Server side basics
Be careful

- Do not type any command starting with "sudo" into a terminal attached to a university computer.
- You have complete control over your AWS server, just as you have complete control over your laptop.
- University computers are shared—"sudo" overrides protection of other users of the computer.
SM: Initialize Git on Betaweb

AWS$ pwd
/home/ec2-user/
AWS$ git clone <you>@betaweb.csug.rochester.edu:/p/csc210/<your team repo>
AWS$ cd <your team repo>
AWS$ cp -r /var/www/html/* .
AWS$ git add .
AWS$ git status
AWS$ git commit
AWS$ git push origin master

Send email to team telling them that you have set up the master directory on Betaweb

CSC 210
Everyone: Getting Betaweb to AWS

AWS> pwd
/home/ec2-user/

AWS> git clone <you>@betaweb.csug.rochester.edu:/p/csc210/<your team repo>

AWS> cd <your team repo>

AWS> ls

Is everything there?
Everyone: Create personal branch

$ git branch
* master

$ git branch martin

$ git branch
  martin

* master

$ git checkout martin
Switched to branch 'martin'

$ git branch
* martin
  master
Git diagram

add (-u) → commit

workspace → index → local repository

pull or rebase

push → fetch

remote repository
Everyone: Editing a file

$ cd 0
$ emacs index.html
$ cat index.html
...
<p>
(describe what you did, including any extra credit, here.)

Added this to my branch on AWS
</p>
...

$ git add index.html
$ git status
# On branch martin
# Changes to be committed:
...
$ git commit
$ git push origin martin
martin@betaweb.csug.rochester.edu’s password:
SM: Setting up as SM

betaweb$ pwd
/home/<scrum master’s home>

betaweb $ git clone /p/csc210/<your team repo>

betaweb $ git branch
* master

betaweb $ git branch staging

betaweb $ git branch -a
* master
  staging
  remotes/origin/HEAD -> origin/master
  remotes/origin/martin
  remotes/origin/master
SM: Merging work

$ git checkout <branch with new work>
$ git pull origin martin
$ git checkout staging
$ git merge martin
$ git push origin staging
SM: Testing new work in staging

$ pwd
/home/ec2-user/<your team repo>
$ git branch staging
$ git checkout staging
$ git pull origin staging
$ cp -r ./* /var/www/html/
SM: Testing new work

ReadMe for Assignment 0

Assignment 0

(describe what you did, including any extra credit, here.) Added this to my branch on AWS
SM: Testing new work in staging

$ pwd
/home/ec2-user/<your team repo>

$ emacs index.html
...

<p>
(describe what you did, including any extra credit, here.)
</p>

<p>
Added this to my branch on AWS
</p>
SM: Testing new work in staging

ReadMe for Assignment 0

Assignment 0

(describe what you did, including any extra credit, here.)

Added this to my branch on AWS
$ git add index.html
$ git commit
$ git push origin staging

$user>@betaweb .. Password:
$ git checkout master
$ git merge staging
$ git push origin master

Betaweb password:

Send email to team saying that the Master has been updated
Quiz

- Write CSS code put a paragraph: in the horizontal center of the page, occupying half the page’s width, and right-aligning the paragraph’s text?
- What is the difference between “display: none” and visibility: “hidden”
- Is PHP code static or dynamic web content?
- Given <?php $bar = “foo” ?> write a php block that will put “foo” on the page.
Quiz Answers

1. p {
   margin-left: auto;
   margin-right: auto;
   width: 50%;
}

2. “display: none” leaves no space; “visibility: hidden” leaves space.

3. PHP is dynamic web content?

4. <?php $bar ?>
Server side basics
Usually when you type a URL in your browser:

- Your computer looks up the server's IP address using DNS
- Your browser connects to that IP address and requests the given file
- The web server software (e.g. Apache) grabs that file from the server's local file system
- The server sends back its contents to you
Some URLs actually specify programs that the web server should *run*, and then send their output back to you as the result:

- The above URL tells the server `facebook.com` to run the program `home.php` and send back its output.
Server-Side web programming

- Server-side pages are programs written using one of many web programming languages/frameworks
  - examples: PHP, Java/JSP, Ruby on Rails, ASP.NET, Python, Perl
Also called *server side scripting*:

- Dynamically edit, change or add any content to a Web page
- Respond to user queries or data submitted from HTML forms
- Access any data or databases and return the results to a browser
- Customize a Web page to make it more useful for individual users
- Provide security since your server code cannot be viewed from a browser
Server-Side web programming (cont.)

