CS 312
Internet Concepts

Web Misc:
Internet Search and Webpage Accessibility

Dr. Michele Weigle
Department of Computer Science
Old Dominion University
mweigle@cs.odu.edu

http://www.cs.odu.edu/~mweigle/CS312-F08/

Generic Search Engines

- Some very popular ones
  - » Google
  - » Yahoo
  - » MSN
- Some common characteristics
  - » Huge set of results
  - » Extremely prompt retrieval
  - » Organized retrieval results

How do they achieve such amazing results?

How Do Search Engines Work?

- Prep work
  - » *crawling* – an automated way of browsing the web for pages
  - » archiving, analyzing, organizing, indexing
- Retrieval
  - » query matching
  - » ranking search results
  - » displaying search results

Search Query Syntax

- The space between keywords is interpreted as AND for some search engines, but as OR for others.
- Query: I'm interested in cancer in adults.
  - » Boolean logic: AND
  - » Search query: +cancer +adults
- Query: I'm interested in radiation, but not nuclear.
  - » Boolean logic: NOT
  - » Search query: radiation -nuclear

Search Query Syntax

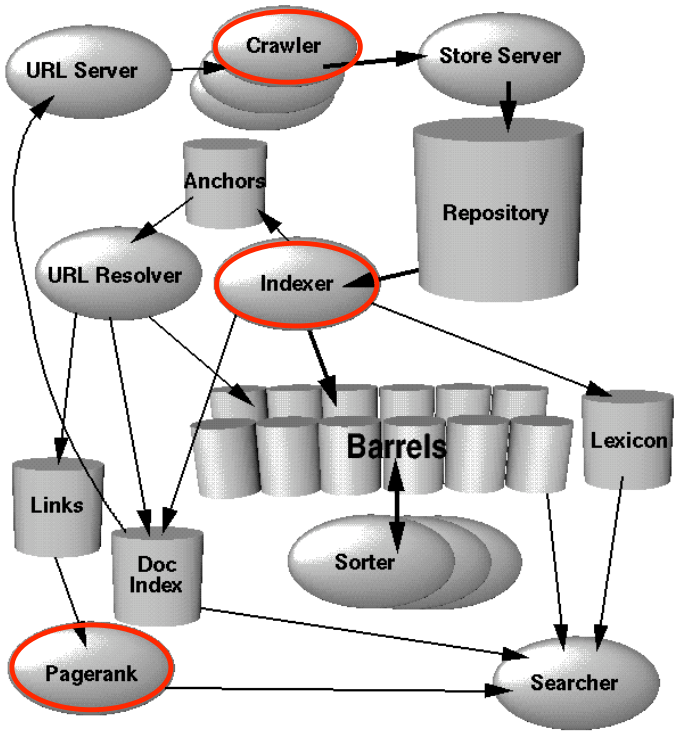
- Sophisticated query: I want to learn about cat (feline) behavior.
  - » Boolean logic: OR, AND
  - » Search query: +(cat OR feline) +behavior

Google's Three Main Components

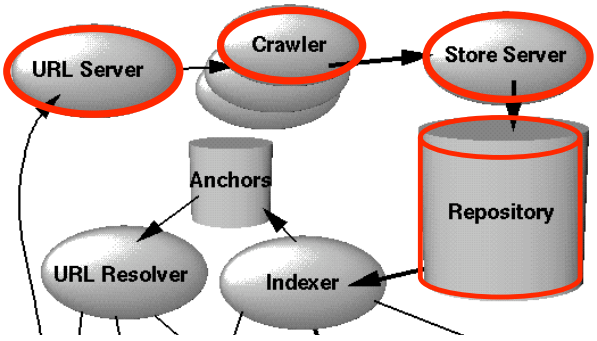
- Crawling
- Indexing
- Page Weighting

Our discussion is based on
"The Anatomy of a Large-Scale Hypertextual Web Search Engine", by Sergey Brin and Lawrence Page at http://www-db.stanford.edu/~backrub/google.html
and
"How Google Works" at http://www.googleguide.com/google_works.html

Google Architecture



Google Architecture



- Distributed servers
  - » Mostly implemented in C, C++
- The *URLserver* sends URLs to *Crawlers*, which fetch the pages.
- The *Store Server* compresses fetched pages and stores them in a *Repository*.

