Technologie Web
CGI

Alexandre Pauchet

INSA Rouen - Département ASI

BO.B.RC.18, pauchet@insa-rouen.fr

Plan

Introduction (1/3)
Description

Passerelle entre l'utilisateur et le serveur WEB

- un programme exécuté **côté serveur**
- peut être exécuté à la demande explicite de l'utilisateur
- permet de produire dynamiquement un document
  - dynamique côté serveur
  - statique côté client
- prend en entrée les informations envoyées par le navigateur (type, version, données d'un formulaire, ...)
- donne en sortie un document pouvant être traité par le navigateur, sauvegardé ou encore ouvert par une application sur le client

Introduction (2/3)
Scripts CGI

Choix du langage

- Selon le serveur
  - disponibilité des langages
  - le type d'OS
  - puissance
  - ...
- Selon les fonctionnalités nécessaires pour le traitement
  - accès BD
  - création de graphiques
  - manipulation importante de texte
  - ...

Introduction (3/3)
Langages courants

PERL

- interprété
- puissant dans le traitement des chaînes de caractères
- nombreuses librairies (BD, CGI, ...)

Shell

- interprété
- nombreux outils externes (awk, sed, uncgi, ...)

C, C++

- compilé
- puissant et rapide
- nombreuses librairies disponibles

Principes (1/5)
Requête/Réponse HTTP (rappel)

Exemple

Requête

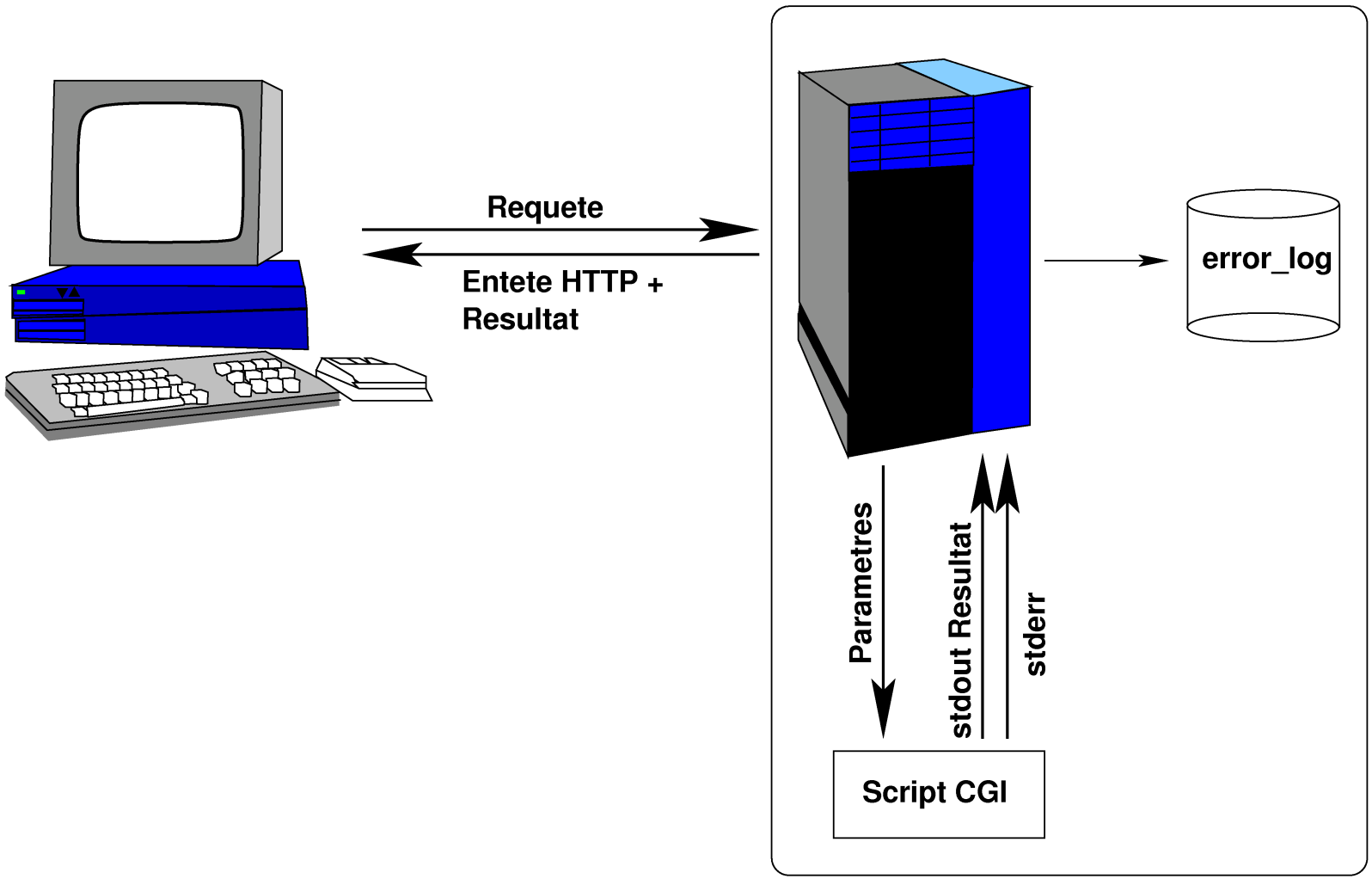
```
GET /TW/CM02-xhtml/Hello.xhtml HTTP/1.1
Host: localhost
```