- **Web server:**
  - contains software that allows it to run server side programs
  - sends back their output as responses to web requests
- **Each language/framework has its pros and cons**
  - we use PHP
What is PHP?

- PHP stands for "PHP Hypertext Preprocessor"
- Server-side scripting language
- Used to make web pages dynamic:
  - provide different content depending on context
  - interface with other services: database, e-mail, etc.
  - authenticate users
  - process form information
- PHP code can be embedded in XHTML code
Lifecycle of a PHP web request

User's computer

Web Browser

GET

Hello.php

Server computer

Web Server

Hello world!
Why PHP?

- Free and open source
- Compatible
  - as of November 2006, there were more than 19 million websites (domain names) using PHP.
- Simple
Viewing PHP output

Hello world!
PHP Basic Syntax
PHP syntax template

- Contents of a .php file between `<?php` and `?>` are executed as PHP code.
- All other contents are output as pure HTML.
- We can switch back and forth between HTML and PHP "modes".
Hello world! Escape "chars" are the SAME as in Java! You can have line breaks in a string. A string can use "single-quotes". It's cool!
Variables

```php
$name = expression;
$user_name = "mundruid78";
$age = 16;
$drinking_age = $age + 5;
$this_class_rocks = TRUE;
```

- names are case sensitive
- names always begin with $, on both declaration and usage
- always implicitly declared by assignment (type is not written)
- a loosely typed language (like JavaScript or Python)
Variables

- basic types: int, float, boolean, string, array, object, NULL
- test type of variable with is_type functions, e.g. is_string
- gettype function returns a variable's type as a string
- PHP converts between types automatically in many cases:
  - string → int auto-conversion on +
  - int → float auto-conversion on /
- type-cast with (type):
  - $age = (int) "21";
Arithmetic operators

- +  -  *  /  %  .  ++  --
- =  +=  -=  *=  /=  %=  .=

- many operators auto-convert types: 5 + "7" is 12
Comments

- like Java, but # is also allowed
  - a lot of PHP code uses # comments instead of //
String Type

- zero-based indexing using bracket notation
- there is no char type; each letter is itself a String
- string concatenation operator is . (period), not +
  - $5 + "2 turtle doves" === 7$
  - $5 . "2 turtle doves" === "52 turtle doves"
- can be specified with "" or ""
$name = "Stefanie Hatcher";
$length = strlen($name);
$cmp = strcmp($name, "Brian Le");
$index = strpos($name, "e");
$first = substr($name, 9, 5);
$name = strtoupper($name);
## String Functions (cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Java Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>strlen</td>
<td>length</td>
</tr>
<tr>
<td>strpos</td>
<td>indexOf</td>
</tr>
<tr>
<td>substr</td>
<td>substring</td>
</tr>
<tr>
<td><code>strtolower, strtoupper</code></td>
<td>toLowerCase, toUpperCase</td>
</tr>
<tr>
<td>trim</td>
<td>trim</td>
</tr>
<tr>
<td><code>explode, implode</code></td>
<td>split, join</td>
</tr>
<tr>
<td><code>strcmp</code></td>
<td>compareTo</td>
</tr>
</tbody>
</table>
Interpreted Strings

```php
$age = 16;
print "You are " . $age . " years old.\n";
print "You are \$age years old.\n"; # You are 16 years old.
```

- strings inside " " are interpreted
  - variables that appear inside them will have their values inserted into the string
- strings inside ' ' are not interpreted:

```php
print 'You are \$age years old.\n'; # You are $age years old. \n
```

CSC 210
Interpreted Strings (cont.)

print "Today is your {$age}th birthday.\n";

- if necessary to avoid ambiguity, can enclose variable in {}

```php
print "Today is your $ageth birthday.\n"; # $ageth not found
print "Today is your {$age}th birthday.\n";
```
Interpreted Strings (cont.)

```php
$name = "Xenia";
$name = NULL;
if (isset($name)) {
    print "This line isn't going to be reached.\n";
}
```

