.

Common : Utilisable dans tous les systèmes d'exploitation et avec tous les langages de programmation.
Gateway : CGI dans accès aux services
Interface : CGI est un protocole formellement spécifié.

Le "Common Gateway Interface" est un protocole qui permet au page html d'exécuter les programmes. Ceci autorise les possibilités illimitées aux pages html.



Execution des Scripts :

On peut exécuter un script à partir d'un page html via l'usage d'un formulaire.

Syntaxe de la balise FORM : <FORM ACTION ="script.cgi" METHOD="get">

script.cgi identifie le programme utilisé pour traiter le formulaire.
méthode définit la méthode à employer pour transmettre au serveur l'information recueillie dans les champs du formulaire.

Les scripts cgi sont installés dans un répertoire cgi-bin du serveur.

..[serveur]/cgi-bin/script.cgi

L'ENSIMAG a fait un lien "soft" dans le cgi-bin vers chaque membre du cours :

```
cgi-bin/jlc   ->  ~jlc/cgi-bin
```

Il faut que les programmes en cgi-bin soient executable par tout le monde!
Fait le commande :

```
cd cgi-bin        //   aller au repertoire
chmod a+x *       //   all have access executable
```

Exemple d'une communication d'un client vers un serveur :

```
GET /cgi-bin/script.cgi HTTP/1.0
Accept:  www/source
Accept:  text/html
Accept: image/gif
User-Agent: Lynx/2.4  libwww/2.14
From: jlc@ensisun.imag.fr
```

Une réponse de cgi pourrait être

```
HTTP/1.0  200 OK
Date:  Thursday, 24 February 2000  16:24:00  GMT
Server: Apache 1.24
Mime-version: 1.0
content-type: text/html
content-length: 200
```

Exemple d'un Script pour un html dynamique.

Un script peux créer une page dynamique .

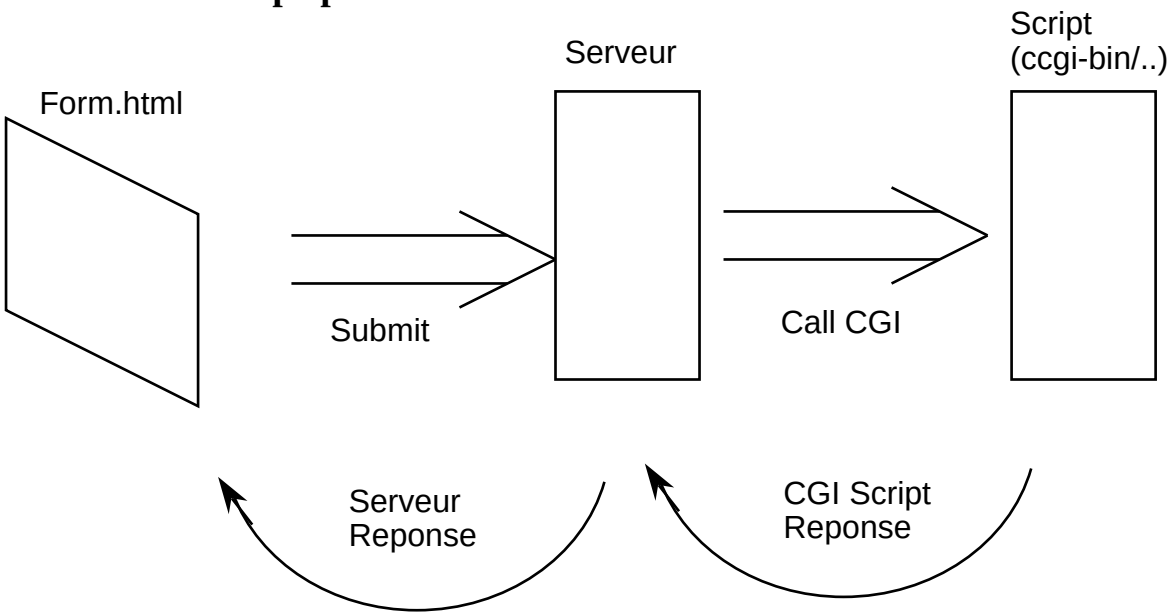
Considère le page HTML: (s5.0.html)

