UNIX

An Introduction To the Command Line

Why do we need to learn about Unix?

- Our Web server, *Gibson*, and over half of all Web servers in the world, run on a variant of the Unix operating system.
- Learning how to do basic file and directory management on Unix systems is necessary if you ever want to move on to more advanced Web development (like right now).
- Sometimes, tasks are are quicker and easier to accomplish on the command-line than with a GUI.
- Unix and its variants run under the hood on many Netbooks and desktop computers, as well as OS X and iPhone.

A Little Unix History

- Unix is a computer operating system developed by Ken Thompson, Dennis Ritchie, Brian Kernighan and others in the late 1960's
- In 1973, it was rewritten in the C programming language and ported to multiple hardware platforms
- Unix originally ran on "time-shared" mainframe computers using teletypes and video display terminals for access
- Unix emphasizes writing small, portable programs that perform a single task well

A Little More Unix History

- Unix and "Unix-like" systems are generally very stable
- A great deal of Unix support and development is done by the user community
- Unix is able to run effectively on less powerful hardware than a Windows server
- LAMP servers (Linux, Apache, MySQL, and PHP) can be downloaded and installed from multiple resources for free under the GNU license structure

So, Where Do I Click?

- *Usually, you don't!*
- "Classic" Unix operates using a command-line interface (CLI)
- Users connect to Unix machines using Telnet or SSH
- Once connected, users type short, terse commands like *pwd*, *cp*, *ls*, etc.
- Everything is case sensitive
- Guess what?
  - Mac OS X (under the hood) and Linux are both variants of Unix
  - So are iOS and Android OS and Chrome OS

Why Are Unix Commands So Terse?

- Remember, Unix was developed to work with teletypes and video display terminals – very slow by today's standards
- No "processing" happened at the user's workstation
- Every command had to be entered by hand, sent to the mainframe computer, and returned
- Unix commands were designed to minimize the amount of typing / bandwidth needed
- Unix commands were developed for programmers – experienced computer users who could "handle" terse commands

Connecting to gibson.rit.edu via the command line

- On a Mac
  - Fire up the *Terminal* application
  - Type: `ssh abc1234@banjo.rit.edu`
  - Type: your `password`
- On a PC
  - Fire up *PuTTY*
  - Connect to banjo.rit.edu using SSH (Secure Shell)

The "Root" Of All Unix

- One of the interesting aspects of Unix is that *everything* is treated like a file
  - Storage Devices
  - Printers
  - Scanners
- As a result, *everything* in Unix is a *descendant* of one directory: *the Root*
- The Root Directory in Unix is represented by: /

The "Path" to Success

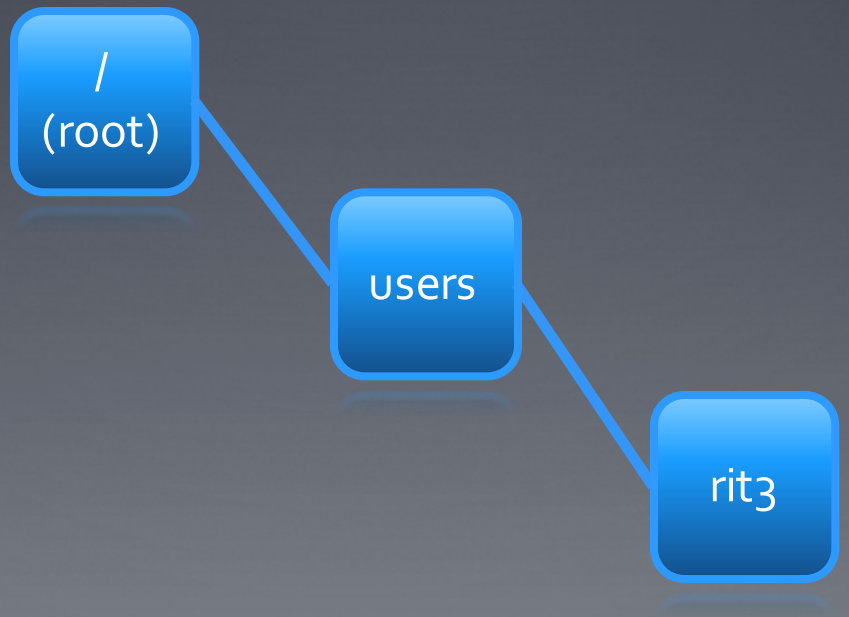
- Paths are used to identify a specific file or location (directory) in the Unix system
- Quick review:
  - Absolute Paths
  - Relative Paths

Why Use Relative Paths?