Réponse

```
HTTP/1.1 200 OK
...
Content-Type: application/html+xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
      strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title>Page XHTML Type</title>
  <meta http-equiv="content-type" content="text/html;charset=utf-8" />
  <meta http-equiv="Content-Style-Type" content="text/css" />
</head>
<body>
  <p>Hello world!</p>
</body>
</html>
```

Principes (2/5)
Fonctionnement



Principes (3/5)
Fonctionnement

Description du fonctionnement

1. Le client effectue une requête HTTP
2. Le serveur détecte la demande d'un CGI
3. Exécution du CGI dans un environnement spécifique, le CGI peut recevoir des paramètres d'entrée de deux manières :
   - soit la variable QUERY_STRING
   - soit l'entrée standard

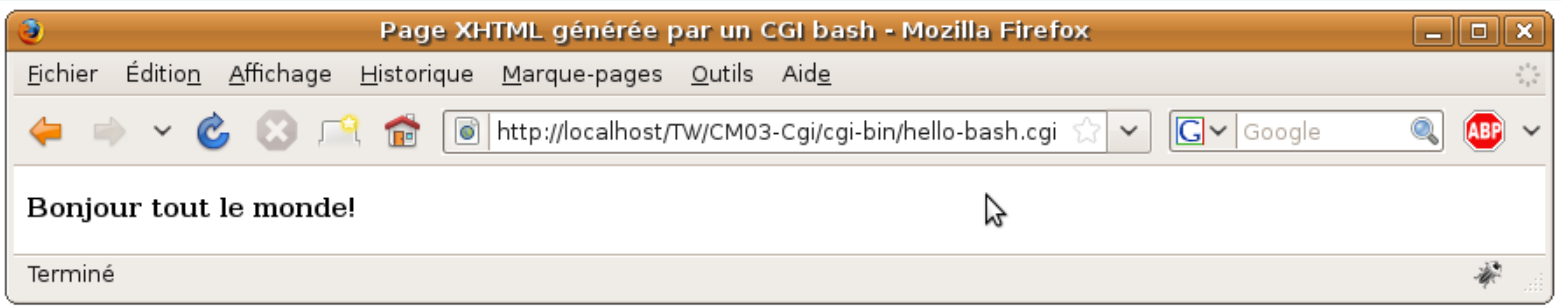
   De plus, des variables renseignent le CGI sur le client et l'environnement d'exécution
4. Le CGI envoie un document sur le flot de sortie standard
   le flot d'erreur est connecté au fichier de log d'erreurs
5. Le serveur lui ajoute un entête HTTP et le retourne au client

