Introduction to Web Programming

PIC 40A, UCLA

Introduction to Web Programming

Before writing websites, it’s important to have a context for what the internet is and roughly how web browsers work.

Information relayed between a web browser and the websites comes in many forms from **html** and **css** for display, through to scripts (such as **JavaScript**, etc.) that enable interactivity and some dynamic features. There is also much going on "behind the scenes" that enables information to be processed on the side of a webpage such as the use of **PHP** and **databases**.

Networks

A **network** is a group of computers that can send/receive resources (data) to/from each other.

The **internet** is a network of networks.

Aside: there is debate about whether "internet" should be capitalized. Originally it was capitalized, being a proper noun. But its frequent casual use over time has gradually decreased the frequency of "Internet".

Networks

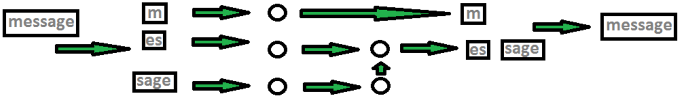
There are two types of networks: **Local Area Network**s (**LAN**s) and **Wide Area Network**s (**WAN**s).

A **LAN** connects computers within a small geographic region (school, hospital, etc.), whereas a **WAN** connects computers over large distances, using special cables/lines.

A Brief History of the Internet

In the 1960s, Leonard Kleinrock (UCLA Professor of Computer Science) and his team developed the **packet switching** method to allow messages to be sent electronically more quickly and efficiently over a network called **ARPANET.** At the time it was funded by the Advanced Research Projects Agency and the US Department of Defense.

With packet switching, instead of a message being sent as a whole, it is broken into pieces, the pieces are sent their separate ways, and the pieces get reassembled on the receiving end.



A Brief History of the Internet

In 1969, the internet began - sort of. On October 29, 1969 at 22:30, Kleinrock and his team attempted to send the message "LOGIN" from a computer at UCLA to collaborators at the Stanford Research Institute using ARPANET.

A Brief History of the Internet

In 1969, the internet began - sort of. On October 29, 1969 at 22:30, Kleinrock and his team attempted to send the message "LOGIN" from a computer at UCLA to collaborators at the Stanford Research Institute using ARPANET.

Only "LO" got sent before the system crashed...

A Brief History of the Internet

Things grew quickly and more computers became part of the ARPANET. But there was no standard format of sending/receiving data between computers and this made communication difficult...

In the 1970s, the **TCP** (Transmission Control Protocol) was established along with **IP** (Internet Protocol). Together they form the **TCP/IP Protocol**.

A Brief History of the Internet

TCP dictates how information is broken into packets and reassembled, and how errors and confirmations in data transmission are managed.

IP is used for ascribing addresses to sources and destinations.

By IPv4 standards (older), an IP address is of the form **nnn.nnn.nnn.nnn** where each of **nnn** is an integer from 0-255. Although very common, eventually this formatting will run out of possible addresses.

By IPv6 standards (newer, has many more possible addresses), an IP address is of the form **nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn:nnnn**, i.e. 8 blocks, separated by a colon. Each block is a hexadecimal (base 16, uses digits 0-9,a-f) number.

A Brief History of the Internet

In principle, one could type **172.217.11.68** or **0:0:0:0:0:ffff:acd9:b44** instead of **www.google.com** since that is the address of the Google server and this could be done for other sites, too.

But having direct access to a server can be dangerous due to hackers and various attacks so this sort of direct access can be blocked on different networks and by different websites. It is also the case some machines are hosting multiple websites, such as the large web hosting companies.

A Brief History of the Internet

By the 1980's, many personal computers had the internet and networks other than ARPANET began to take over.

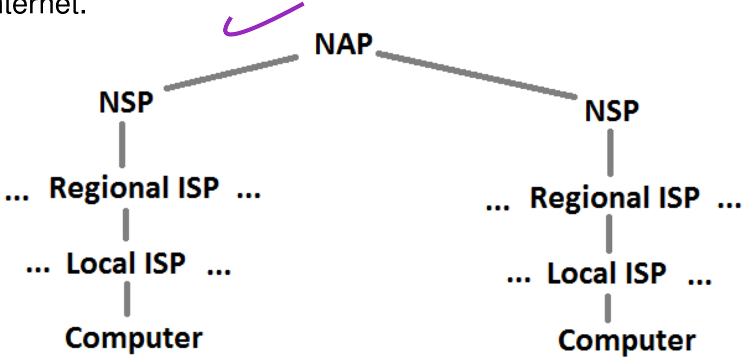
In the 1990's, the internet began to expand with regular folks having access to email, chatrooms, etc.