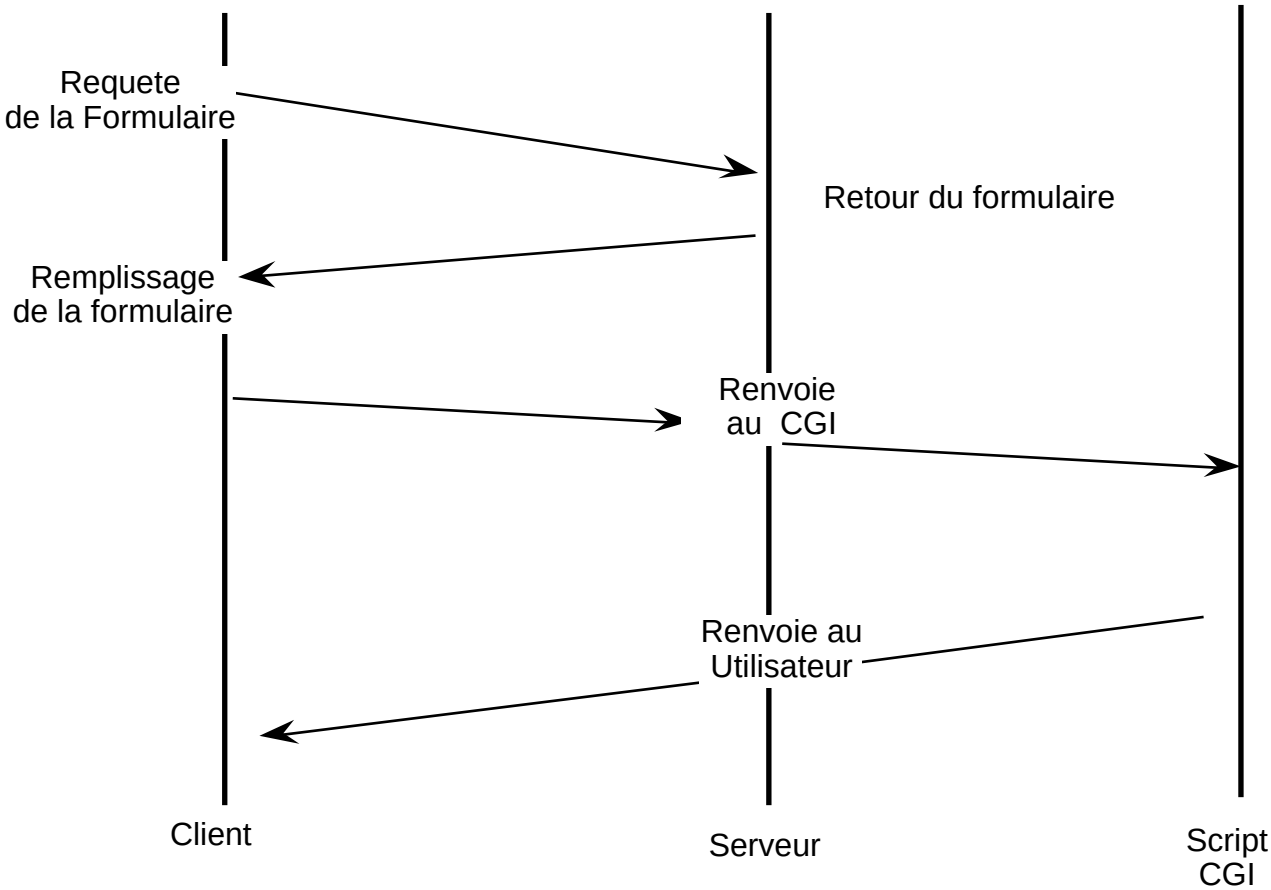
```
<HTML>
<TITLE>HTML Dynamique</TITLE>
<BODY   BGCOLOR=#FFFFFF >
<FORM ACTION ="/cgi-bin/s5.1.cgi">
<H1>Genere du html dynamic:</H1>
<INPUT TYPE=SUBMIT Value="Do it">
</FORM>
</BODY>
</HTML>
```

avec le script s5.1.cgi installé en cgi-bin.

```
#!/usr/local/bin/perl
# script name : s5.0.cgi
print "Content-type: text/html", "\n\n";
print "<HTML>" , "\n";
print  "<TITLE>Exemple d'une html dynamique</TITLE>" , "\n";
print "<BODY>" , "\n";
print "<H1>Who is logged in?</H1>" , "\n";
print `who`;
print "</BODY>" , "\n";
print "</HTML>" , "\n";
exit(0);
```

Execution d'un script par une ficher HTML





PERL

PERL : **P**ractical **E**xtraction and **R**eport **L**anguage

Un langage de script développé en 1986 par Larry Wall.
PERL est très populaire chez les ingénieurs système et Web.
PERL est un langage adapté pour manipuler nombres, textes, fichiers
et répertoires, réseau et programmes,

PERL est une langage "interprété". Sa syntaxe est inspirée de C, shell, awk, sed.
PERL dispose des opérateurs bien adaptés au traitement des requêtes de html.
Mais, on aura pu utiliser n'importe quel langage. Il suffit que la programme
soit accessible au serveur.