Principes (4/5)
Mon premier CGI en BASH

hello-bash.cgi

```
#!/bin/bash

echo "Content-type: text/xml"
echo
echo "<?xml version='1.0' encoding='UTF-8'?>"
echo "<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Strict//FR' 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-
      strict.dtd'>"
echo "<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='fr' lang='fr'>"
echo "<head>"
echo "  <title>Page XHTML générée par un CGI bash</title>"
echo "  <meta http-equiv='content-type' content='text/html;charset=utf-8' />"
echo "</head>"
echo "<body>"
echo "  <p><strong>Bonjour tout le monde!</strong></p>"
echo "</body>"
echo "</html>"
```



Principes (5/5)
Requête HTTP

Exemple

Requête

```
GET /TW/CM03-Cgi/cgi-bin/hello-bash.cgi HTTP/1.1
Host: localhost
```

Réponse

```
HTTP/1.1 200 OK
Date: Tue, 28 Jul 2009 13:17:34 GMT
Server: Apache/2.2.11 (Ubuntu) PHP/5.2.6-3ubuntu4.1 with Suhosin-Patch
Vary: Accept-Encoding
Content-Length: 433
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Strict//FR' 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-
      strict.dtd'>
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='fr' lang='fr'>
<head>
  <title>Page XHTML générée par un CGI bash</title>
  <meta http-equiv='content-type' content='text/html;charset=utf-8' />
</head>
<body>
  <p><strong>Bonjour tout le monde!</strong></p>
</body>
</html>
```

Quelques exemples (1/5)
Mon premier CGI en Perl

hello-perl.cgi

```
#!/usr/bin/perl

print "Content-type: text/xml\n\n";
print "<?xml version='1.0' encoding='UTF-8'?>\n";
print "<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Strict//FR' 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-
      strict.dtd'>\n";
print "<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='fr' lang='fr'>\n";
print "<head>\n";
print " <title>Page générée par un script CGI en Perl</title>\n";
print " <meta http-equiv='content-type' content='text/html;charset=utf-8' />\n";
print "</head>\n";
print "<body>\n";
print " <p><i>Bonjour tout le monde !</i></p>\n";
print "</body>\n";
print "</html>\n";
```

Quelques exemples (2/5)
Mon premier CGI en C++

hello-C++.cpp

```
#include <iostream>

using namespace std;

int main()
{
  cout << "Content-type: text/xml\n\n";
  cout << "<?xml version='1.0' encoding='UTF-8'?>\n";
  cout << "<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Strict//FR' 'http://www.w3.org/TR/xhtml1/DTD/
       xhtml1-strict.dtd'>\n";
  cout << "<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='fr' lang='fr'>\n";
  cout << "<head>\n";
  cout << "     <title>Page générée par un script CGI en C++</title>\n";
  cout << " <meta http-equiv='content-type' content='text/html;charset=utf-8' />\n";
  cout << "</head>\n";
  cout << "<body>\n";
  cout << "    <p><b><i>Bonjour tout le monde !</i></b></p>\n";
  cout << "</body>\n";
  cout << "</html>\n";
  return 0;
}
```

Quelques exemples (3/5)
Les variables d'environnement

Les *variables d'environnement CGI* sont des variables transmises à un programme CGI, par le **serveur Web** l'invoquant, lors de son exécution.
Elles fournissent des informations sur la requête du client, le serveur, le client.
Liste (non exhaustive) des variables

- Variables en rapport avec le serveur
  - SERVER_SOFTWARE : nom et version du serveur HTTP
  - SERVER_NAME : nom d'hôte, alias DNS ou adresse IP du serveur.
- Variables spécifiques à la requête
  - REQUEST_METHOD : méthode utilisée pour faire la requête. Pour HTTP, elle contient généralement "GET" ou "POST".
  - QUERY_STRING : tout ce qui suit le " ?" dans l'URL envoyée par le client (*i.e.* les variables provenant d'un formulaire envoyé avec la méthode "GET").
  - REMOTE_ADDR : adresse IP du client.
  - REMOTE_USER : nom d'utilisateur du client, si le script est protégé et si le serveur supporte l'identification.
- Variables provenant du client
  - HTTP_ACCEPT : types de données MIME acceptés par le client.
  - HTTP_ACCEPT_LANGUAGE : langages acceptés par le client.

