Overview

- Internet basics (protocols, servers, ...)
- Unix command line tools for networking (ssh, scp, etc)
- File transfer in Ruby
- Hypertext transfer in Ruby
- Web services in Ruby
- Black diamond: making your own service

The Network Interface

- A typical laptop or desktop has several I/O connections
  - A look at the hardware shows the heart of the system consists of a processor, main memory (RAM), and a communication controller that manages data transfers

Networks

- In a small local area network machines often talk directly to each other
  - Use ethernet cables to connect each machine to a switch
  - Switches are often placed in a closet, with cables running to offices
Networks

- The only difference between the network on the previous page and a small local wireless network is the physical connection
- A wireless hub acts as the switch for communication between local machines

IP Addresses

- Each machine in a network has a unique identifier
- The most widely used standard (IPv4) is to assign an IP address to each machine
- With IPv4, an address is a 32-bit number
- Usually written in “dotted decimal” notation as four numbers between 0 and 255 separated by periods

PowerBook G4
Switch (aka Hub)
Local Systems

A larger network uses routers to pass information from one local subnetwork to another
- A router is like a switch, except it has an additional WAN (wide area network) connection
- Most wireless base stations are routers

One of the main purposes of a router is to assign dynamic IP addresses
- Some machines have a permanent address, e.g., fugu, in my office in Deschutes, has an address assigned by the CIS systems staff
- When fintan (my laptop) is at home, it gets an IP address from the wireless router, which in turn gets its address from Comcast
- When fintan is in my office, I connect via ethernet and get a new dynamic address from a CIS department router

IP Addresses

- 10.0.0.1
- 10.0.0.7
- 10.0.0.2

IP Addresses

- 128.223.6.18
- 128.223.234.150
- 10.0.0.12

128.223.6.18
fugu
128.223.234.150
fintan
10.0.0.12
fintan
Comcast
Domain Name System

- It is rarely necessary for a programmer or user to learn a system's IP address
- The domain name system (DNS) has names for systems with permanent IP addresses
  - names are single words separated by periods
  - each name maps to a unique IP address
    - fugu.cs.uoregon.edu => 128.223.6.18
    - shell.uoregon.edu => 128.223.142.32
    - apple.com => 17.149.160.49
    - ftp.ncbi.nlm.nih.gov => 165.112.7.10
- Note: There is no simple way to convert parts of a symbolic name to numbers in the dotted decimal notation
  - most system names do not have exactly four words
  - There is a pattern
    - machines in.edu are 128.x.x.x
    - machines in uoregon.edu are 128.223.x.x

Domain Name System

- Names are assigned hierarchically
  - if a new computer is installed in the CIS department, the system staff assigns the number (128.223.6.x)
  - if a new department or institute is started on campus, it gets a range of numbers from the computer center (128.223.x.x)
- Systems with dynamic IP addresses also have names
  - at home my laptop is fintan.local
  - at work it has a name like dyna6-207.cs.uoregon.edu
- But as the term implies, dynamic names are changing all the time, so don’t expect to be able to refer to a machine by it’s dynamic name

Ports

- Operating systems use an abstraction known as a port to manage network communication
  - ports are implemented in software, inside the OS
  - when a message arrives on the physical connection, it includes a port number
  - the OS passes the message on to software that is "listening" on the port

- Examples of ports
  - 20: FTP
  - 22: SSH
  - 25: Mail

Protocols

- Computer networking is based on the idea of a communication protocol
- A protocol defines the formats of strings and the order in which they are sent
- Example: UDP (Universal Data Protocol)
  - used to send small packets of data
  - a message starts with the IP address of the sender (32 bits = 4 characters)
  - these characters are followed by the IP address of the receiver
  - there are fields to define the protocol being used, the port numbers, data length, etc
- An application can send data to an application on another system (assuming the remote application is listening on the designated port) by composing a message with this structure and “printing” it on its own output port

<table>
<thead>
<tr>
<th>src</th>
<th>dest</th>
<th>U</th>
<th>sp</th>
<th>dp</th>
<th>n</th>
<th>chk</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 32 bits (4 chars)
Most internet communication is organized around **clients and servers**
- a client initiates a conversation by sending a message to a server
- the client knows the server's IP address and the port the server uses
- example: the web server at teleost.cs.uoregon.edu (128.223.8.115) is listening on port 80

The server is a program that is always running
- many servers are started by the system administrator when the OS starts (e.g. mail servers, web servers)
- users can start their own server software (e.g. database server)
- may need to check first to see a port isn’t already being used