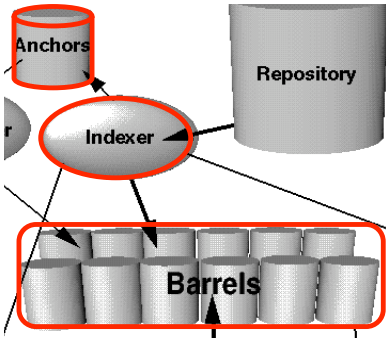
Google's Crawler, the Googlebot

- Consists of many computers requesting and fetching pages
  - » Googlebot can request thousands of different pages simultaneously
- When Googlebot fetches a page, it adds all of the links in the page to a queue for subsequent crawling (*deep crawling*)
  - » Googlebot can quickly build a list of links that can cover broad reaches of the web
- Google continuously recrawls popular frequently changing web pages at a rate roughly proportional to how often the pages change (*fresh crawls*)

Submitting URLs to Googlebot

- http://www.google.com/addurl.html
- Rejects URLs Google suspects are trying to deceive users
  - » including hidden text or links on a page
  - » stuffing a page with irrelevant words
  - » cloaking (aka bait and switch)
  - » using sneaky redirects
  - » creating doorways, domains, or sub-domains with substantially similar content
  - » sending automated queries to Google
  - » linking to bad neighbors.

Google Architecture

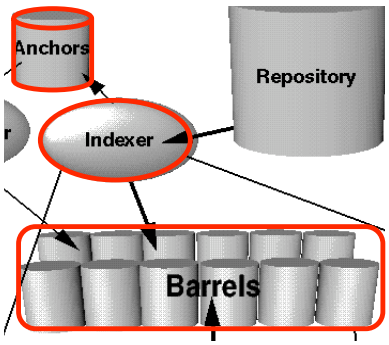


- The *Indexer* uncompresses documents, parses them, converts them into a set of hits (word occurrences).
  - » For each document in which the word appears, indexer stores the position of the word, font size, capitalization
  - » Hits are distributed into *Barrels* as a partially sorted forward index

Forward Index

Document	Words
Document 1	the,cow,says,moo
Document 2	the,cat,and,the,hat
Document 3	the,dish,ran,away,with,the,spoon

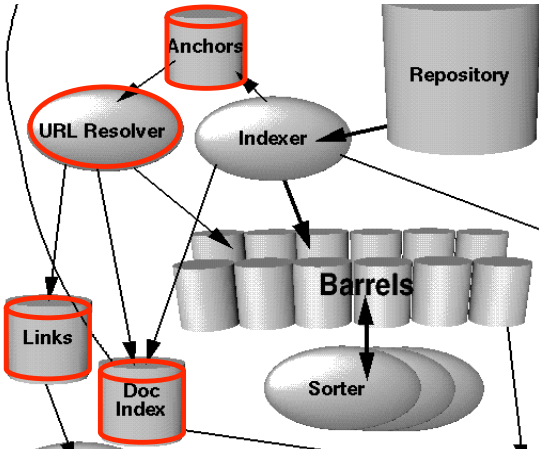
Google Architecture



- The *Indexer* parses out all the links in each web page into an *anchors* file
  - » contains enough information to determine where each link points from and to, and the text of the link (*anchor*)

Google Architecture

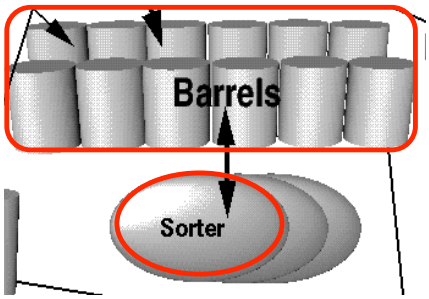
- The *URLresolver* reads the *anchors* file
  - » converts relative URLs into absolute URLs
  - » converts absolute URLs into docIDs
- Puts the anchor text into the forward index, associated with the docID that the anchor points to



- Generates a database of links, which are pairs of docIDs

Google Architecture

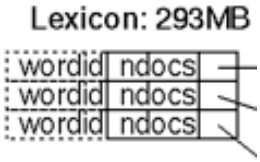
- The *Sorter* takes the *barrels* of hits, which are sorted by docID, and re-sorts them by wordID to generate the inverted index