- Easier and less error-prone
- No need to specify the complete path
- Not bound to a *specific* file system/server
- You can mimic the directory structure on your local system
  - Use identical relative paths
  - Port the entire site to the server without recoding
- Most of the time you will use Relative Paths

An Example

- Here's an example: `/users/rit3`
- This represents the path from the *root* directory to the *rit3* directory



There's No Place Like Home

- When you log into Unix, you are automatically placed in your *login* directory (this is often called your "home" directory)
- Your login directory is your workspace on the Unix server
  - This is where you can create files, sub-directories, etc
- On Gibson, your login directory name is your *user id*
- Your web files should be in the *www* directory

R·I·T

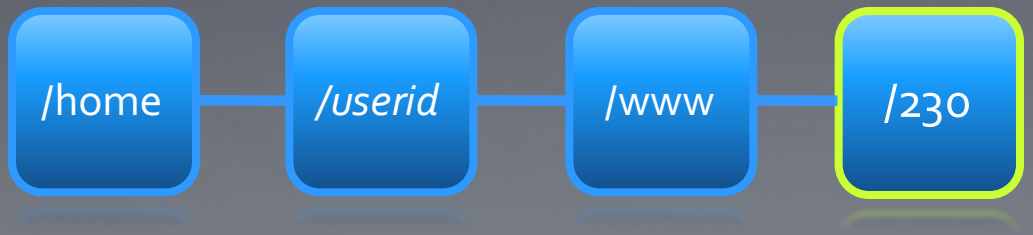
The Current Working Directory

- When working in Unix, you are *always* in a directory
  - This directory is called the *Current Working Directory*
  - This is the directory upon which any commands you issue will be executed
  - There are commands that allow you to change your current working directory
- When you first sign on, your *login* directory is your Current Working Directory

/home/userid
Your Current Working Directory at sign on

Where Am I?

- Unix commands allow us to change our current working directory easily
- Sometimes, you may lose track of where you are in the directory structure
- The *pwd* command (print working directory) will show you
- pwd tells you where you are, starting with root



pwd would return:
/home/userid/www/230

R·I·T

What's in my Current Working Directory?

- The *ls* command *l*i*s*ts the contents of your current working directory
- Displays only the names of the files and directories
- There are *many* options for this command – type *man ls* at the Unix prompt for more detail

```
$ ls
230             Hello737.php    pfw
330             Old             phpInfo2.php
```

List with Detail

- The *ls –l* command generates a "long" listing
  - a "d" in the first column indicates a directory
  - a "-" in the first column indicates a file
  - The next nine columns show the permissions for this item

```
ls -l
total 10
drwxr-xr-x   2 msrvks   faculty       96 Dec 10  2009 iweb
drwxr-xr-x   8 msrvks   faculty     1024 Jan 25  2010 737
-rwxr-xr-x   1 msrvks   faculty      342 Jan 25  2010 Hello737.php
drwxr-xr-x   2 msrvks   faculty       96 Nov 28  2009 Old
drwxr--r-x  11 msrvks   faculty     1024 Apr 16 23:01 pfw
-rwxr-xr-x   1 msrvks   faculty      329 Jan 25  2010 phpInfo2.php
drwxr-xr-x  11 msrvks   faculty     1024 Apr 26 17:32 testing
```

Moving Around

- The *cd* command allows us to change our current directory
  - To change to a directory, you must have permission to do so
  - There are three (3) ways to use the cd command:
    - cd *path* – changes to the directory specified in the path
      - This can be an absolute or relative path
    - *cd* – With no argument, changes to your login directory
    - *cd /* – Changes to your root directory

How Do I Create A New Directory?

- The *mkdir* command (*m*a*k*e *dir*ectory) allows you to create a new directory within your current working directory.
  - You must have write permission on the current working directory to use this command
  - Example: *mkdir project1* creates a directory called *project1* in the current working directory

```
$pwd
/home/acjvks/www/230
$ ls -l
total 0
-rw-------   1 msrvks   faculty        0 Sep 26 16:11 file1.txt
-rw-------   1 msrvks   faculty        0 Sep 26 16:12 file2.txt
-rw-------   1 msrvks   faculty        0 Sep 26 16:12 file3.txt
$ mkdir project1
$ ls -l
total 0
-rw-------   1 msrvks   faculty        0 Sep 26 16:11 file1.txt
-rw-------   1 msrvks   faculty        0 Sep 26 16:12 file2.txt
-rw-------   1 msrvks   faculty        0 Sep 26 16:12 file3.txt
drwx------   2 msrvks   faculty       96 Sep 26 16:13 project1
```

How Can I Delete a Directory?