### Shell Commands

**ssh**
- start a shell on a remote system
  ```
  [fintan:conery] % ssh fugu.cs.uoregon.edu
  Last login: Sun Feb 24 18:57:47 2008 from hsd1.or.comcast.net
  [fugu:conery] % hostname
  fugu.cs.uoregon.edu
  [fugu:conery] % ls /Volumes/BigDisk/
  Backup/  Books/  OmniFocus/
  [fugu:conery] % logout
  Connection to fugu.cs.uoregon.edu closed.
  ```
- additional arguments are passed as a command to the remote shell
  ```
  [fintan:conery] % ssh fugu.cs.uoregon.edu ls /Volumes/Bigdisk
  Backup
  Books
  OmniFocus
  ```

**scp**
- like the standard cp (copy files) program, but copies to/from a remote system
  ```
  100% 75KB 74.8KB/s  00:00
  [fintan:conery] % cd Classes/199/programs/
  [fintan:programs] % scp hello.rb fugu.cs.uoregon.edu:
  hello.rb
  100% 42 0.0KB/s  00:00
  [fintan:programs] % ssh fugu.cs.uoregon.edu ls -l
  ...-rwxr-xr-x  1 conery staff  42 Feb 24 22:04 hello.rb
  ```
Security

- The "s" in `scp` and `ssh` stands for "secure"
- The first time you connect to a remote host that machine sends a unique key
  - the key is stored in your home directory, in `.ssh/known_hosts`
- When you run `ssh` or `scp`, every line transmitted to the remote host is encrypted with that host's key
- Encryption is very important: you don't want bad guys to be monitoring the traffic between your local system and the remote host
- especially when you type a password, enter a credit card number, ...

Security

- In order to login or run a command on a remote system with `ssh` you need to have an account on that system
- Usually the remote system will prompt you for a password when you try to connect:
  ```bash
  [fintan@conery]
  more .ssh/known_hosts
  fugu.cs.uoregon.edu,128.223.6.18 ssh-rsa
  AAAAB3NzaC1yc2EAAAABIwAAAIEAlAUIaoZohrfJIAXHyBYPjgKED25h6V9Q...
  ix.cs.uoregon.edu,128.223.6.41 ssh-rsa
  AAAAB3NzaC1yc2EAAAABIwAAAIEA3XufbRvfvex4maNoj3DD0t3Fa8Syz5y9y...
  ...
  ```
- You can use a program named `ssh-keygen` to make a pair of encryption keys
  - one is your **private key** -- it is kept on your local system
  - install the other key -- the **public key** -- on the remote system
- When you run `ssh` or `scp`
  - outgoing messages are encrypted with your private key
  - the remote system sees where the message is coming from, and if it has a public key for that system, uses it to decode the message
  - if decryption succeeds the command is executed, without prompting for a password
  - very useful for scripts that automate remote commands and file transfers (e.g. `nci`)

Networks and Ruby

- Ruby scripts that access the internet use one of two general strategies
- One strategy is to run Unix commands that transfer data or carry out remote operations
  - example from my `nci` script:
  ```bash
  [dyna6-207:conery] % ls Notebooks
  Checked In/ Classes.scriv/ OmniFocus.ofocus/
  [dyna6-207:conery] % nci Classes
  tar zcf Classes.scriv.tgz Classes.scriv
  scp Classes.scriv.tgz fugu.cs.uoregon.edu:/Users/conery/Notebooks
  Classes.scriv.tgz 100% 187KB 187.2KB/s 00:01
  ssh fugu.cs.uoregon.edu 'cd /Users/conery/Notebooks; tar zxf Classes.scriv.tgz'
  ...
  ```
- The Ruby code that runs a remote command:
  ```ruby
  def remote(cmd)
    rcmd = "cd #{NotebookDirectory}; #{cmd}"
    puts "ssh #{Server} `#{rcmd}`"
    system("ssh #{Server} `#{rcmd}`") or abort("failed: #{cmd}")
  end

def checkin(filename)
  tarfile = filename + ".tgz"
  scp Classes.scriv.tgz #{Server}:#{NotebookDirectory}
  ssh #{Server} 'cd #{NotebookDirectory};
  tar zxf Classes.scriv.tgz'
  ...
  ```
Using `system` to run a shell command is fine for simple operations. But in other scripts we need a `Ruby program to be a client` to interact with a server:
- connect to a web server to get a page, use regular expressions to extract desired fields
- connect to an FTP server to download selected files
- connect to a mail server to fetch one or more messages

The basic technique for this sort of operation is the same as using the File class for I/O:
- instead of calling `File.open` to make a `File object`, call a constructor to make a `server object`
- call methods that interact with the server to upload or download data
- close the connection

Example, using FTP (file transfer protocol):
```ruby
f = Net::FTP.new('ftp.ncbi.nih.gov')
f.login
a = f.list
f.close
```

The File Transfer Protocol is what is known as an application layer protocol:
- IP, TCP, UDP, etc are lower level protocols for sending single packets or messages
- FTP, HTTP, and other application protocols require a client and server to exchange a series of messages to send data from one machine to another

Before showing how to write a script that uses FTP we'll see some examples of how a user runs an FTP client:
- Most of the time you'll want to use a GUI application
  - at right: a window from Vicomsoft FTP for OS/X