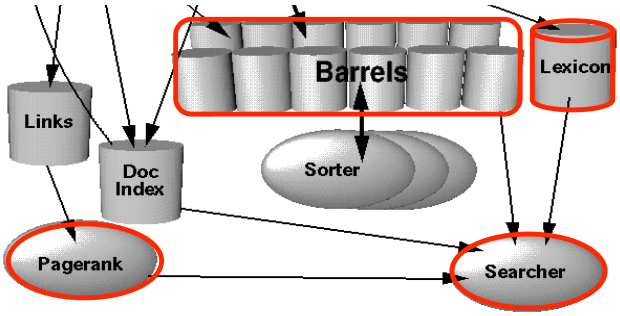


Inverted Index

Word	Documents
the	Document 1, Document 3, Document 4, Document 5
cow	Document 2, Document 3, Document 4
says	Document 5
moo	Document 7

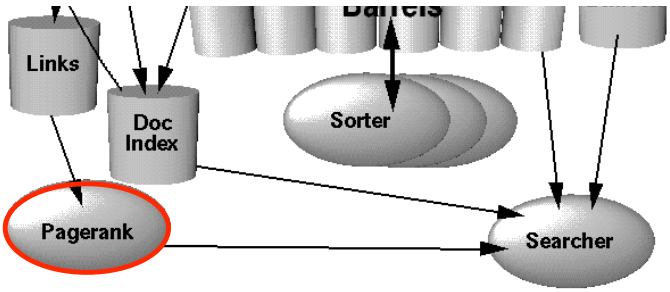
Google Architecture





- A program called DumpLexicon takes the inverted list together with the *lexicon* produced by the indexer and generates a new lexicon to be used by the *Searcher*
- The *Searcher* is run by a web server and uses the new lexicon together with the inverted index and the *PageRank* to answer queries

Google Architecture



- PageRank
  - » Assigns more importance (higher ranking) to pages that have more pages link to it
  - » Assigns heavier weights to inbound links from more important pages
- In short PageRank is a "vote", by all the other pages on the Web, about how important a page is

Google's PageRank

- The PageRank (PR) of page A is

  PR(A) = (1-d) + d * (PR(T1)/C(T1) + … + PR(Tn)/C(Tn))

  » d is a damping factor

  » T1…Tn are pages that have inbound links to A

  » C(Ti) is the number of outgoing links from Ti

- d is usually set at 0.85

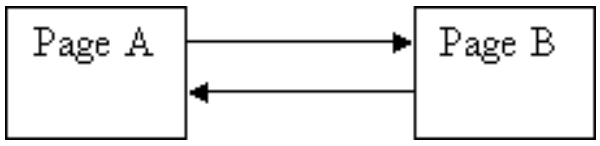
- The average PRs of all web pages should be 1

http://www.ianrogers.net/google-page-rank/

How is PageRank Calculated?

- PageRank or PR(A) can be calculated using a simple iterative algorithm.

- We'll start with a simple example



Each page has one outgoing link
(i.e. C(A) = 1 and C(B) = 1).

http://www.ianrogers.net/google-page-rank/

How is PageRank Calculated?

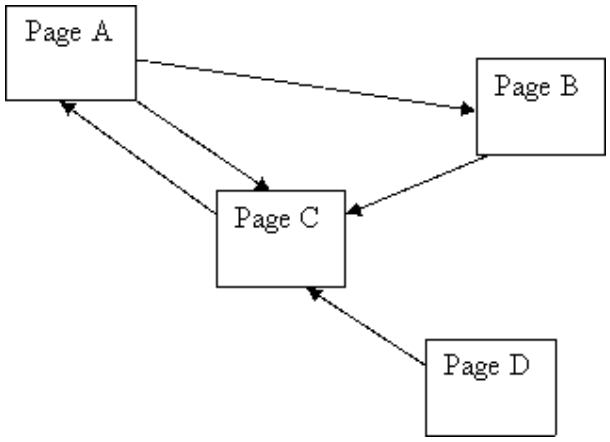
$d = 0.85$
$PR(A) = (1-d) + d(PR(T1)/C(T1) + \ldots + PR(Tn)/C(Tn))$

- Start the iteration by first setting PR(A) and PR(B) both to be 0:
  - » PR(A)= 0.15 + 0.85(0/1) = 0.15
  - » PR(B)= 0.15 + 0.85(0.15/1) = 0.2775
- Continue on the next iteration:
  - » PR(A)= 0.15 + 0.85(0.2775/1) = 0.385875
  - » PR(B)= 0.15 + 0.85(0.385875/1) = 0.47799375
- The iteration continues until the page rank values converge until the average of the PRs is 1

http://www.ianrogers.net/google-page-rank/

How is PageRank Calculated?