- Deleting a directory in your current working directory is as easy as creating a new directory.
  - Use the *rmdir* (*rem*ove *dir*ectory) command
  - You must have write access to the current working directory
  - For example, to remove our newly created project1 directory, we would issue the command: *rmdir project1*

```
$ls -l
total 0
-rw-------   1 msrvks   faculty        0 Sep 26 16:11 file1.txt
-rw-------   1 msrvks   faculty        0 Sep 26 16:12 file2.txt
-rw-------   1 msrvks   faculty        0 Sep 26 16:12 file3.txt
drwx------   2 msrvks   faculty       96 Sep 26 16:13 project1
$ rmdir project1
$ ls -l
total 0
-rw-------   1 msrvks   faculty        0 Sep 26 16:11 file1.txt
-rw-------   1 msrvks   faculty        0 Sep 26 16:12 file2.txt
-rw-------   1 msrvks   faculty        0 Sep 26 16:12 file3.txt
```

Absolute vs. Relative Paths

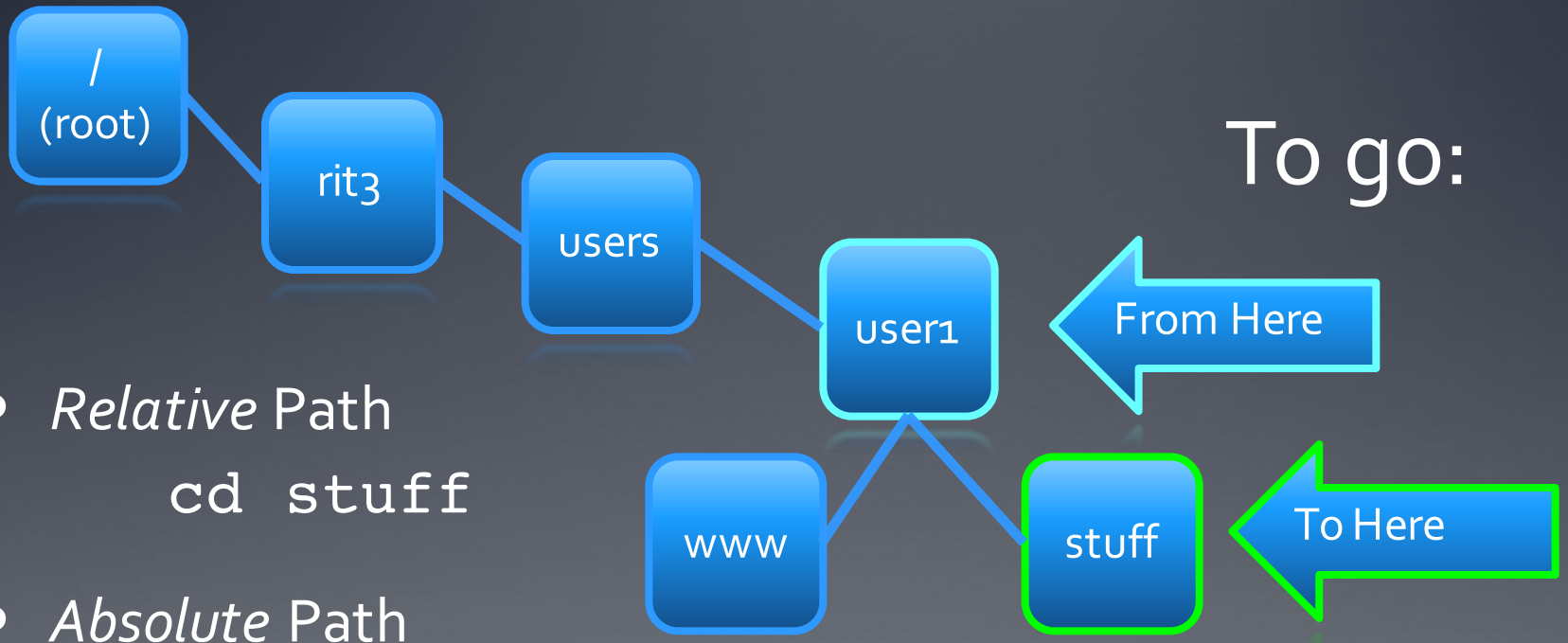
- Absolute Path
  - Full path from root to target
  - Example (prj1.css)

```
/home/userid/www/230/prj1/prj1.css
```

- Relative Path
  - Path from current working directory to target

```
www/230/prj1/prj1.css
```

Absolute vs. Relative Paths Example 1



To go:

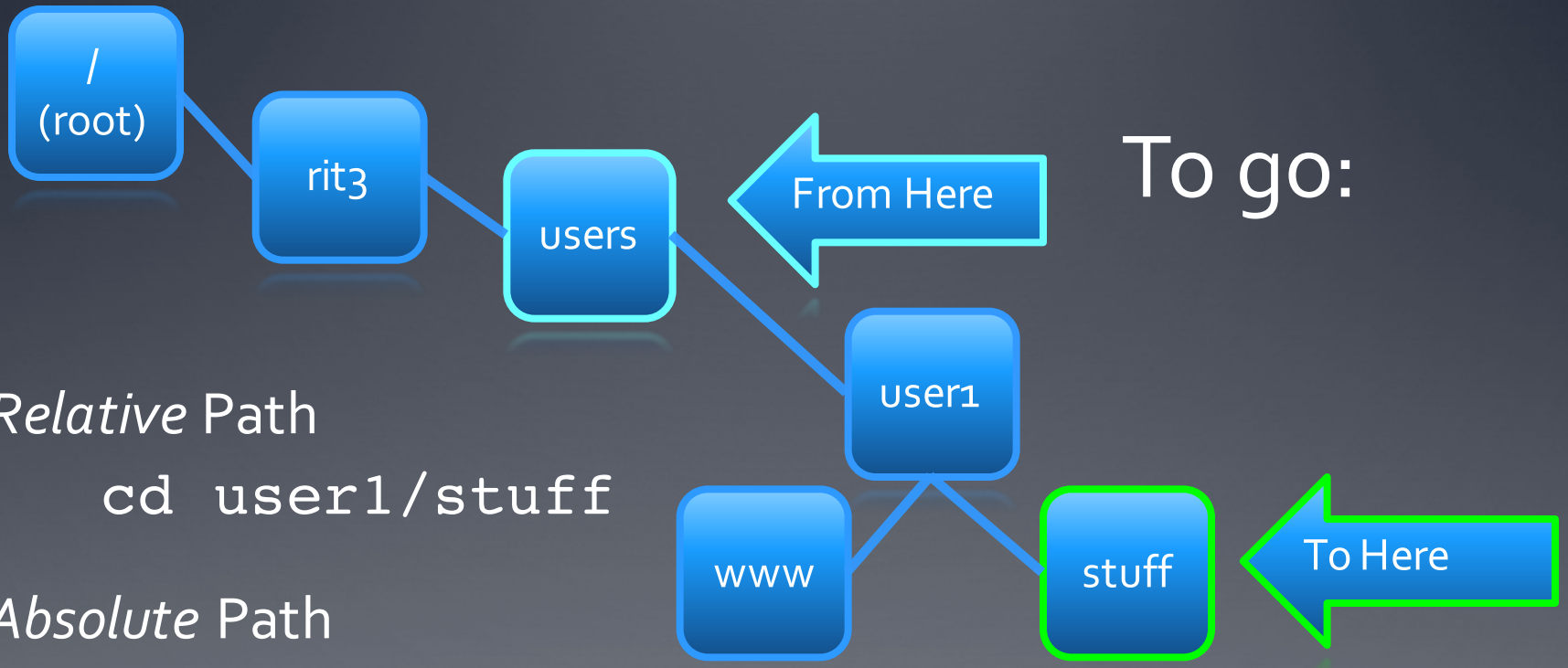
- *Relative* Path

```
cd stuff
```

- *Absolute* Path

```
cd /rit3/users/user1/stuff
```

Absolute vs. Relative Paths Example 2



- *Relative* Path

```
cd user1/stuff
```

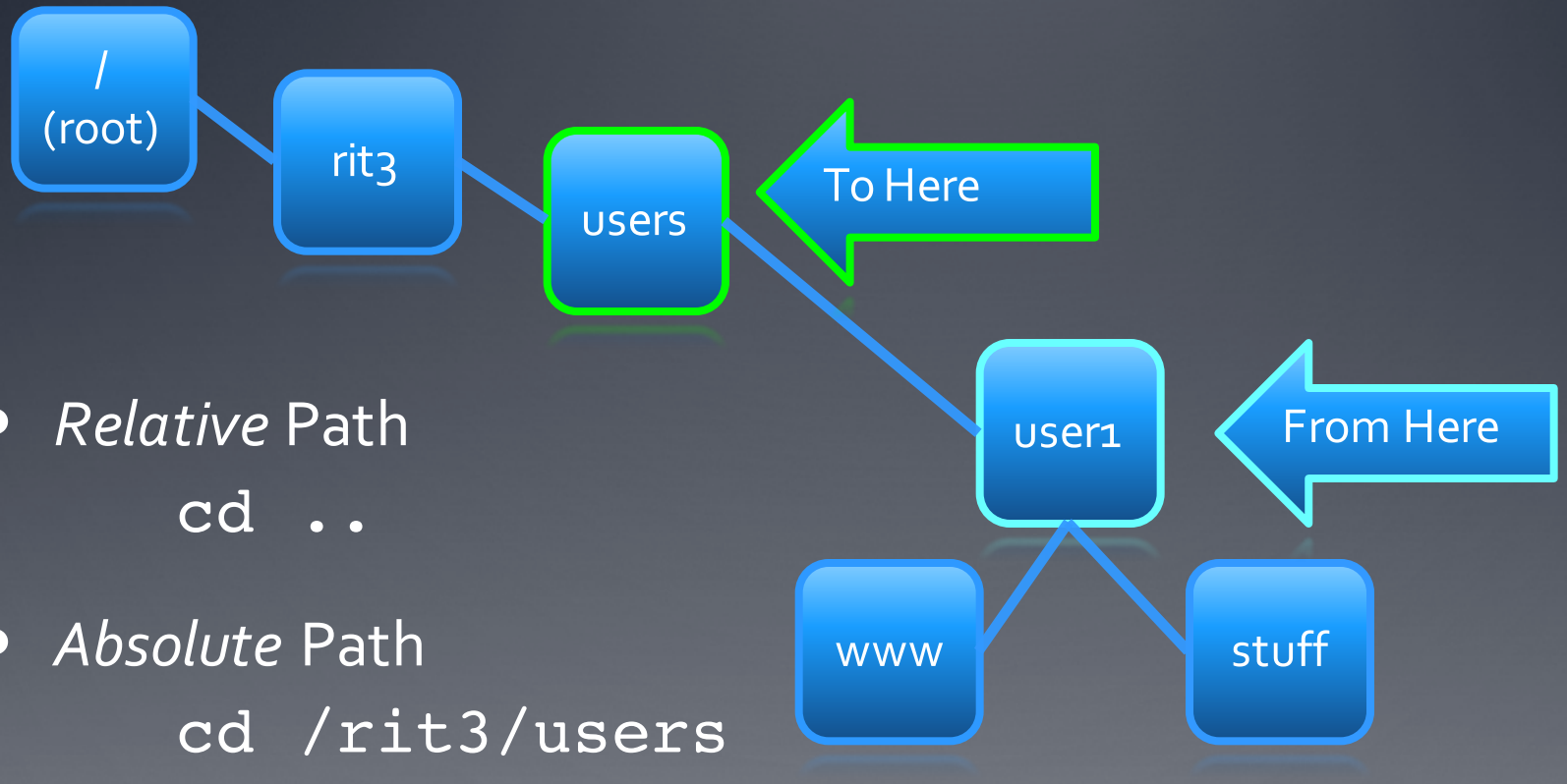
- *Absolute* Path

```
cd /rit3/users/user1/stuff
```

The "Parent" Directory

- Every directory has only *one* parent directory
- The parent directory is the directory that is next "upwards" (closer to the root) from the current directory
- In path specifications, we can use the characters ".." to represent a parent directory