✓ **Remark:** the world wide web is a collection of webpages that are accessed through the internet. But the internet (the network of networks) is much bigger.

Structure of the Internet

Communication between computers connected to the internet is managed by sending information from the computer to its local **Internet Service Provider (ISP)**, then to a regional ISP, then to a **Network Service Provider (NSP)**, then through a **Network Access Point (NAP)**, and back down to the other computer.

NSPs are sometimes called **backbones** because they are the backbone of the internet.



URLs, URIs, and URNs

A **Uniform Resource Identifier (URI)** is a string of characters used to identify a resource.

The most common form of URI online is a **Uniform Resource Locator (URL)**. A URL specifies the location of a resource on the web - there is an associated address. An example is **www.math.ucla.edu**.

A **Uniform Resource Name (URN)** is basically a scheme to identify a resource by its name, but an address is not necessarily given. An example would be the *ISBN number of a book* - that does not give it a location, but it should tell us unambiguously what we are looking for.

Hosts and Servers

A **host** is an end device in a network, e.g. a local workstation, laptop, or a file server. Each host has an IP address. A **hostname** is a name that points to a specific host.

A **server** is a computer that has software installed to receive requests from other hosts and return data.

All servers are hosts. Not all hosts are servers.

Domain Names

A **domain name** identifies one or more IP addresses. An example of a domain name is **ucla.edu** within **www.pic.ucla.edu**.

A domain name can have multiple subdomains, forming a hierarchy. The **edu** is the top-level domain name. This encloses the second-level domain name **ucla**, which encloses another domain name **pic**.

People developing websites can purchase domains within the top-level domains (com, edu, ca, org, etc.). So **ucla** is a domain belonging to the **edu** top level domain name.

Once a domain has been created, subdomains can be created, such as **math.ucla.edu** within **ucla.edu**.

Domain Names

A **fully qualified domain name** is one where all levels of the hierarchy are specified to identify a single host (usually).

www.pic.ucla.edu is fully qualified but **pic.ucla.edu** and **ucla.edu** are only domains.

Adding the prefixes like **www** in **www.pic.ucla.edu** makes the fully qualified domain name into what is called a **logical hostname**. The host itself could have a different name; for the PIC website, this is likely the **laguna** server at **laguna.pic.ucla.edu**.

Domain Names

You may also notice many URLs are prefixed by **http:**// or **https:**//. These are specifying the means that data are transferred**.**

http = hypertext transfer protocol

https = hypertext transfer protocol secure

And there are many others.

Domain Names

The domain name part of a URL is case insensitive, but navigation through folders, subfolders, and files may or may not be case sensitive, depending on the server.

wWw.pic.ucla.edu - good
www.pic.ucla.edu - good

www.pic.ucla.edu/about - good.
www.pic.ucla.edu/AboUt - might not work (it does in this case).

According to Stack Overflow, one should assume URLs **are** case sensitive.

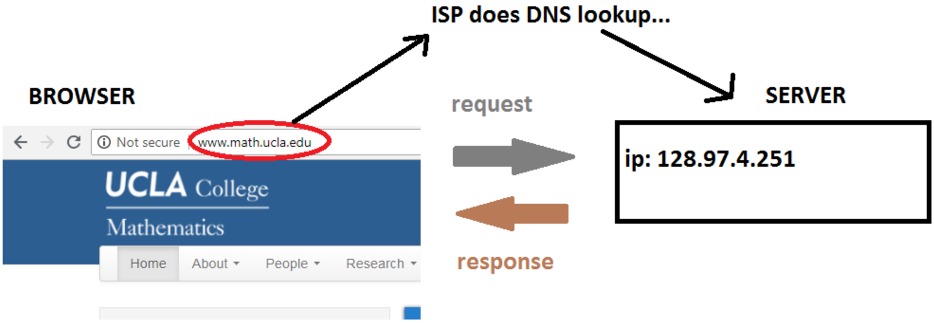
How Web Browsers Work

A web browser is a program that communicates with an Internet Service Provider (ISP) in order to send/receive information to/from a server.

At a very superficial level, the brower asks for data from the server; the server sends data back and everything just works...