Le premier ligne doit être une "shebang" commande.
Le shell UNIX interpret cette ligne comme la specification de l'interpréter

Le commande print envoie un texte au STDOUT.
Pour un scripte CGI, STDOUT est dirigé vers l'interpréteur HTML.

Types de Variables en PERL

$variable	scalar - un valeur numérique ou une chaîne
@variable	array - une liste de valeurs indexé par les entiers
%variable	hash - Une liste de valeurs indexé par les clefs symboliques
&variable	subroutine - Une scripte en perl
*variable	typeglob - un type de donnée déclaré

exemples :

Scalars

```
$number = 10;
```

Arrays:

```
@table = ("first", "second", "third");
$table[0] = "first"
```

Hashes

```
%weekdays = ("sun" => "dimanche", "mon" => "lundi", "tue" => "mardi")
ou bien
$weekdays("wed") = "Mercredi";
```

Les chaînes entourées des marques de quotation " " sont interprétés

Si on les entoure des "single-quotes" `` ou '' ils sont conservées..

Par exemple :

```
$variable = "value";
print "Value is $variable", "\n";
print 'Variable is $variable',  "\n";
```

Execution:

```
Value is value
Variable is $variable
```

Exemple d'un script CGI appelé par href

Soit le script s5.1.cgi installé en cgi-bin.

```
#!/usr/local/bin/perl
#file s5.1.cgi
print "Content-type: text/html", "\n\n";
print "<HTML>" , "\n";
print  "<TITLE>Current Users</TITLE>" , "\n";
print "<BODY>" , "\n";

print `who`;   # le commande "who" est executee

print "</BODY>" , "\n";
print "</HTML>" , "\n";
exit(0);
```

la ligne "print `who`;" fait appelle à la commande Unix, "who".
La sortie de "who" au STDOUT.
L'html du formulaire s5.0.html:

```
<HTML>
<TITLE>Who is logged in? </TITLE>
<BODY   BGCOLOR=#FFFFFF >
<H1>Who is logged on?</H1>
<A HREF="/cgi-bin/jlc/s5.1.cgi"> Who</A> is logged on
</FORM>
</BODY>
</HTML>
```

On peut utiliser "href" ou "form" pour appeler le script.
La différence est dans le passage de paramètres.

Les paramètres sont codés :?arg=value&arg=value
Exemple :
/cgi-bin/jlc/s5.1.cgi?arg=value&arg=value

Avec des formulaires, la chaîne de caractères des paramètres sont composées automatiquement. S'on utilise "href" , il faut construire une chaîne de caractères codés pour les paramètres.

Exemple d'un script CGI appelé par un formulaire

L'html du formulaire s5.1.html

```
<HTML>
<TITLE>Qui est sur la systeme ?</TITLE>
<BODY   BGCOLOR=#FFFFFF >
<FORM ACTION ="/cgi-bin/jlc/s5.1.cgi" METHOD="get">
<H1>Who is logged on?</H1>
<INPUT TYPE=SUBMIT Value="Who">
</FORM>
</BODY>
</HTML>
```

Note le "?" a la fin de l'appel.
http://ensisun.imag.fr/cgi-bin/jlc/s5.1.cgi?

Passage des Paramètres aux SCRIPTs : GET et POST

Il y a deux méthodes de passage des paramètres aux scripts :

GET les données sont transmises par avec la requête à l'URL.

POST les données sont envoyées et lues via STDIN.

Méthode GET ou POST ?

GET
- Avantage : un questionnaire peut être conservé sous forme d'une URL, http://ensisun.imag.fr/cgi-bin/a?name=Crowley&prenom=Jim
- Inconvénient : la chaîne de caractères a une taille limitée.

POST
- Avantage : la taille de la requête n'est pas restreinte par la taille de URL
- Inconvénient : impossibilité de garder la requête autrement que sous un fichier.