Absolute vs. Relative Paths Moving Up One Level



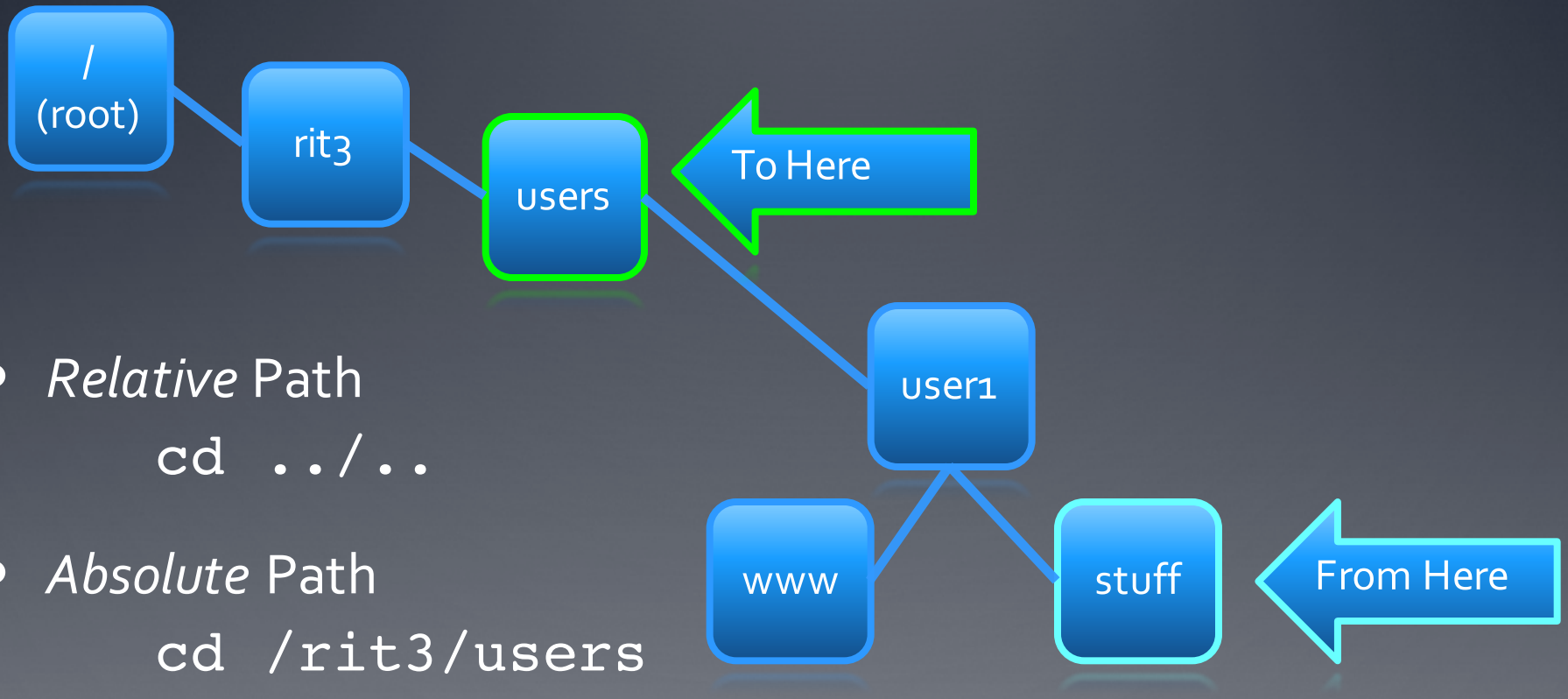
- *Relative* Path

```
cd ..
```

- *Absolute* Path

```
cd /rit3/users
```

Absolute vs. Relative Paths Moving Up Multiple Levels



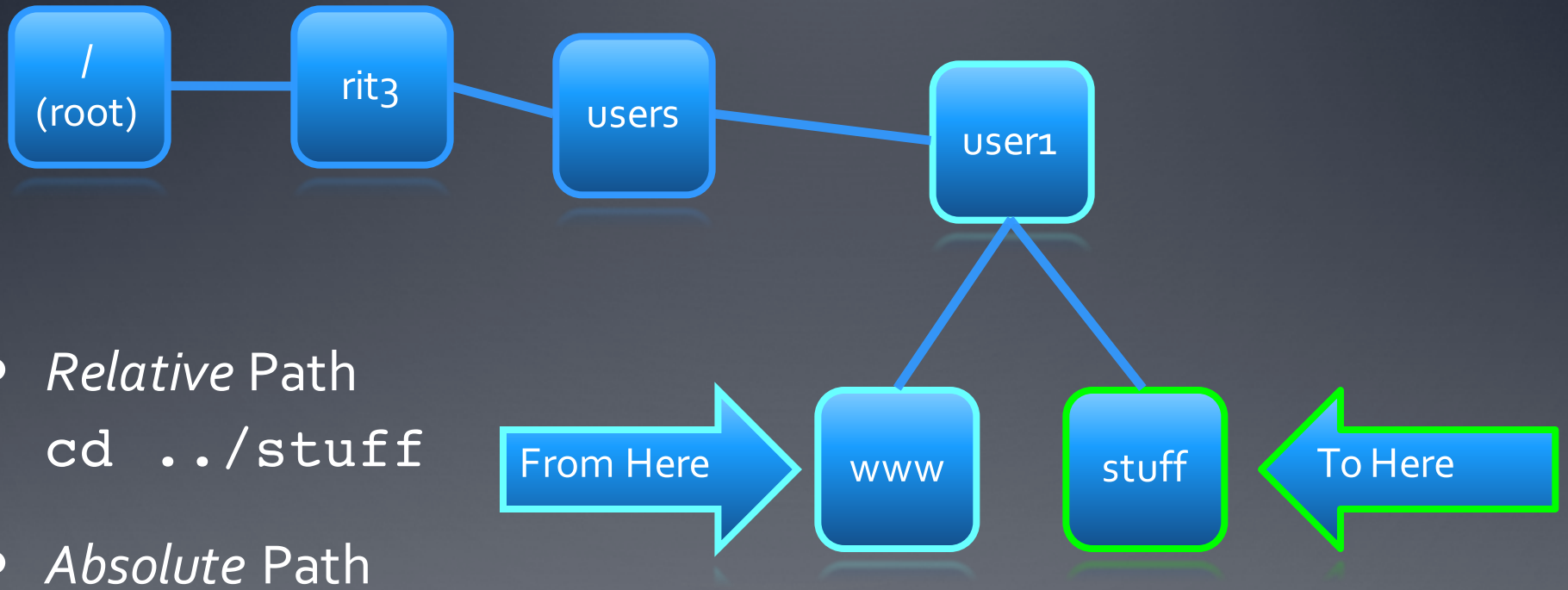
- *Relative* Path

```
cd ../..
```

- *Absolute* Path

```
cd /rit3/users
```

Absolute vs. Relative Paths Moving To Adjacent Locations



- *Relative* Path
  `cd ../stuff`
- *Absolute* Path
  `cd /rit3/users/user1/stuff`

Back To Some Unix Commands
Copying, Moving, & Renaming Files

Copying Files

- Format: *cp file1 file2*
- If *file2* already exists, it will be overwritten
- if *file2* does not exist, it will be created

Moving Files

- Format: *mv file1 file2*
- renames *file1* to *file2*
- *file2* may include a path
  - *mv file1 ../directory/file2*
  - The ".." moves up one level

Do More With Less
Using Wildcard Characters

- Used by Unix for simple pattern matches
- The *asterisk "*"*
  - Can represent *zero or more* characters in a file name
  - Can be placed anywhere in the file name
  - * *alone* means match everything
  - What would the command *rm *.html* do?
- The *question mark "?"*
  - Represents a *single* character in a file name
  - What would *rm Picture??.png* do?

Deleting Files

- Format: *rm filename*
- BE CAREFUL!!!!!
- There is no "Undo" command or trash from which you can get your deleted file
- Once the command executes, your file is gone forever!

Wildcards and Deleting

- Always be *very* careful when using a wildcard in an *rm* command
- Examples:
  - rm *.com – remove all files with names ending in *.com*
  - *rm proj?.doc* – remove all files in the form proj?.doc where the ? indicates any *single* character
  - *rm –r media* – can delete a folder *and* its contents in one step
- Caution: Be very careful using *rm **
  - Why?

Unix File Permissions

What does all this mean?

```
% ls -l

total 46
-rwx------   1 acjvks   faculty     2765 Apr 12  1999 #.mrg...login
drwx------   2 acjvks   faculty     8192 Feb 18  2000 bin
drwxr-xr-x   5 acjvks   faculty     8192 Sep  3 16:18 files
drwx------   2 acjvks   faculty     8192 Sep 14 14:26 mail
drwx------   3 acjvks   faculty     8192 Sep 21 14:52 mydir
drwxr-xr-x   5 acjvks   faculty     8192 Sep 20 19:00 www
```