How Web Browsers Work

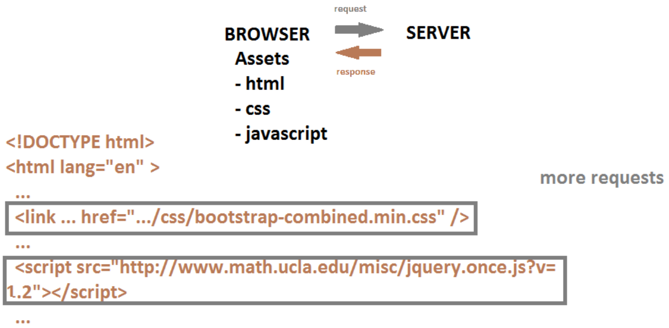
When we type a URL that is comprehensible to a human, by performing a **DNS** (Domain Name System) lookup through the ISP, the web browser connects with the proper server.



Remark: some big companies like Google and Cloudflare also have their own DNS lookup.

How Web Browsers Work

Once the connection to the server is established, information is sent in different forms and the web browser may request more assets - things that help it to display information, enable interactivity, etc.



```
<!DOCTYPE html>
<html lang="en" >
  ...
  <link ... href=".../css/bootstrap-combined.min.css" />
  ...
  <script src="http://www.math.ucla.edu/misc/jquery.once.js?v=
1.2"></script>
  ...
```

How Web Browsers Work

A request has a precise form that we will be relevant later. But a few key items to get a sense of things:

- **Host:** where the connection is sought, such as math.ucla.edu
- **Method:** the purpose of the request such as GET (to get a resource) or POST (to log in), etc.
- **Path:** specific directory within the website such as math.ucla.edu/ **courses**
- **Post-Body:** what is being POSTed such as the login credentials

How Web Browsers Work

Likewise, a response also has a precise form. A few key components:

- **Content-Type:** the type of information it is providing such as "text/html" (html), "text/plain" (plain text), "image/jpeg" (jpeg picture), etc.
- **Status:** result of the request (100's = in progress, 200's = success, 300's = redirection, 400's = error on client side, 500's = error on server side)
- **Response-Body:** the desired file

How Web Browsers Work

HTML: Hyper**T**ext **M**arkup **L**anguage. It is a standardized system used to provide layout and structure to text (and graphics, etc.) on a webpage.

Hypertext refers to text that links to other documents.

How Web Browsers Work

CSS: Cascading **S**tyle **S**heets. These are used to provide formatting specifications to elements of a page, such as font size, margin width, background color, etc.

The **cascading** refers to a hierarchy of how these formats are applied.

Neither HTML nor CSS are programming languages.

How Web Browsers Work

JavaScript: is a very high level, client-side programming langauge used most often for adding interactivity to webpages.

Adding a clock that counts down, peforming calculations and displaying results based on a user's input, etc., could easily be done with JavaScript.

Remark: sometimes "client-side" is called "front end" because it is what the client can directly see/interact with.

LAMP

A very common and popular platform for developing websites is **LAMP** (Linux-Apache-MySQL-PHP). That is,

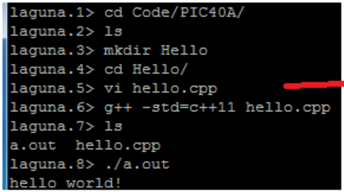
- the operating system is Linux (some use Windows and are WAMP);
- the web server software is Apache;
- MySQL (or alternatives like MongoDB) manages the website organization and information retrieval; and
- PHP (or alternatives like Perl or Python) is used for the server-side programming.

LAMP

Linux is an open source operating system (originally developed by Linus Torvalds) written in C.

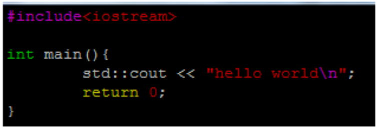
It allows multiple users to process multiple things at once, and easy management of data.

Many Linux functionalities can be performed with the Command Line rather than with Graphical User Interfaces, although there is graphical support, too.



```
laguna.1> cd Code/PIC40A/
laguna.2> ls
laguna.3> mkdir Hello
laguna.4> cd Hello/
laguna.5> vi hello.cpp
laguna.6> g++ -std=c++11 hello.cpp
laguna.7> ls
a.out  hello.cpp
laguna.8> ./a.out
hello world!
```

opens plain text editor



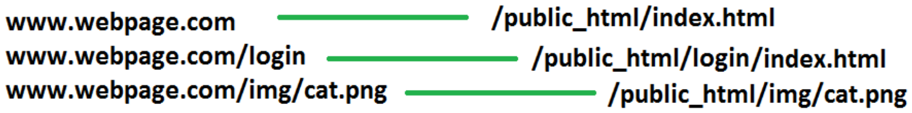
```
#include<iostream>

int main(){
        std::cout << "hello world\n";
        return 0;
}
```

LAMP

Apache is the most popular web server software. It is pronounced "a patchy", apparently because during its initial development, it used existing files from older server projects along with extra code to patch things together. It was *a patchy server*.