Here's a case with more pages



http://www.ianrogers.net/google-page-rank/

How is PageRank Calculated?

The iterative results:

a: 0.00000 b: 0.00000 c: 0.00000 d: 0.00000
a: 0.15000 b: 0.21375 c: 0.39544 d: 0.15000
a: 0.48612 b: 0.35660 c: 0.78721 d: 0.15000
a: 0.81913 b: 0.49813 c: 1.04904 d: 0.15000
a: 1.04169 b: 0.59272 c: 1.22403 d: 0.15000
a: 1.19042 b: 0.65593 c: 1.34097 d: 0.15000
….
….
a: 1.49011 b: 0.78329 c: 1.57660 d: 0.15000
a: 1.49011 b: 0.78330 c: 1.57660 d: 0.15000
a: 1.49011 b: 0.78330 c: 1.57660 d: 0.15000
Average pagerank = 1.0000

The calculations always converge.

http://www.ianrogers.net/google-page-rank/

Manipulating PageRank / Webspam

- Google bomb
  - » is created if many sites link to the page using the same anchor text (increases the page rank)
  - » ex: In 1999, a search for "more evil than Satan himself" resulted in the Microsoft homepage
- spamdexing
  - » deliberately modifying HTML pages to increase the chance of their being placed close to the beginning of search engine results
- link doping
  - » embedding a large number of gratuitous hyperlinks on a website, in exchange for reciprocal links
- Google Jacking (page hijacking)
  - » creating a rogue copy of a popular website which shows contents similar to the original to a web crawler, but redirects web surfers to unrelated or malicious websites.

Recording Internet History

- Internet Archive
  - » non-profit that was founded to build an Internet library, with the purpose of offering permanent access for researchers, historians, and scholars to historical collections that exist in digital format
  - » http://www.archive.org
- Internet Archive's Wayback Machine
  - » allows you to browse through 85 billion web pages archived from 1996 to a few months ago
  - » http://www.archive.org/web/web.php

Web Page Accessibility

- Are all pages posted on the web necessarily available to all?
- What about dynamically generated pages?
- Are there some ways to help enhance accessibility?
- Is it difficult to be able to restrict access on certain pages?

Enhancing Your Page's Popularity

- Submit your page to search engines
  - » http://www.google.com/addurl.html
- 12 Things to Do to Improve Your PageRank
  - » http://stason.org/articles/money/seo/google/12_things_to_do_to_improve_your_site_google_page_rank.html
- 12 Things **Not** to Do to Improve Your Page Rank
  - » http://stason.org/articles/money/seo/google/12_things_not_to_do_to_improve_your_site_google_page_rank.html

Restricting Accessibility

- Robots Exclusion Protocol
  - » used to give instructions about a site to web robots (like Googlebot)
  - » instructions are in placed in `/robots.txt` at the root of a website
    - ❖ need to be the website owner to use this
- Robots can ignore your `/robots.txt`
  - » malware robots - scan the web for security vulnerabilities
  - » email address harvesters used by spammers

http://www.robotstxt.org/

Examples of /robots.txt

Against indexing the entire site

```
User-agent: *
Disallow: /
```

*User-agent: **
means all unspecified robots.

Against indexing selected folders or files

```
User-agent: *
Disallow: /~mweigle/
Disallow: /~mweigle/files/foo.txt
```


Examples of /robots.txt

To allow only specific robots

```
User-agent: googlebot
Disallow:
User-agent: yahoobot
Disallow:
User-agent: *
Disallow: /
```

Does not disallow anything for googlebot or yahoobot.

Disallows all to all other robots.

What Can a Webpage Author Do?

- Use robots <meta> tag
  - » prevent individual pages from indexing

<meta name="robots" content="*tag1, tag2*">

- Possible *tags*
  - » noindex – don't index this page
  - » nofollow – don't scan this page for links to crawl

- Default is index, follow

http://www.robotstxt.org/

Example robots meta tags

Add in the <head> section one of the tags :

