

Unix System Administration

Chris Schenk

Lecture 18 – Thursday Mar 13

CSCI 4113, Spring 2008

PHP

- PHP: PHP Hypertext Processor
 - Recursive name geeks at work
- C-like syntax, no types on variables
- Don't have to include anything for functions
 - All functions are available by default
- Fantastic website with function lists and uses
 - <http://www.php.net/manual/en>
- Yet another tool for a sysadmin
 - Sometimes we gotta do some programming

PHP Invocation

- PHP is built for the web by default
 - But can be used for command line scripts
- For web, libphp4.so or libphp5.so must be loaded in Apache
 - .so (shared object) is equivalent to .dll (dynamic link library) under windows
- For command line, an interpreter exists
 - `/usr/bin/php`
 - Works similar to `/bin/bash` for interpreting scripts
 - `#!/usr/bin/php -q`

PHP Files

- By default, files are read as text
 - For presentation to a browser as HTML
- Code must be included in special tags
 - ```
#!/usr/bin/php -q
<?php
print "Hello world\n";
?>
```
- If tags don't exist, text is printed verbatim
  - This is to allow easy typing of HTML without having to 'print' each and every line of HTML

# PHP Syntax

- Again, C/C++ like syntax, without types
  - Also you can get away without use of parenthesis sometimes
- Squiggly brackets *\*must\** be used
  - For if/for/while statements, even if they're one line
  - ```
if($bigSays == "Do work son.") {  
    doWork();  
}
```
- Variables do NOT need to be pre-defined
 - `$command = "dig cs.colorado.edu mx";`
- Always end lines with a semi-colon like C/C++

Variables and Quotes

- Untyped

- Value is interpreted depending on operation

- ```
$string = "The value is: "; #new string
$num1 = "12"; #initialized as string
print "$string $num1\n"; #prints "The value is: 12"
$num1++; #now is 13
print "$string $num1\n"; #prints "The value is: 13"
```

- Double quotes allows expansion of variables

- Arrays don't expand well in quotes

- Use concatenation with period "."

- ```
print "The A record is: ".$values[0]['A']."\n";
```

- Single quotes print literally

- ```
print '$string $num1\n' #prints "$string $num1\n"
```

# Arrays

- Arrays are all dictionaries by default

- key->value pairs

- ```
$randomCrap = array();    #create array
$randomCrap[0] = 5;
$randomCrap['name'] = "chris";
$var = "hello"
$randomCrap[$var] = "there!";
```

- Append to the end of an array:

- ```
$numbers = array(); #create array
$numbers[] = 8; #array position 0
$numbers[] = 143; #array position 1
```

- Print the values of an array:

- ```
print_r($numbers);
```

Functions

- Defined like any other function, practically

```
- function doStuff() {  
    print "Word, I'm doing stuff.\n";  
    print "3 + 4 is ".(3+4)." \n";  
}
```

- Can also take parameters

```
- function doStuffWithParam($num) {  
    print "Your number is $num\n";  
}
```

- Can return early with the 'return' statement
 - Just as always with C/C++

Built-in Functions

- Automatically included libraries if available

- Never have to `#include` or `import` anything!

- `$values = dns_get_record("cs.colorado.edu");`

- `$res = mysql_connect("localhost", "chris", "passw0rd");`

- `if (preg_match("/cs\.colorado\.edu/i", $hostname)) {
 print "Host is in the cs.colorado.edu domain.\n";
}`

- `$file = "/var/log/auth.log";
if($handle = fopen($file)) {
 while ($line = fgets($handle)) {
 print "LINE: $line";
 }
}`

- Suppress warnings with an `@` symbol

- `$handle = @fopen($file);`

Other Constructs

- for loops:

- ```
for ($i=0; $i<10; $i++) {
 print "Index is $i\n";
}
```

- foreach loops:

- ```
foreach ($names as $name) {  
    print "Name is $name\n";  
}
```

- Good for iterating on non-integer based arrays

- Standard keywords work

- break, continue

PHP Examples

- They're friggin' everywhere
 - And a lot of them are bad too
 - No sense of good coding standards
 - Lots of insecure code
- Lab06 has PHP examples in part 4
 - Take a look, but they aren't the best either
 - I should know, I wrote 2/3 of them
- Writing secure code is important for us all
 - Sysadmins have to worry about bad code running on systems, lots of exploits are out there

MySQL

- Free database available for many platforms
 - <http://dev.mysql.com/doc/>
- Two main packages
 - mysql-server, mysql-client
- Client package is the command-line tool and libraries to connect to the db with code
- Server is for actually serving the DB
- SQL == Structured Query Language
 - Not standard among different implementations!