A **web server** manages data of a website and delivers/serves it to clients who visit the page. Based on the page they request, it will retrieve data from different folders.

LAMP



Many server software require a folder **public_html** where the webpage material is stored.

When no path is specified besides the path to the main page, the file **index.html** will be served (this should be the name of the main webpage!).

Likewise, within subdirectories, the default HTML page to be served is **index.html**.

This is the case for the PIC server! There must be a **public_html** directory and by default **index.html** will be served in any given folder.

LAMP

MySQL is a type of **Structured Query Language** (SQL) database. A database is a structed collection of data in a computer that can be accessed in different ways.

It is kind of like a spreadsheet with different row/column headers and ways of sorting and searching through the data.

When data is needed, such as when a user’s profile information needs to be loaded or a password needs to be validated, requests can be made through MySQL queries to the database.

SQLite is a lightweight alternative to MySQL; it has fewer features.

LAMP

PHP (**P**HP: **H**ypertext **P**reprocessor) is a high level server-side programming language. Sometimes this "server-side" is known as the "back end" because it is not visible to the client.

When PHP scripts run, they generate HTML code that a web brower displays.

Using Linux

Learning commands in a Linux operating system has quite a learning curve. Here, we'll go over a few of the basics that should be sufficient in getting around with basic web programming.

In this course we will use the **laguna** server. Logging in can be done with the credentials: **username@laguna.pic.ucla.edu**

If you are using Windows, open the Command Prompt. If you are on Mac or already using Unix, open the Terminal. Type: **ssh username@laguna.pic.ucla.edu**. You will be prompted for a password (you won't see anything as you type it).

Using Linux

Here are some of the most basic/useful commands:

ls: lists the contents of the current directory

pwd: gives the absolute location of the current directory

cd: change directory, should be followed by a valid new directory, e.g.

```
> cd myFolder
> cd ../myOtherFolder
```

The **..** indicates the parent directory so above, we are to go up one directory and into the myOtherFolder folder. In Linux / is used to separate directory hierarchies.
Note: we use $>$ to indicate the start of a Linux command.

Using Linux

mkdir: make a directory, should be followed by the directory name

rmdir: remove a directory, should be followed by the directory name and can only be done if a directory is empty

rm: remove a file, should be followed by the file name.

mv: move/rename a file, must be followed by the file (or its relative location) and then by the destination folder. Suppose **foo.txt** is in the current directory.

```
> mkdir FooFolder
> mv foo.txt ./FooFolder
```

The **.** signifies the current directory. Technically we don't need the **./** and it should still work.

Using Linux

```
> mv foo.txt bar.txt
> rm bar.txt
```

Above we renamed **foo.txt** to **bar.txt**, assuming **foo.txt** was in our local directory. Then we deleted the file **bar.txt**.

Using Linux

The **touch** command creates an empty file with a given name.

```
> touch foo.dat
```

Now we have an empty file **foo.dat**.

Using Linux

cp: copy a file from a source to a destination.

*: indicates anything/all.

```
> mkdir Dest
> touch a
> touch b
> cp * Dest
```

Here **a** and **b** will be moved into **Dest**.

Using Linux

The **-f** flag can be used to force a command without Linux asking for approval. The **-r** flag indicates a recursive call.

```
> rm -f example.txt
```

The command above will remove **example.txt** without any prompts.

```
> cp -r X Y
```

The command above will recursively copy the folder **X** to the folder **Y**.

The **-rf** can be used together, too. And **-r** is needed to recursively delete non-empty directories.

Using Linux

The IP address of a website can be looked up with the **dig** command with the **+short** option added:

```
> dig +short www.google.com
142.250.72.164
```

vi

Perhaps the most primitive text editor in the world, **vi** can be handy for **very, very simple text files.** Using it for anything substantial is rather painful.