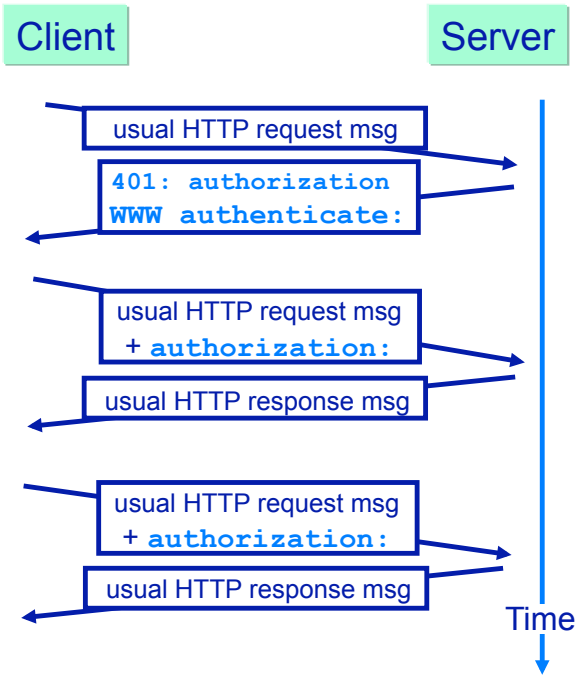
```
<meta name="robots" content="index, follow">
<meta name="robots" content="index, nofollow">
<meta name="robots" content="noindex, follow">
<meta name="robots" content="noindex, nofollow">
```

Accessibility with Robot Specifications

- Will all robots follow the instructions?
  - » not the bad guys
- Can pages still can be accessed when not indexed?
  - » if they're not protected
- Is there any way available to prevent public access but still allow protected access?
  - » yes!

To Prevent Public Access

- Basic authentication
  - » allows web browsers to provide credentials in the form of a user name and password when making a request
  - » protection is per-directory, not per-file
  - » the requesting client must authenticate itself to the server with a user-ID and a password for each directory being protected



More on Basic Authentication

- Assumes that the connection between the client and server computers is secure and trusted
  - » all information sent in plain-text
- Could use HTTPS for encryption

Basic Authentication and .htaccess

- Special file named `.htaccess`
  - » must be publicly readable (chmod 644)
  - » dot is part of the filename
- States where a corresponding encrypted file (generated in some special way) for this authentication is located
- Only recommended if you don't have access to main server configuration files
  - » i.e., you don't own the webserver

.htaccess Operation

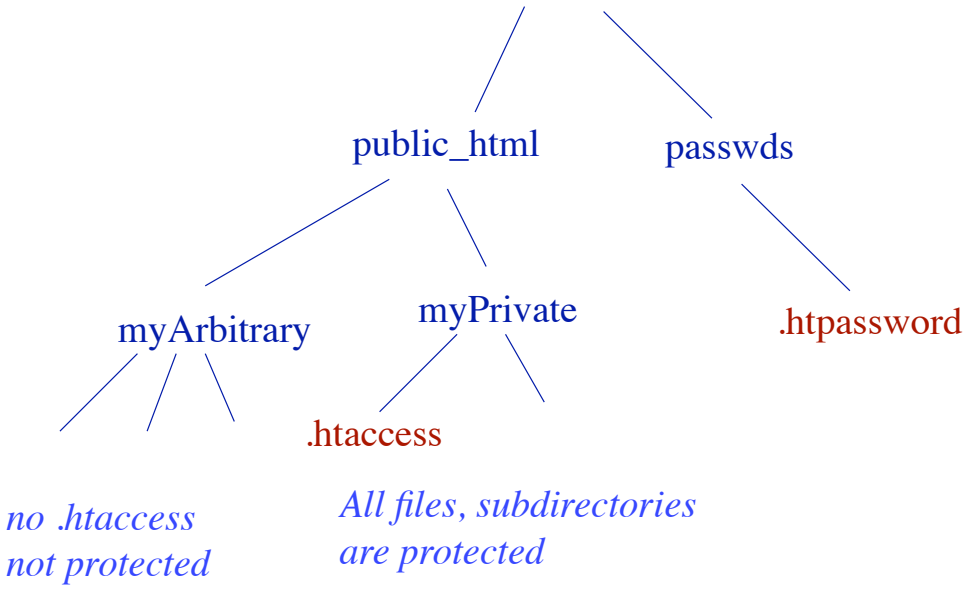
- When a HTTP server tries to access a directory, it first looks in the directory for this file
- If this file does not exist at the directory then the server considers that this directory is not protected and accesses the files in it normally.
- Otherwise, the server reads this file to find out where the encrypted file is
- Server reads the encrypted file and uses the information there to verify authentication

.htaccess Operation

- Before accessing any files in this directory, the server checks the request's authentication header.
- When the authentication is satisfied by the request information, then the server delivers the requested file in this directory to the requesting client.
- If not satisfied, then sends code 401 (unauthorized).