- a variable is NULL if
  - it has not been set to any value (undefined variables)
  - it has been assigned the constant NULL
  - it has been deleted using the unset function
- can test if a variable is NULL using the isset function
- NULL prints as an empty string (no output)
for loop (same as Java)

```php
for (initialization; condition; update) {
    statements;
}
```

```php
for ($i = 0; $i < 10; $i++) {
    print "$i squared is $i * $i . " . \n";
}
```
bool (Boolean) type

```php
$feels_like_summer = FALSE;
$php_is_great = TRUE;
$student_count = 7;
$nonzero = (bool) $student_count; # TRUE
```

- the following values are considered to be FALSE (all others are TRUE):
  - 0 and 0.0 (but NOT 0.00 or 0.000)
  - "", "0", and NULL (includes unset variables)
  - arrays with 0 elements
- FALSE prints as an empty string (no output); TRUE prints as a 1
if (condition) {
    statements;
} elseif (condition) {
    statements;
} else {
    statements;
}
while loop (same as Java)

```php
while (condition) {
    statements;
}
```

```php
do {
    statements;
} while (condition);
```
Math operations

$a = 3;$
$b = 4;$
$c = sqrt(pow($a, 2) + pow($b, 2));

math functions

<table>
<thead>
<tr>
<th>abs</th>
<th>ceil</th>
<th>cos</th>
<th>floor</th>
<th>log</th>
<th>log10</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>pow</td>
<td>rand</td>
<td>round</td>
<td>sin</td>
<td>sqrt</td>
<td>tan</td>
</tr>
</tbody>
</table>

math constants

| M_PI | M_E  | M_LN2 |
Int and Float Types

- int for integers and float for reals
- division between two int values can produce a float

```php
$a = 7 / 2; # float: 3.5
$b = (int) $a; # int: 3
$c = round($a); # float: 4.0
$d = "123"; # string: "123"
$e = (int) $d; # int: 123
```
For your first PHP exercise, echo the following statement to the browser:

“Twinkle, Twinkle little star.”

Next, create two variables, one for the word “Twinkle” and one for the word “star”. Echo the statement to the browser, this time substituting the variables for the relevant words. Change the value of each variable to whatever you like, and echo the statement a third time. Remember to include code to show your statements on different lines.
PHP exercise 2

- PHP includes all the standard arithmetic operators. For this PHP exercise, you will use them along with variables to print equations to the browser. In your script, create the following variables:
  
  ```php
  $x=10;
  $y=7;
  ```

- Write code to print out the following:
  
  10 + 7 = 17
  10 - 7 = 3
  10 * 7 = 70
  10 / 7 = 1.4285714285714
  10 % 7 = 3

- Use numbers only in the above variable assignments, not in the echo statements.
PHP exercise 3

- Arithmetic-assignment operators perform an arithmetic operation on the variable at the same time as assigning a new value. For this PHP exercise, write a script to reproduce the output below. Manipulate only one variable using no simple arithmetic operators to produce the values given in the statements.

- Hint: In the script each statement ends with "Value is now $variable."

Value is now 8.
Add 2. Value is now 10.
Subtract 4. Value is now 6.
Multiply by 5. Value is now 30.
Divide by 3. Value is now 10.
Increment value by one. Value is now 11.
Decrement value by one. Value is now 10.
When you are writing scripts, you will often need to see exactly what is inside your variables. For this PHP exercise, think of the ways you can do that, then write a script that outputs the following, using the echo statement only for line breaks.

```php
string(5) "Harry"
Harry
int(28)
NULL
```
For this PHP exercise, write a script using the following variable:

```php
$around="around";
```

Single quotes and double quotes don't work the same way in PHP. Using single quotes (' ') and the concatenation operator, echo the following to the browser, using the variable you created:

```
What goes around, comes around.
```
PHP exercise 5

In this PHP exercise, you will use a conditional statement to determine what gets printed to the browser. Write a script that gets the current month and prints one of the following responses, depending on whether it's August or not:

- It's August, so it's really hot.
- Not August, so at least not in the peak of the heat.

Hint: the function to get the current month is `date('F', time())` for the month's full name.
Loops are very useful in creating lists and tables. In this PHP exercise, you will use a loop to create a list of equations for squares.

Using a for loop, write a script that will send to the browser a list of squares for the numbers 1-12. Use the format, "1 * 1 = 1", and be sure to include code to print each formula on a different line.
PHP exercise 7

- HTML tables involve a lot of repetitive coding - a perfect place to use for loops. You can do even more if you nest the for loops.

- In this PHP exercise, use two for loops, one nested inside another. Create the following multiplication table:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
</tr>
</tbody>
</table>