To **create/open** a file:

```
> vi foo.txt
```

This opens up a vi session to read/edit/write the file "foo.txt".

vi

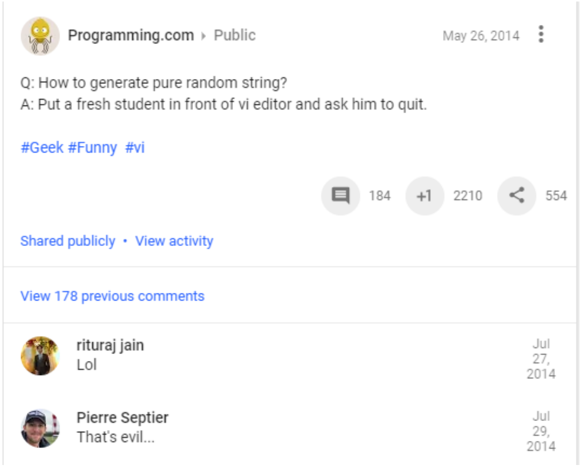
To insert/edit text, type **i** to indicate you wish to insert. Then you can edit.

Use the arrow keys to navigate the file.

Use **delete** to delete items; sometimes **shift** may also need to be pressed.

One can also delete characters by pressing **Esc** to enter command mode and then pressing **x**.

vi



Programming.com › Public May 26, 2014

Q: How to generate pure random string?
A: Put a fresh student in front of vi editor and ask him to quit.

#Geek #Funny #vi

184 +1 2210 554

Shared publicly • View activity

View 178 previous comments

rituraj jain
Lol
Jul 27, 2014

Pierre Septier
That's evil...
Jul 29, 2014

vi

Quitting is a whole other story**...**

To **quit and save**: press **Esc** then type **:wq** then **Enter.**

To **quit without saving**: press **Esc** then type **:q!** then **Enter.**

less

less is a handy text viewer. One simply types **less** plus the name of a file to view its contents. Scrolling can be done with the arrow keys. To escape, press **q**.

Transferring Files

Transferring files between computers can be done through the Command Prompt/Terminal and **scp** (secure copy protocol).

To **copy** the local file **foo.txt** to a directory **Foo** on the **user**'s **laguna** server space:

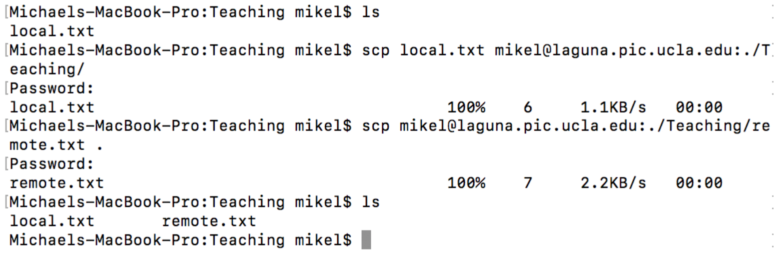
```
> scp foo.txt user@pic.laguna.ucla.edu:./Foo/
```

To **copy** the file **foo.txt** from the **Foo** folder of **user**'s **laguna** server space to the local directory:

```
> scp user@pic.laguna.ucla.edu:./Foo/foo.txt .
```

Note that the remote host files have the format *user@host:.[rest_of_path]*.

Transferring Files

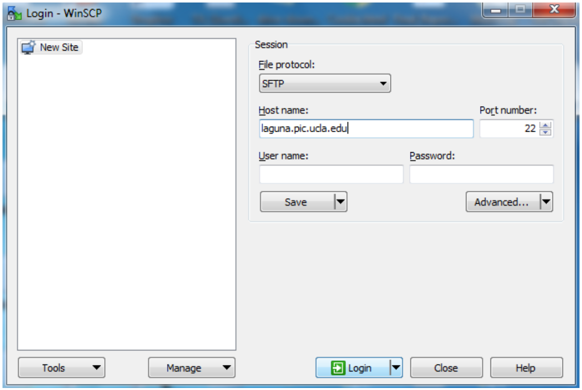


```
[Michaels-MacBook-Pro:Teaching mikel$ ls
local.txt
[Michaels-MacBook-Pro:Teaching mikel$ scp local.txt mikel@laguna.pic.ucla.edu:./T
eaching/
[Password:
local.txt                                          100%    6     1.1KB/s   00:00
[Michaels-MacBook-Pro:Teaching mikel$ scp mikel@laguna.pic.ucla.edu:./Teaching/re
mote.txt .
[Password:
remote.txt                                         100%    7     2.2KB/s   00:00
[Michaels-MacBook-Pro:Teaching mikel$ ls
local.txt       remote.txt
Michaels-MacBook-Pro:Teaching mikel$
```

Transferring Files

A more user-friendly approach is to use an **FTP Client** (File Transfer Protocol). We consider the example with **WinScp** for Windows.

One specifies the host name to log in.