Quelques exemples (4/5)
CGI affichant les variables d'environnement

variables.cgi

```
#!/bin/bash

echo Content-Type: text/plain
echo

export | sed -e"s/declare -x //g"
```

Quelques exemples (5/5)
Les variables d'environnement

```
DOCUMENT_ROOT="/var/www"
GATEWAY_INTERFACE="CGI/1.1"
HTTP_ACCEPT="text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
HTTP_ACCEPT_CHARSET="ISO-8859-1,utf-8;q=0.7,*;q=0.7"
HTTP_ACCEPT_ENCODING="gzip,deflate"
HTTP_ACCEPT_LANGUAGE="fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3"
HTTP_CONNECTION="keep-alive"
HTTP_HOST="localhost"
HTTP_KEEP_ALIVE="300"
HTTP_REFERER="http://localhost/TW/CM03-Cgi/cgi-bin/"
HTTP_USER_AGENT="Mozilla/5.0 (X11; U; Linux i686; fr; rv:1.9.0.12) Gecko/2009070811 Ubuntu/9.04 (jaunty
      ) Firefox/3.0.12"
OLDPWD
PATH="/usr/local/bin:/usr/bin:/bin"
PWD="/home/apauchet/Enseignements/TechnoWeb/Cours/CGI/Exemples/cgi-bin"
QUERY_STRING=""
REMOTE_ADDR="127.0.0.1"
REMOTE_PORT="46054"
REQUEST_METHOD="GET"
REQUEST_URI="/TW/CM03-Cgi/cgi-bin/variables.cgi"
SCRIPT_FILENAME="/home/apauchet/Enseignements/TechnoWeb/WWW-ApacheHttp/CM03-Cgi/cgi-bin/variables.cgi"
SCRIPT_NAME="/TW/CM03-Cgi/cgi-bin/variables.cgi"
SERVER_ADDR="127.0.0.1"
SERVER_ADMIN="Alexandre.Pauchet@insa-rouen.fr"
SERVER_NAME="localhost"
SERVER_PORT="80"
SERVER_PROTOCOL="HTTP/1.1"
SERVER_SIGNATURE="<address>Apache/2.2.11 (Ubuntu) PHP/5.2.6-3ubuntu4.1 with Suhosin-Patch Server at
      localhost Port 80</address>"
SERVER_SOFTWARE="Apache/2.2.11 (Ubuntu) PHP/5.2.6-3ubuntu4.1 with Suhosin-Patch"
SHLVL="1"
```

Appel d'un CGI (1/6)
Principe d'appel

- Les scripts CGI peuvent être appelés où on peut mettre une URL
  - `<a href="...">` lors d'un clic sur un lien
  - `<img src="...">` lors du chargement d'une image
  - `<form action="...">` lors de l'envoi d'un formulaire

- Selon la méthode d'envoi (GET ou POST), les paramètres envoyés à un CGI sont disponibles :

  `GET` : dans la variable d'environnement `QUERY_STRING`
  - `<a href="...">`
  - `<img src="...">`
  - `<form action="..." method="GET">`

  `POST` : sur l'entrée standard du CGI
  - `<form action="..." method="POST">`

Appel d'un CGI (2/6)
Exemple de passage de paramètres

Exemple

Méthode `GET`

Paramètres :

prenom=Alexandre nom=Pauchet sexe=masculin

URL :

```
http://localhost/TW/CM03-Cgi/cgi-bin/formulaire.cgi?prenom=
    Alexandre&nom=Pauchet&sexe=masculin&action=Envoyer
```