Passage des Paramètres avec GET

```
<HTML>
<TITLE>Exemple d'une Forme</TITLE>
<BODY BGCOLOR=#FFFFFF >
<FORM ACTION ="/cgi-bin/s5.2.cgi" METHOD="get">
<H1>Veuillez completer ce formulaire</H1>
<P>Nom: <input name="name" size="48">
<br>
<INPUT TYPE=SUBMIT Value="Valider">
<INPUT TYPE=RESET Value="Reinitialiser">
</FORM>
</BODY>
</HTML>
```

avec le script s5.2.cgi

```
#!/usr/local/bin/perl
#file name s5.2.cgi
print "Content-type: text/html", "\n\n";
print "<HTML>" , "\n";
print  "<TITLE>s5.2.cgi</TITLE>" , "\n";
print "<BODY>" , "\n";
$query_string = $ENV{'QUERY_STRING'};
print "Script called with $query_string", "\n";
print "</BODY>" , "\n";
print "</HTML>" , "\n";
```

exit(0);

La requête est :
http://ensisun.imag.fr/cgi-bin/s5.2.cgi?name=Jim+Crowley

La chaîne "name=Jim+Crowley" est retourné par la variable d'environnement "'QUERY_STRING';

La variable de l'environnement QUERY-STRING contient une chaîne avec tous les paramètres, avec la forme :

cle=value.
Value est limité à la lettre et nombre.

Les variables d'environnement sont accessibles via $ENV{...}

Environnement du serveur

Les programmes du CGI disposent d'un nombre important de variable d'environnement .

DOCUMENT_ROOT	Répertoire contenant le serveur
GATEWAY_INTERFACE	Version CGI supportée par le serveur
HTTP_HOST	Adresse IP de la machine du serveur
SCRIPT_NAME	URL du chemin du script
SERVER_ADMIN	Adresse électronique de l'administrateur du serveur
SERVER_NAME	Nom (complet ou IP) du serveur gérant les scripts CGI
SERVER_PORT	Numéro de port réceptionnant les requêtes (normalement 80)
SERVER_PROTOCOL	Protocole et version du serveur (HTTP 1.1)
SERVER_SOFTWARE	Nom et version du serveur
HTTP_ACCEPT	Types MIME supportés par le client
HTTP_COOKIE	Propriétés associées par le client à la ressource consultée
HTTP_USER_AGENT	Informations sur le client (cf. User-agent dans les requêtes)
REMOTE_ADDR	Adresse IP de la machine cliente
REMOTE_HOST	Adresse symbolique de la machine cliente
REMOTE_IDENT	Login de la personne effectuant la requête (pas toujours disponible)
REMOTE_USER	Identificateur du client dans un mode d'authentification
REQUEST_METHOD	Méthode de la requête (GET | HEAD | PUT | POST | DELETE | LINK)
AUTH_TYPE	Contient le mode d'authentification si une authentification est requise sur l'URL
CONTENT_LENGTH	Taille en octets des informations jointes à la requête
CONTENT_TYPE	Type MIME de la requête (formulaire HTTP: application/x-www-form-urlencoded)
HTTP_CONNECTION	Type de connexion établie par le client (le plus souvent Keep-Alive , le client attend la réponse)
PATH_INFO	Chemin du script
PATH_TRANSLATED	Chemin absolu du script CGI (avec le format protocole://NomServeur:port/path)
QUERY_STRING	Chaîne de caractères caractérisant la requête

Exemple d'accès aux variables d'environnement

```
#!/usr/local/bin/perl
# file s5.3.cgi
print "Content-Type: text/html", "\n\n";
print "<HTML>", "\n";
print "<HEAD> <TITLE>Environnement du serveur</TITLE>
</HEAD>", "\n";
print "<HR><PRE>", "\n";
print "Server Name:   ", $ENV{'SERVER_NAME'}, "<BR>", "\n";
print "Running on port: ", $ENV{'SERVER_PORT'}, "<BR>",
"\n";
print "Server Software",$ENV{'SERVER_SOFTWARE'},  "<BR>",
"\n";
print "Server Protocol",$ENV{'SERVER_PROTOCOL'}, "<BR>",
"\n";
print "Adresse IP de la machine du serveur: ",
$ENV{'HTTP_HOST'}, "<BR>", "\n";
print "Repertoire contenant le serveur",
$ENV{'DOCUMENT_ROOT'}, "<BR>", "\n";
print "Adresse IP de la machine cliente",
$ENV{'REMOTE_ADDR'}, "<BR>", "\n";
print "Adresse symbolique de la machine cliente: ",
$ENV{'REMOTE_HOST'},"<BR>","\n";
print "Login de la personne effectuant la
requete:",$ENV{'REMOTE_IDENT'},"<BR>", "\n";
print "</BODY>" , "\n";
print "</HTML>" , "\n";
exit(0);
```