File protocol: SFTP

Host name: laguna.pic.ucla.edu

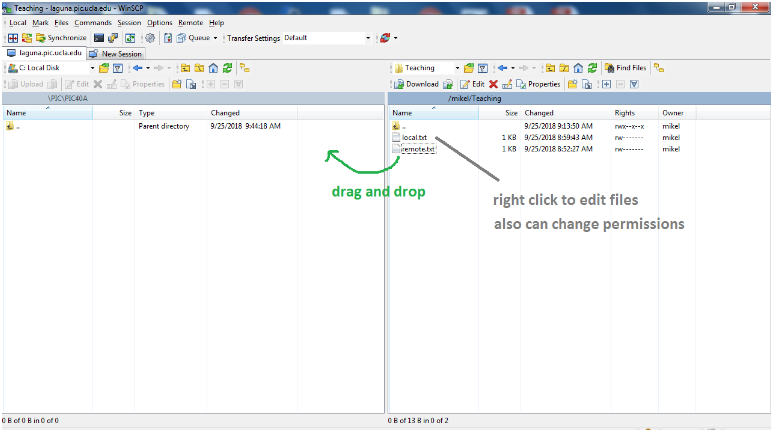
Port number: 22

User name:

Password:

Transferring Files

Once logged in, files can be dragged/dropped to transfer. Files can also be edited through the interface.


Teaching - laguna.pic.ucla.edu - WinSCP
Local Mark Files Commands Session Options Remote Help
Synchronize
Queue
Transfer Settings Default
laguna.pic.ucla.edu
New Session
C: Local Disk
Upload
Edit
Properties
Teaching
Find Files
Download
Edit
Properties
drag and drop
right click to edit files
also can change permissions

\PIC\PIC40A

Name	Size	Type	Changed
..		Parent directory	9/25/2018 9:44:18 AM

0 B of 0 B in 0 of 0

/mikel/Teaching

Name	Size	Changed	Rights	Owner
..		9/25/2018 9:13:50 AM	rwx--x--x	mikel
local.txt	1 KB	9/25/2018 8:59:43 AM	rw-------	mikel
remote.txt	1 KB	9/25/2018 8:52:27 AM	rw-------	mikel

0 B of 13 B in 0 of 2

Permissions

In Linux, all files and folders have a **permission** status. This specifies the **read**, **write**, and **execute** permissions.

In turn, each of these permissions can be specified for the **user**, the **group**, and **other**.

Permissions

Permissions are important so that a webpage can actually be accessed by people surfing the net! And without proper permissions, some scripts may not run and a webpage will malfunction.

Just think: if someone wants to go to your webpage, they need to access the **.html** file you wrote. But if they don't have permission, there is no way for them to see it...

Permissions

For a **folder**,

- **read** indicates whether its contents can be shown;
- **write** indicates whether its contents can be modified; and
- **execute** indicates whether one can access it through **cd**, whether it can be searched

To prevent a folder or *any of its contents* from being accessed, the execute field can be turned off.

Permissions

For a **file**,

- **read** indicates whether its contents can be read;
- **write** indicates whether its contents can be edited; and
- **execute** indicates whether it can be executed (i.e. "run" in the case of a program)

Permissions

Regarding the different groups of people who can access, there is

- **user:** the person who is logged in and has an account with the computer;
- **group:** each user can belong to one or more groups that are set by the administrators;
- **other:** anyone else, the general public, etc.

A user will typically grant more restrictive access to group/other than themselves.

Permissions

The permissions of a **folder** take the form of 10 characters: **d.........** where each **.** is a **-, r, w,** or **x**.

The permissions of a **file** take the form of 10 characters: **-.........** where each **.** is a **-, r, w,** or **x**.

More specifically, the first three **.**'s represent permissions for the user, the next three represent permissions for the group, and the final 3 represent the permissions for other.

Permissions

The permissions for each of the audiences are each 3 characters for read/write/execute, respectively.

If a folder/file can be read/written/executed, respectively, we use **r**/**w**/**x**, respectively. A **-** indicates a lack of that permission. For example:

r-x: can be read and executed, but not written.

Permissions

So a Folder that can be read, written, and executed for all of user/group/other has the form:

drwxrwxrwx

A file that can be read, written, and executed for the user, but only read/executed by everyone else has the form:

-rwxr-xr-x

Permissions

There is also a numerical representation of the permissions granted for the groups.

read = 4, **write = 2**, and **execute = 1**, with the permission for each group being summed.

A file that is **755** has **rwx** for the user (r+w+x=7); and **r-x** for the group and other (r+x=5).

