Web technologies

- **Forms / CGI (common gateway interface)**
  - extract info from a form, create a page, send it back
  - server side code in Perl, PHP, ASP, JSP, ...
  - client side uses CSS for consistency of appearance
    (CSS = cascading style sheets)
- **Javascript**
  - client-side interpreter
- **DOM (document object model)**
  - controlling page properties by program (e.g., Javascript)
- **Ajax (asynchronous Javascript and XML)**
  - asynchronous update of page contents, e.g., Google Map
- **Libraries and APIs**
  - bodies of useful client-side code
    e.g., Prototype, Scriptaculous, Yahoo UI library, Dojo, ...
- **Frameworks**
  - integrated systems for creating web applications
    Ruby on Rails
    Google Web Toolkit (Java->Javascript)
    Django(Python)
    and zillions of others
  - often linked to a database like MySQL or SQLite
- **mashups**
  - combining data from multiple web sources into single application

Web

- **basic components**
  - URL (uniform resource locator)
  - HTTP (hypertext transfer protocol)
  - HTML (hypertext markup language)
  - browser
- **embellishments in browser**
  - helpers or plug-ins to display non-text content
    pictures (e.g., GIF, JPEG), sound, movies, ...
  - forms filled in by user
    client encodes form information in URL or on stdout
    server retrieves it from environment or stdin
    usually with cgi-bin program
    can be written in anything: Perl, PHP, shell, C, Java, ...
  - active content: download code to run on client
    Javascript
    Java applets
    plug-ins (Flash, Quicktime, RealPlayer, Firefox extensions, ...)
    ActiveX
- **HTTP is stateless**
  - server doesn't save anything from one request to next
  - cookies: a way to remember information on the client

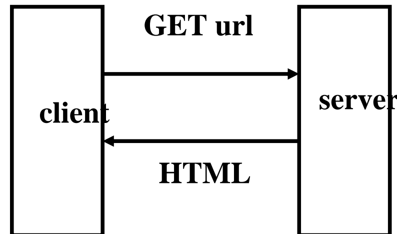
HTTP: Hypertext transfer protocol

- **What happens when you click on a URL?**
- **client sends request:**

  `GET` ***`url`*** **`HTTP/1.0`**

  `[other header info]`

  `(blank line)`



- **server returns**

  `header info`

  `(blank line)`

  `HTML`

  - server returns text that can be created as needed
  - can contain encoded material of many different types

    uses MIME (Multipurpose Internet Mail Extensions)

- **URL format**

  `service://hostname/filename?other_stuff`

`filename?other_stuff` **part can encode**

- data values from client (forms)
- request to run a program on server (cgi-bin)

Forms and CGI-bin programs

- **"common gateway interface"**
  - standard way for client to ask the server to run a program
  - using information provided by the client
  - usually via a form

- **if target file on server is executable program,**
  - e.g., in /cgi-bin directory
- **and if it has right permissions, etc.,**

- **server runs it to produce HTML to send to client**
  - using the contents of the form as input

- **CGI programs can be written in any language**
  - typically Perl, PHP, C, shell, ASP, JSP, ...

- **CGI facility: campuscgi.princeton.edu**
  - anyone can run CGI scripts
  - restrictions on what scripts can access and what they can do

HTML form hello.html

```
<html>
<body>

<FORM ACTION=
  "http://campuscgi.princeton.edu/~bwk/hello1.cgi"
       METHOD=GET>
<INPUT TYPE="submit"
         value="hello1: shell script, plain text">
</FORM>

<FORM ACTION=
  "http://campuscgi.princeton.edu/~bwk/hello2.cgi"
       METHOD=POST>
<INPUT TYPE="submit"
         value="hello2: shell script, html">
</FORM>

[and a bunch of others]

</body>
</html>
```

Simple echo scripts hello[12].cgi

- **Plain text… (hello1.cgi)**

```
#!/bin/sh
echo "Content-type: Text/plain"
echo
echo Hello, world.
```

- **HTML … (hello2.cgi)**

```
#!/bin/sh
echo 'Content-Type: text/html

<html>
<title> Hello2 </title>
<body bgcolor=cyan>
<h1> Hello, world </h1>'

echo "<h2> It's `date` </h2>"
```

- **These have no user input or parameters**
- **though content can change (as in hello2)**

HTML forms: data from users (surv0.html)

```
<html>
<title> COS 333 Survey </title>
<body>
<h2> COS 333 Survey </h2>
<form METHOD=GET ACTION=
   "http://campuscgi.princeton.edu/~bwk/surv2.py">
Name: <input type=text name=Name size=40> <p>
Password: <input type=password name=Pwd <p>
Class: <input type=radio name=Class value=08> '08
      <input type=radio name=Class value=09> '09
<p> CS courses:
<input type=checkbox name=c126> 126
<input type=checkbox name=c217> 217
<p> Experience?
<textarea name=Exp rows=3 cols=40
  wrap></textarea>
<p>
<input type=submit> <input type=reset>
</form>
</body></html>
```