Avec le page html : s5.3.html

```
<HTML>
<TITLE>Echo les Variables d'Environement</TITLE>
<BODY BGCOLOR=#FFFFFF >
<H1>Les Variables de l'Environement</H1>
<p>
Les <A HREF="/cgi-bin/s5.3.cgi">Variables</A>
de l'environement.
<br>
</FORM>
</BODY>
</HTML>
```

Codage des Paramètres

1. Les caractères non ASCII (code > 128) sont remplacés par %xx ou xx est le code ASCII.
2. Les caractères réservés sont remplacés par leur code ASCII. Ils sont :
   < > " # % ! $ \ & ( ) + = } [ ] \ : ; ~ ? , / [TAB]
3. L'espace est remplacé par le signe +
4. Les pairs noms/valeur sont transformés par la chaîne de caractères : nom=valeur
5. Les différentes chaînes de caractères sont contaminées en insérant le symbole & entre les paires : nom1=valeur1&nom2=valeur2&...

Voici un script PERL pour convertir une chaîne a une chaîne codée pour CGI:

```
#!/usr/local/bin/perl
print "String to encode: ";
$string = <STDIN>;
chop($string);

string =~ s/(\W)/sprintf("%%%x", ord($1))/eg;

print "Encoded String: ";
print $string, "\n";
exit(0);
```

On peut décoder les lignes avec une substitution en PERL.
La syntaxe d'une substitution est tr/vieux/nouvelle/

```
#!/usr/local/bin/perl
print "String to decode: ";
$string = <STDIN>;
chop($string);
$string =~ tr/+/ /;
$string =~  s/%([\dA-Fa-f][\dA-Fa-f])/pack("C", hex($1))/eg;

print "Decoded String: ";
print $string, "\n";
exit(0);
```

Extraction des paramètres

En PERL, on peut convertir les chaînes de la forme nom=valeur dans une "hash" ou "nom" est la clé et "valeur" est la valeur avec une commande "split:". Syntaxe : (clé, valeur) = split(/char/, $chaine);

Par exemple :

```
$query_string = $ENV{'QUERY_STRING'};
($field_name, $command) = split(/=/, $query_string);
```

```
#!/usr/local/bin/perl
#file s5.4.cgi
print "Content-type: text/html", "\n\n";
print "<HTML>" , "\n";
print  "<TITLE>Execution d'un commande unix</TITLE>" , "\n";
print "<BODY>" , "\n";
$query_string = $ENV{'QUERY_STRING'};
print "query_string : $query_string.<BR>", "\n";
($field, $value) = split(/=/, $query_string);
print "field = $field.<br>value=  $value. <br>", "\n";
print "<br>", "\n";
print "<br>", "\n";
print `$value`;
print "</BODY>" , "\n";
print "</HTML>" , "\n";
exit(0);
```

avec s5.4.html

```
<HTML>
<TITLE>Commandes UNIX</TITLE>
<BODYBGCOLOR=#FFFFFF >
<FORM ACTION ="/cgi-bin/jlc/s5.4.cgi" METHOD="get">
<H1>Executer un commande UNIX</H1>
<br>
<P>Commande: <input name="commande" size="48">
<br>
<INPUT TYPE=SUBMIT Value="Valider">
<INPUT TYPE=RESET Value="Reinitialiser">
</FORM>
</BODY>
</HTML>
```

S'il y a des arguments, il faut aussi les décoder avec

```
$form_info =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C", hex($1))/eg;
```

Exemple :
s5.5.html : Exécuter une commande à partir d'un formulaire.

```
<HTML>
<TITLE>Commandes UNIX</TITLE>
<BODYBGCOLOR=#FFFFFF >
<FORM ACTION ="/cgi-bin/s5.5.cgi" METHOD="get">
<H1>Executer un commande UNIX</H1>
<P>Commande: <input name="commande" size="48">
<br>
<INPUT TYPE=SUBMIT Value="Valider">
<INPUT TYPE=RESET Value="Reinitialiser">
</FORM>
</BODY>
</HTML>
```

```
s5.5.cgi :
#!/usr/local/bin/perl
# File name s5.5.cgi
print "Content-type: text/html", "\n\n";
print "<HTML>" , "\n";
print  "<TITLE>Execution d'un commande unix</TITLE>" , "\n";
print "<BODY>" , "\n";
#get the query string and decode it
$query_string = $ENV{'QUERY_STRING'};
print "query = $query_string<br>", "\n";
$query_string =~ tr/+/ /;   #substitute " " for "+"
$query_string =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C",
hex($1))/eg;
print "query = $query_string<br>", "\n";
#split the query string at "="
($field_name, $command) = split(/=/, $query_string);
print "Execution du commande :  $command", "\n";
print "<br>", "\n";
print "<br>", "\n";
print `$command`;
print "</BODY>" , "\n";
print "</HTML>" , "\n";
exit(0);
```