.htaccess Operation

- Once the challenge is satisfied, the username and password pair is saved by the browser
  - » allows automatically sending authentication header in later entries into the same directory during the session.
- Since both .htaccess and the corresponding encrypted file are to be read by the server remotely, they both need to have the access permission r for the world (chmod 644)
- The encrypted file should be located in some publicly readable, unprotected directory, not necessary under public_html
  - » directory should be chmod 755



Example .htaccess File

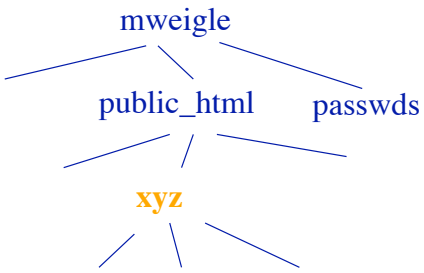
```
AuthType Basic
AuthName "Password Required"
AuthUserFile /home/mweigle/passwds/.htpassword
```


Special Notes on Implementation

- As long as you have a file named .htaccess in a directory
  - » that directory is treated by the web server as protected
  - » this fact does not depend on the existence of the corresponding encrypted file or not
- If you want to list files with a dot in the front of the file names
  - » the corresponding Unix list command must have the parameter –a (ex: ls –a)
  - » otherwise, you will not see such files

Example

- We want to protect all files in the directory
  `/home/mweigle/public_html/xyz/`
- Only allow user John with password x123y



Example
Generate the Encrypted File

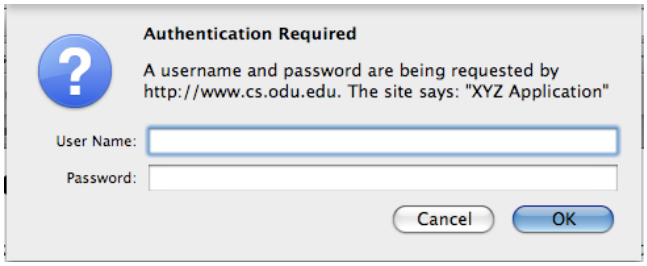
- Run the encryption program htpasswd (no dot in the front) as a Unix command
  /usr/apache/bin/htpasswd -c ~/passwds/.htpasswdxyz John
  - » the parameter -c means to create a new encrypted file.
  - » .htpasswdxyz is the name of the file to be generated
  - » John is the username
- The program will ask for the password twice

```
% /usr/apache/bin/htpasswd -c ~/passwds/.htpasswdxyz John
New password:
Re-type new password:
Adding password for user John
```

Example
Generate the .htaccess File

- Create a Unix text file named .htaccess, stored in ~/public_html/xyz/

```
AuthType Basic
AuthUserFile /home/mweigle/passwds/.htpasswdxyz
AuthName “XYZ Application”
Require valid-user
```



http://www.cs.odu.edu/~mweigle/xyz/

Example
File Permissions

- ~/public_html/xyz/
  - » chmod 755
- ~/public_html/xyz/.htaccess
  - » chmod 644
- ~/passwds/
  - » chmod 755
- ~/passwds/.htpasswdxyz
  - » chmod 644

More About .htaccess

- Require
  - » allow any valid user (in the encrypted file)

    Require valid-user
  - » specify which users can have access

    Require user John

More info:
http://httpd.apache.org/docs/2.0/howto/auth.html#basic

Multiple Users

- Re-run htpasswd once for each new name
- Make sure not to use –c in the command so that the encryptions for the old pairs will not be cleared.
- The program will again ask for password twice for each new username you give in a command.

.htaccess and Access Control

- Access Control (apart from authentication)
  - » control who can and can't view the page
  - » use 'Allow' and 'Deny' directives
    - Allow from *{hostname, IP address, domain name}*
    - Deny from *{hostname, IP address, domain name,* all*}*
  - » use 'Order' directive to specify the order that they should be applied (usually deny before allow)

```
Order Deny,Allow
Deny from all
Allow from .cs.odu.edu
```

only allows browsers from .cs.odu.edu domain to view the page or authenticate

http://www.cs.odu.edu/~mweigle/xyz/

Safety of Basic Authentication

- Only as good as the username and password
- If not using HTTPS, usernames and passwords are sent in plain-text