Résultat :

```
QUERY_STRING="prenom=Alexandre&nom=Pauchet&sexe=masculin&action=Envoyer"
```

Appel d'un CGI (3/6)
Le codage des paramètres

Les paramètres sont "url-encodés"

- les paramètres sont séparés par le caractère '&'
- les espaces sont remplacés par des '+'
- les caractères spéciaux sont remplacés par leur code hexa %xx

⇒ Lors de l'utilisation d'un formulaire les données envoyées sont automatiquement url-encodées, le CGI doit les décoder

Exemple

`caract%E8res+sp%E9ciaux` ⇒ `caractères spéciaux`

Appel d'un CGI (4/6)

Exemple de formulaire appelant un CGI en GET

formulaire-get.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//FR" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
      strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
  <head>
    <title>Exemple Formulaire + CGI</title>
      <meta http-equiv="content-type" content="text/html;charset=utf-8" />
  </head>
  <body>
    <form action="/TW/cgi-bin/formulaire-get.cgi" method="GET">
      <label>Prénom</label> <input type="text" name="prenom" size="10"/>
      <label>Nom</label>    <input type="text" name="nom" size="20"/><br/>
      <label>Femme</label> <input type="radio" name="sexe" value="feminin"/>
      <label>Homme</label> <input type="radio" name="sexe" value="masculin"/><br/>
      <input type="submit" name="action" value="Envoyer"/>
      <input type="reset" value="Effacer"/>
    </form>
  </body>
</html>
```

Appel d'un CGI (5/6)
Le CGI appelé par le formulaire en GET

formulaire-get.cgi

```
#!/usr/bin/perl

print "Content-type: text/plain\n\n";

$requete = $ENV{'QUERY_STRING'};
print "requete=$requete\n";
@paires = split(/&/, $requete);
foreach $element_valeur (@paires) {
  ($element, $valeur) = split(/=/, $element_valeur);
  $valeur =~ tr/+/ /;
  print "$element = $valeur\n";
}
```

Affichage :