L'exemple suivant permet d'exécuter une logicielle ou commande Unix
a partir d'un page web:

Attn. Ceci peut être TRES dangereux. L'utilisateur peut taper "rm -r *"
ou bien n'importe quel autre commande.

Passage des Paramètres avec POST

Avec POST, il faut lire la réponse par STDIN.
En PERL, ceci se fait par :

```
$size = $ENV{'CONTENT_LENGTH'};
read (STDIN, $query_string, $size};
```

exemple : s5.6.html

```
<HTML>
<TITLE>Exemple d'une Forme</TITLE>
<BODY BGCOLOR=#FFFFFF >
<H1>Veuillez completer ce formulaire</H1>
<FORM ACTION ="/cgi-bin/s5.6.cgi" METHOD="POST">
<P>Nom: <input name="name" size="48">
<br>
<INPUT TYPE=SUBMIT Value="Valider">
<INPUT TYPE=RESET Value="Reinitialiser">
</FORM>
</BODY>
</HTML>
```

s5.6.cgi

```
#!/usr/local/bin/perl
print "Content-type: text/html", "\n\n";
print "<HTML>" , "\n";
print  "<TITLE>s5.6.cgi</TITLE>" , "\n";
print "<BODY>" , "\n";
$size = $ENV{'CONTENT_LENGTH'};
read (STDIN, $query_string, $size);
print "Script called with $query_string <p>", "\n";
print "Size of query_string is  $size.", "\n";
print "query string is  $query_string<br>", "\n";

$query_string =~ tr/+/ /;   #substitute " " for "+"
$query_string =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C",
hex($1))/eg;

print "decode of query :  $query_string<br>", "\n";
#split the query string at "="
($field_name, $command) = split(/=/, $query_string);
```

```
print "Execution du commande :  $command", "\n";
print "<br>", "\n";
print "<br>", "\n";

print `$command`;

print "</BODY>" , "\n";
print "</HTML>" , "\n";
exit(0);
```

Accommodation de GET et POST.

La choix de méthode (GET ou POST) est disponible dans la variable d'environnement : "REQUEST_METHOD".
On peut écrire des scripts qui acceptent les deux méthodes GET et POST.

```
$method = $ENV{'REQUEST_METHOD'};
if ($method eq "GET")
{
   $query = $ENV{'QUERY_STRING'};
} elseif  ($method eq "POST")
{
   read (STDIN, $query, $ENV{'CONTENT_LENGTH');
} else
{
   &return_error(500, "Server Error", "Unsupported
Method");
}
```

Exemple :

```
#!/usr/local/bin/perl
#file s5.7.cgi
print "Content-type: text/html", "\n\n";
print "<HTML>" , "\n";
print  "<TITLE>s5.7.cgi</TITLE>" , "\n";
print "<BODY>" , "\n";
$method = $ENV{'REQUEST_METHOD'};
if ($method eq "GET")
{
    $query = $ENV{'QUERY_STRING'};
} elsif  ($method eq "POST")
{
    $size = $ENV{'CONTENT_LENGTH'};
    read (STDIN, $query_string, $size);
} else
{
    &return_error(500, "Server Error", "Unsupported
Method");
}
print "Script called with query_string: $query", "\n";
print "</BODY>" , "\n";
print "</HTML>" , "\n";
exit(0);
```

exemple : s5.7.html

```
<HTML>
<TITLE>Exemple d'une Forme</TITLE>
<BODY BGCOLOR=#FFFFFF >
<H1>Veuillez completer ce formulaire</H1>
<FORM ACTION ="/cgi-bin/jlc/s5.7.cgi" METHOD="GET">
<P>Nom: <input name="name" size="48">
<br>
<INPUT TYPE=SUBMIT Value="Valider">
<INPUT TYPE=RESET Value="Reinitialiser">
</FORM>
</BODY>
</HTML>
```