Permissions

On the command line in Linux, **chmod** can change **Foo** (file or folder) to have a given set of permissions. To make **Foo** have permissions 755:

```
> chmod 755 Foo
```

Alternatively, the FTP clients allow access modifications.

For most web programming applications in this course, having files/directories at the 755 level should be sufficient.

Caveats of Having a Webpage on the Server

A few things to do / be aware of...

Always **check** your file/folder **permissions** are set correctly.

There may be a **delay** of upwards of a minute or two (usually just seconds) before the server has saved your changes and you can see your new/updated webpages.

Caveats of Having a Webpage on the Server

If you are really unlucky, the FTP client may fail to transfer a file and in the process delete the file you were trying to overwrite. This usually doesn't happen unless you make a lot of consecutive updates over a small time window.

This last point is really only relevant for the PHP scripts we'll later write, but due OS differences in return carriages/new lines, if a file was edited in a Windows environment, PHP scripts may fail to compile unless the **dos2unix** command is run on them through the command line:

```
> dos2unix file_name
```

or

```
> dos2unix *
```

to update all files.

Bash Scripts

When working with Linux it can sometimes be extremely useful to write short scripts that execute a sequence of Linux commands, possibly in response to user inputs. Here, we provide an ultra-simplistic overview of a few key features in writing so called **bash scripts**. We will look at:

- setting up a bash script;
- how to run Linux commands in a script;
- passing arguments to scripts;
- variables and updating variables;
- comparison operators;
- if-statements;
- while-loops; and
- functions

Bash Scripts

Bash scripts have **.sh** extensions. After writing a script, the permissions need to be changed to give **execute** privileges.

The first line of a bash script must specify the path to the program used to interpret the script. It begins with a **#!** (she-bang). To determine this path, typing **which bash** in the command line gives the path.
hello_world.sh

```
#!/usr/local/bin/bash

echo "hello world"
```

The **echo** command prints to the console. In these examples, strings will be in double quotes.

Bash Scripts

Running the script:

```
> ./hello_world.sh
hello world
```

Bash Scripts

By enclosing a Linux command within **$(** and **)**, a linux command can be run and the output stored as a string (without new lines). Without the **$** then the command is only run.

The variables **$0**, **$1**, **$2**, ..., **$9** represent arguments passed to a script (with **$0** being the script name itself). Each argument is passed after the script name and separated by spaces.

Bash Scripts

newpage.sh

```
#!/usr/local/bin/bash
echo "Running $0"

( mkdir $1 )
( chmod 755 $1)
( touch $1/index.html )
( chmod 755 $1/index.html)
disp=$( ls $1)
echo $disp
```

Bash Scripts

Suppose we are in **public_html** and we want to make a new webpage in a new folder **Page**. Then...

```
> ./newpage.sh Page
Running ./newpage.sh
index.html
> ls
Page newpage.sh
```

could create the directory for such a page and the corresponding (empty) html file.

By using double quotes, the variables are **interpolated** and evaluated in the string. If single quotes were used, the first printout would be “Running $0".

Bash Scripts

A variable can be defined by defining a new symbol with an equals sign and a value. There **should not be a space** between the symbol defined and the equals sign. As a best practice, variables should be lowercase (to avoid updating/overriding environment variables that are uppercase).

At definition, a variable is not prefixed by a **$**. References to it later on must be prefixed by **$**. Math mode is one exception, however.

When enclosed in **((** and **))**, a variable can be interpreted and updated mathematically.

Bash Scripts

math.sh

```
#!/usr/local/bin/bash

val=0 #note the lack of space!
echo "val = $val"
((val = val+1)) # in math mode, no $, update val
echo "val = $val"
((i = val*3))
echo $i
```

Running...

```
> ./math.sh
val = 0
val = 1
3
```

A comment in bash is prefixed by **#**.

Bash Scripts

The **read** command can be used to set/update variables based on user input. With **read**, we can specify one or more variables to set. The variables are set one word at a time (spaces separate variables) except for the last variable that stores everything remaining if there are no more variables but there are more words.

reading.sh

```
#!/usr/local/bin/bash

echo "Enter something: "
read foo bar
echo "Variables are: "
echo "$foo"
echo "$bar"
```

Bash Scripts

```
> ./reading.sh
Enter something:
cats are awesome
Variables are:
cats
are awesome
```

Bash Scripts

Comparison operations can be done with bash. There are different operations for strings and integers.

For strings, we have **==** (equality), **<** (less than), **<=** (less than or equal), **>** (greater than), **>=** (greater than or equal), and **!=** (not equal) — based on ASCII table.