```
requete=prenom=Bob&nom=Inette&sexe=feminin&action=Envoyer
prenom = Bob
nom = Inette
sexe = feminin
action = Envoyer
```

Appel d'un CGI (6/6)
Quelques pièges à éviter

Remarque

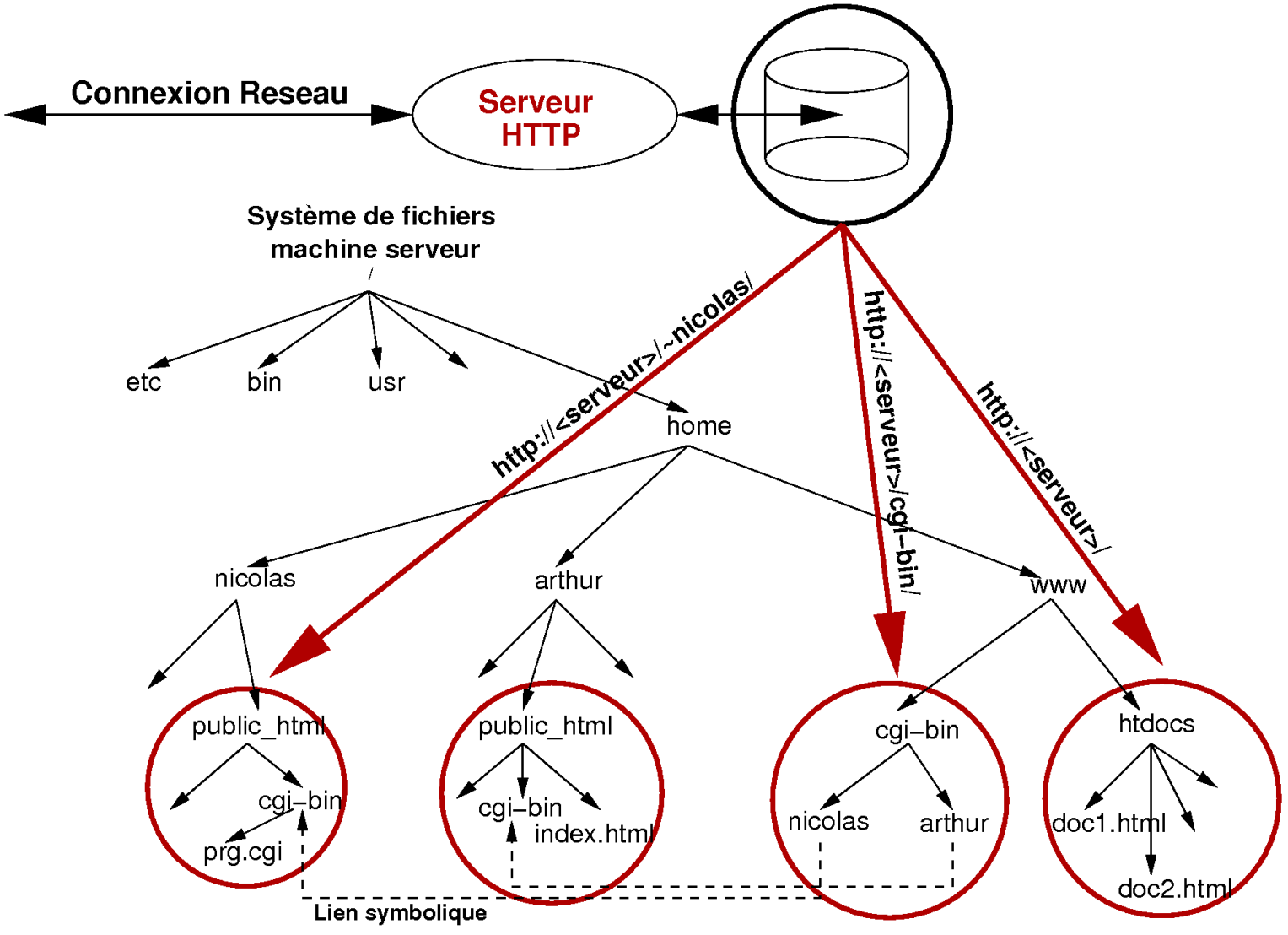
- Gérer les incohérences que l'utilisateur peut saisir dans les formulaires
- Ne pas commencer à envoyer un début de page, si on n'est pas sûr du résultat
- Envoyer le `Content-Type`
- Ne pas oublier la ligne vierge du protocole HTTP
- Envoyer le bon type
- Penser à protéger les ressources communes (accès concurrent des CGI)
- Rendre exécutable le CGI

Sécurité (1/3)
Configuration du serveur pour les CGI

Configuration sécurisée

- pas de configuration basée (uniquement) sur l'extension
- localiser tous les CGI dans un même espace
- possibilité de désactiver rapidement les CGI d'un compte
- exécution des scripts
  - soit par `nobody`
  - soit par l'utilisateur avec ses droits : `suEXEC`

Sécurité (2/3)
Les CGI dans le système de fichiers



Sécurité (3/3)
Sécuriser les CGI

Quelques trous de sécurité et leur solution :

- **injection de commandes**
  ⇒ vérifier la présence de caractères suspects dans les paramètres (`..&|;‘$`)
- **buffer overflow**
  ⇒ limiter la taille des paramètres envoyés, mais aussi paramétrer le serveur avec `LimitRequestLine`, `LimitRequestBody`
- **denial of service (DOS)**
  ⇒ limiter les ressources allouées en paramétrant le serveur avec `RLimitCPU`, `RLimitMEM`, `RLimitNPROC`
- **script infini**
  ⇒ timeout dans le script directement (en C avec `alarm`)
- **accès a des fichiers interdits** (config ou système)
  ⇒ protéger l'accès aux répertoires, interdire de remonter l'arborescence, accéder aux commandes par leur chemin absolu (/bin/ls)