URL encoding of form data

- **how form data gets from client to server**
  - http://hostname/restofpotentially/very/very/longline
  - everything after hostname is interpreted by server
  - usually /program?encoded_arguments
- **if form uses GET, encoded in URL format in QUERY_STRING environment variable**
  - limited length
  - visible in browser, logs, ...
  - can be bookmarked
  - usually used if no change of state at server
- **if form uses POST, encoded in URL format on stdin (CONTENT_LENGTH bytes)**
  - sent as part of message, not in URL itself
  - read from stdin by server
  - no limit on length
  - usually used if causes change of state on server
- **URL format:**
  - keywords in keyword lists separated by +
  - parameters sent as `name=value&name=value`
  - funny characters encoded as %NN (hex)
  - someone has to parse the string
    most scripting languages have URL decoders in libraries

Retrieving info from forms (surv2.py)

- **HTTP server passes info to your cgi program in environment variables**
- **form data available in environment variable QUERY_STRING (GET) or on stdin (POST)**

```
#!/usr/princeton/bin/python

import os
import cgi
form = cgi.FieldStorage()

print "Content-Type: text/html"
print ""
print "<html>"
print "<title> COS 333 Survey </title>"
print "<body>"
print "<h1> COS 333 Survey </h1>"
for i in form.keys():
   print "%s = %s <br>" % (i, form[i].value)
for i in os.environ.keys():
   print "%s = %s <br>" % (i, os.environ[i])
```

Defensive programming

```
char postString[1024];

contentLength =
    atoi(getenv("CONTENT_LENGTH"));
cin.read(postString,contentLength);
```

from a C++ book (4th edition, 2003)

- **program defensively**

  "Always validate all your inputs -- the world outside your function should be treated as hostile and bent upon your destruction."

  Howard & LeBlanc, *Writing Secure Code*, p 80

PHP (www.php.com)

- **an alternative to Perl for Web pages**
  - Rasmus Lerdorf (1997), Andi Gutmans, Zeev Suraski
  - originally Personal Home Pages
  - then PHP Hypertext Processor

- **sort of like Perl turned inside-out**
  - text sent by server
  - after PHP code within it has been executed

```
<html>
<title> PHP hello </title>
<body>
<h2> Hello from PHP </h2>
<?php
echo $_SESSION["NAME"] . "<br>";
echo $_SERVER["HTTP_USER_AGENT"] . "<br>";
echo $_SERVER["REMOTE_ADDR"] . "<br>";
echo $_SERVER["REMOTE_HOST"] . "<br>";
phpinfo();
?>
</body>
</html>
```

PHP version of survey (survey.php)

```
<html>
<title>COS 333 Survey</title>
<h4> COS 333 Survey</h4>

<?php
echo "ENV====\n";
foreach ($_ENV as $key => $value) {
        echo "<br> $key = $value\n";
}
echo "POST=====\n";
$s = "";
foreach ($_POST as $key => $value) {
        echo "<br> $key = $value\n";
        $s .= "$key = $value\n";
}
echo "SERVER======\n";
foreach ($_SERVER as $key => $value) {
        echo "<br> $key = $value\n";
}
?>
<P>
<?php $b = mail("bwk", "survey reply", $s);
echo "mail status = $b\n";
echo "mail message = [$s]\n";
?>

</body>
</html>
```

Formatter in PHP

```
<?
$line = ''; $space = '';
$rh = STDIN;
while (!feof($rh)) {
   $d = rtrim(fgets($rh));
   if (strlen($d) == 0) {
      printline();
      print "\n";
   } else {
      #$words = split("/[\s]+/", $d); #doesn't work
      $words = explode(" ", $d);
      $c = count($words);
      for ($i = 0; $i < $c; $i++)
         if (strlen($words[$i]) > 0)
            addword($words[$i]);
   }
}
fclose($rh);
printline();

function addword($w) {
   global $line, $space;
   if (strlen($line) + strlen($w) > 60)
      printline();
   $line .= $space . $w;
   $space = ' ';
}
function printline() {
   global $line, $space;
   if (strlen($line) > 0)
      print "$line\n";
   $line = ''; $space = '';
}
# the \n after the next line shows up in the output!!
# even if it's removed!!
?>
```

Formatter in Ruby

```
$space = ''
$line = ''

def addword(wd)
   printline() if $line.length()+wd.length()>60
   $line = "#{$line}#{$space}#{wd}"
   $space = ' '
end

def printline()
   print "#{$line}\n" if ($line.length() > 0)
   $line = $space = ''
end

while line = gets()
   line.chop       # get rid of newline
   if (line =~ /^$/)
      printline()
      print "\n"
   else
      line.split().each {|wd| addword(wd) }
   end
end
printline()
```

Why scripting languages?

- **very expressive**
- **efficient enough (usually)**
- **extensible (usually)**
- **portable**

- **good for glue, prototyping**
- **often good enough for production**

- **see John Ousterhout on scripting languages:**
  http://home.pacbell.net/ouster/scripting.html

- **downsides:**
  - creeping featurism
  - inconsistencies among similar languages