For integers, for the same operations, we have **-eq**, **-lt**, **-le**, **-gt**, **-ge**, and **-ne**.

Bash Scripts

The basic syntax for **if** conditionals is illustrated below:

```
if [ condition ];
then stuff_to_do
fi

if [ condition ];
then stuff_to_do
elif [ condition ];
then stuff_to_do_here
fi

if [ condition ];
then stuff_to_do
else stuff_to_do_else
fi
```

Note: There must be spaces around the square brackets!!!

Bash Scripts

We also examine a loop structure, **while.** The syntax is below:

```
while [ condition ];
do
stuff_to_do
done
```

Additionally, **-a** signifies logical and and **-o** signifies logical or, e.g.,

```
if [ $str == "cat" -o $str == "dog" ]; then ...
```

```
if [ $num -ge 0 -a $num -lt 10 ]; then ...
```

Bash Scripts

bottles.sh

```
#!/usr/local/bin/bash

num=$1 # no space!

while [ $num -gt 0 ]; do
if [ $num -eq 1 ]; then
  echo "$num bottle of Kombucha on the wall"
  echo "Take it down, pass it around"
  echo "No more Kombucha"
else
  echo "$num bottles of Kombucha on the wall"
  echo "Take one down, pass it around"
fi
((num = num-1))
done
```

Bash Scripts

```
./bottles 3
3 bottles of Kombucha on the wall
Take one down, pass it around
2 bottles of Kombucha on the wall
Take one down, pass it around
1 bottle of Kombucha on the wall
Take it down, pass it around
No more Kombucha
```

Note how each **echo** also generates a new line.

Bash Scripts

Bash scripts can also have functions. But a function cannot return a value (could set a global variable instead). Function parameters are not listed in their “signature" but the parameters are stored in **$1**, **$2**, etc.

The **function** keyword plus a function name and a set of braces define a function. Again, spacing is important with the braces.

To call a function, we write its name followed by its arguments, separated by spaces.

Bash Scripts

add_first_to_end_of_second.sh

```
#!/usr/local/bin/bash

function append_first_to_second {
  ( echo $1 >> $2  )
}

append_first_to_second $1 $2
```

Linux has its own **echo** command to display to the console. This can be rerouted to write to a file that does not exist with **>** or to append to (and possibly create a file) a file with **>>**.

Bash Scripts

```
> ls
add_first_to_end_of_second.sh
> ./add_first_to_end_of_second.sh message foo.txt
> less foo.txt
message
```

Summary

- The **internet** is a network of networks, which make use of communication between various levels of service providers.
- When a web browser displays a webpage, it is sending requests and receiving data from a web **server** and both end machines are **hosts**.
- The **TCP/IP** allows hosts to have addresses and specifies how information is sent and managed.
- Many websites are hosted by **LAMP** servers.
- Linux is a popular open source operating system.
- Some important commands include: **ls**, **pwd**, **cd**, **mkdir**, **rmdir**, **rm**, **mv**, **touch**, **scp**, and **chmod**.
- Files can have **user/group/other** access privileges for **read**, **write**, and **execute**, with either octal representation or numbers.
- In many cases (not all), a good default access level is **755**.
- A common convention is that websites are found in a **public_html** directory and **index.html** is served.

Exercises I

1. What is packet switching?
2. What is the difference between the “front end” and “back end” on a website?
3. Write a Linux command to move the file bar.txt up one directory.
4. Use Linux commands only:
    - Create a directory **Intro_Ex** in **public_html**.
    - Within that folder, create a directory **F** and an empty file **foo.txt**.
    - Within **F**, make an empty file **bar.txt**.

    Make it so that a user can navigate to **http://www.pic.ucla.edu/~your_username/Intro_Ex** and be unable to view anything. *But* if they navigate to **http://www.pic.ucla.edu/~your_username/Intro_Ex/F**, then they can see its contents (i.e. **bar.txt**) and if they navigate to **http://www.pic.ucla.edu/~your_username/Intro_Ex/foo.txt**, they can open that empty file.

Exercises II

5. Write a bash script **delete.sh** that repeatedly prompts as user for a file name to delete (one at a time) until they indicate they have no more files to delete. You may assume they enter valid file names.
6. Write a bash script **calc.sh** that allows a user to enter two numbers and an operation (in the middle) and prints the result. The operations can be **add**, **subtract**, and **multiply**. For example

```
> ./calc.sh 3 subract 8
-5
```