Permissions & Directory Flag

User and Group

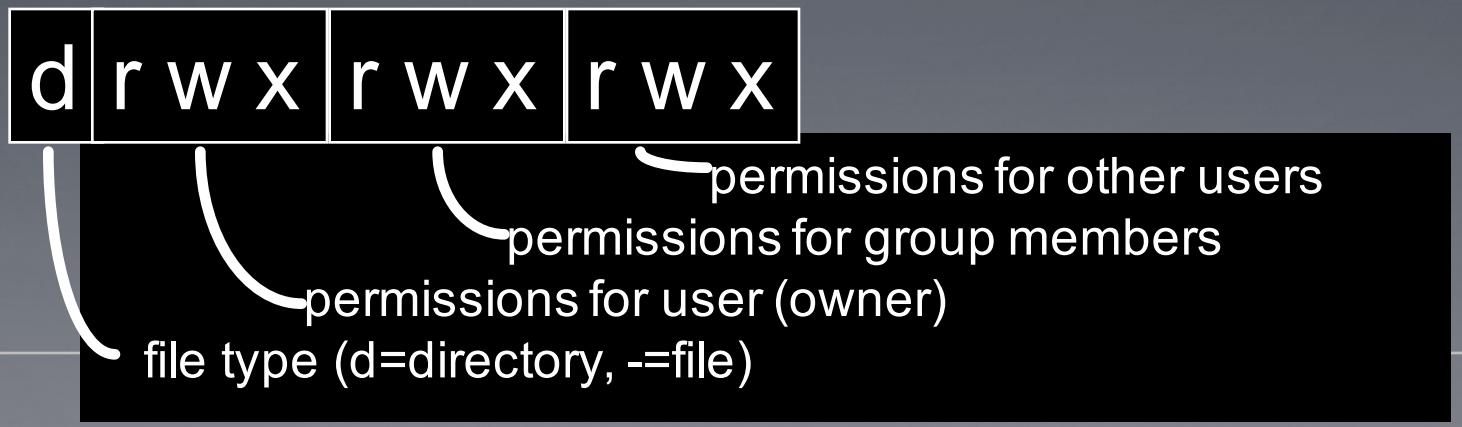
File Size

Date & Time Last Modified

File Name

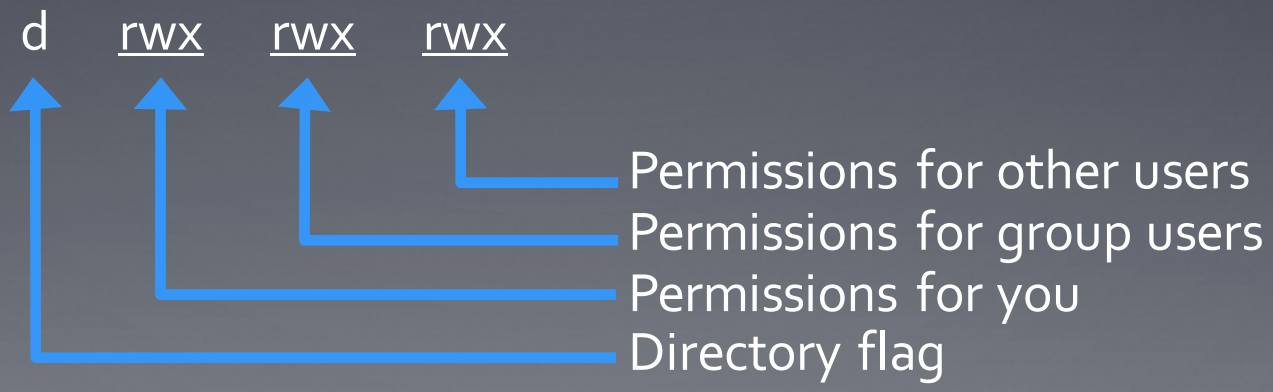
Permissions and Directories

- Three categories of ownership
  - user, group, others
- Three options for protection
  - read, write, execute



The Permissions Column

- The Permissions column of the directory listing provides the user with two important pieces of information
  - Whether this entry in the listing represents a file or directory
  - What permissions have been assigned to this file or directory
- The Permissions Column consists of 10 characters (spaces added for clarity):



The "d" column

d rwx rwx rwx

- Indicates if this row in the listing represents a directory or a file
- A "d" in this position indicates that this row represents a *directory*
- A "-" in this position indicates that this row represents a *file*
- A "#" in this row represents a symbolic link – we will not discuss symbolic links in this course

The Permissions Groups

d *rwx* *rwx* *rwx*

- Permissions are represented by three, three-character sets.
- The three groups define the *level* of the permissions
  - Group 1: *Owner*
  - Group 2: *Group*
  - Group 3: *Others*
- Each group has three characters:
  - *r* represents read access for this level
  - *w* represents write access for this level
  - *x* represents execution permission for this level

Directory and File Permissions

- Permissions have different meanings depending on if you are looking at a file or a directory:
  - Read permission:
    - **File**: read and copy the file
    - **Directory**: list the contents of the directory
  - Write permission:
    - **File:** modify and delete the file
    - **Directory**: rename and delete files
  - Execute permission:
    - **File:** run the program (if it is one)
    - **Directory:** access to the directory

Using the chmod Command

- *chmod 744 file.txt*
- Modifies the permissions for the file: *file.txt*
- The permissions are:
  - User has read, write and execute permission

    rwx = 4+2+1
  - Group has read permission only

    r = 4
  - Others have read permission only

    r = 4

Important Permissions

- Directory settings
  - 755 allows browsers to enter and list directories
  - rwx r-x r-x
- File settings
  - 644 allows browsers to read files (web pages)
  - rw- r-- r--
- Web servers are "other" users on remote servers
  - File permissions = 4
  - Directory permission = 5

Gibson web site permissions

- Your *login account directory* should have drwx--x--x (711 permissions)
  - This means that users (web servers and others) may open your login directory, but not view the contents.
- Your *www* folder has drwxr-xr-x (755 permissions)
  - This means that users may open the directory, and list its contents.
  - Your *iweb* directory should have the same permissions
- Your iwebhome page (*index.html* in your iweb folder) has -rw-r--r-- (644 permissions)
  - This means users may read the file (they don't need to execute it)