MySQL Login

- Must specify a username and password
 - `% mysql -u root -p`
 - If you omit the password after `-p`, you are prompted
- Logins are a combo of user and host
 - `root@localhost` is default
 - Sometimes an install script will have you set password
 - Other `user@host` combos are added later
 - Host is where the user can connect FROM
- A third piece is usually the database
 - Users are granted access to specific databases

SQL Data Queries

- `<command> <values> <boolean test>`
 - Not the best generalization
- **Select data from a table**
 - `SELECT <columns> FROM <table> WHERE <these things are true>`
- **Insert values into a table**
 - `INSERT INTO <table> <columns> VALUES <values>`
- **Delete rows in a table**
 - `DELETE FROM <table>WHERE <these things are true>`
- **Modify rows in a table**
 - `UPDATE <table> SET <column>=<value> WHERE <these things are true>`

Column Types

- Lots of different ones
 - char, varchar, int, timestamp, blob, and more
- Every column must specify a type
 - Char – single character
 - Varchar(n) – String of (max) length n
 - Int – a simple integer
 - Float/double – for real numbers
- Types can be defined non-null, have init values
 - `id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT`

SQL Meta Queries

- Stuff for creating DBs/tables, altering tables
- Create a database;
 - `CREATE DATABASE <db>;`
- Create a table
 - `CREATE TABLE <table>;`
- Alter an existing table
 - `ALTER TABLE <table> ADD <column> <type>;`
- Drop a table or database
 - `DROP TABLE <table>;`
 - `DROP DATABASE <db>;`

Creating Users

- Easiest way to create users is to first create a database and add users to it
 - `CREATE DATABASE mydb;`
- Users added with the 'GRANT' command
- `GRANT ALL PRIVILEGES ON <db>.<tables> TO '<user>'@'<host>';`
 - `GRANT ALL PRIVILEGES ON mydb.* to 'chris'@'localhost'`
 - Host is where the user can connect FROM!

Sample Database

- **CREATE DATABASE login_monitor;**
- **USE login_monitor;**
 - Tell MySQL that we're modifying this database
- **Create a table for tracking user logins**
 - ```
CREATE TABLE logins (
 id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
 username VARCHAR(20),
 ip VARCHAR(15),
 time TIMESTAMP DEFAULT 0);
```
- **Insert some data into it:**
  - ```
INSERT INTO logins (username, ip, time) VALUES  
  ('chris', '128.138.242.249', NOW());
```

Viewing Sample Data

- Select the information out of the 'logins' table

```
- mysql> SELECT * FROM logins;
```

```
+-----+-----+-----+-----+
| id | username | ip | time |
+-----+-----+-----+-----+
| 1 | chris | 128.138.242.249 | 2008-03-12 22:36:09 |
+-----+-----+-----+-----+
```

```
1 row in set (0.00 sec)
```

- Select a specific column

```
- mysql> SELECT username FROM logins;
```

```
+-----+
| username |
+-----+
| chris |
+-----+
```

```
1 row in set (0.00 sec)
```

Selecting Specific Data

- Let's add one more row:

```
- INSERT INTO logins (username, ip, time) VALUES ('root',  
  '128.138.242.249', NOW());
```

- Select only root:

```
- mysql> SELECT * FROM logins WHERE username='root';  
+----+-----+-----+-----+  
| id | username | ip | time |  
+----+-----+-----+-----+  
| 2 | root | 128.138.242.249 | 2008-03-12 22:41:44 |  
+----+-----+-----+-----+  
1 row in set (0.00 sec)
```

- Select IP 128.138.242.249:

```
- mysql> SELECT * FROM logins WHERE ip='128.138.242.249';  
+----+-----+-----+-----+  
| id | username | ip | time |  
+----+-----+-----+-----+  
| 1 | chris | 128.138.242.249 | 2008-03-12 22:36:09 |  
| 2 | root | 128.138.242.249 | 2008-03-12 22:41:44 |  
+----+-----+-----+-----+  
2 rows in set (0.00 sec)
```