The FTP protocol predates graphical user interfaces:
- in the old days, users ran a command line application to make a connection and transfer files

The old program was named `ftp`:
- a newer version, `sftp`, is similar, but does encrypted communications
- when you first connect to a server it may take some exploration to see if it supports (or even requires) secure communication
- use trial and error or read the server documentation (e.g. from a web page at the remote site)

Note: the same advice (explore and experiment) also applies to GUI clients

To start the program, just type its name:
```bash
[fugiconery] % ftp
```
Command Line Application for FTP

- **ftp** was designed to be an interactive program
  - It prints a prompt and waits for you to type a command
    
    
    ftp> help
    Commands may be abbreviated. Commands are:
    $ features mls prompt site
    ... 
    - The first thing you need to do is establish a connection with the open command
      - Type `open` followed by the name of the system you want to connect to
        
        ftp> open ftp.ncbi.nih.gov
        ... 
        220 FTP Server ready.

Aside: Anonymous Login

- When you make a connection, **ftp** will prompt you for your login name and password on the server
  - The string between parens is a default value; if you type return, this value is used
    
    Name (ftp.ncbi.nih.gov:conery):
    
    - Public file servers, such as the one at NCBI, support anonymous logins
      - Anyone is allowed to download data from the server
      - The convention for making an anonymous connection: use `anonymous` as the login name, and use your e-mail address as your password:
        
        Name (ftp.ncbi.nih.gov:conery): anonymous
        331 Anonymous login ok, send your complete email address as your password.
        Password: **************
        230 Anonymous access granted, restrictions apply.

Command Line Application for FTP

- Other useful **ftp** commands:
  - `ls` list the files in a directory on the server (remote system)
  - `cd` change to directory X on the server
  - `get` download the file named X

- Below is an example session with the server at NCBI (edited to fit on the slides)

  Name (ftp.ncbi.nih.gov:conery): anonymous
  Password: **************
  230 Anonymous access granted, restrictions apply.
  ftp> ls
  dr-xr-xr-x  8 ftp anonymous  4096 Sep 29 2004 blast
  dr-xr-xr-x  3 ftp anonymous  4096 Sep 13 2004 cgap
  ...
Command Line Application for FTP

```
ftp>
```

```
ftp> cd Escherichia_coli_K12
```

```
250 CWD command successful
```

```
ftp> ls
```

```
200 PORT command successful
```

```
150 Opening ASCII mode data connection for file list
```

```
-r--r--r--   1 ftp    1076622 Feb 20 15:41 NC_000913.GeneMark-2.5f
-r--r--r--   1 ftp     267932 Feb 20 15:41 NC_000913.GeneMarkHMM-2.6m
-r--r--r--   1 ftp     179108 Feb 20 15:41 NC_000913.Glimmer3
-r--r--r--   1 ftp   20221984 Dec 26 16:44 NC_000913.asn
-r--r--r--   1 ftp       2131 Jun 12  2006 NC_000913.gbk
-r--r--r--   1 ftp    8377468 Feb 19 20:58 NC_000913.gff
```

```
Files with names ending .gbk are “genbank files”
These are text files containing the annotation of a gene, chromosome, or complete chromosome
```

```
ftp>
```

```
ftp> get NC_000913.gbk
```

```
local: NC_000913.gbk remote: NC_000913.gbk
```

```
200 PORT command successful
```

```
150 Opening BINARY mode data connection for NC_000913.gbk (13099426 bytes)
```

```
100% |****************************| 12792 KB    1.51 MB/s    00:00 ETA
```

```
226 Transfer complete.
```

```
ftp>
```

```
quit
```

```
221 Goodbye.
```

```
You can supply a second argument to get, and it will be used as the local name of the downloaded file:
```

```
>ftp get NC_000913.gbk ecoli.gbk
```

```
ftp will transfer files in binary mode or text mode
```

```
In binary mode files are copied exactly as they are, bit for bit, with no changes
+ use this mode for programs, compressed tar files (.tar or .tgz or .tar.Z) etc
```

```
In text mode ftp translates formats
+ e.g. lines on Windows systems end with two bytes (CR/LF), but lines on Unix systems end with one byte (newline)
+ ftp will convert these end of line markers so text files are downloaded properly
```

```
To switch operating modes use the ascii or binary command
```

```
Note: when you type a get command ftp tells you which mode it is using:
```

```
ftp> get NC_000913.gbk
```

```
local: NC_000913.gbk remote: NC_000913.gbk
```

```
200 PORT command successful
```

```
150 Opening BINARY mode data connection for NC_000913.gbk
```

```
There is one other option you might need to deal with
```

```
ftp can use either active mode or passive mode
+ the difference is in whether the server uses a second port for control signals (e.g. to interrupt a long transfer)
```

```
If you type a command like ls in passive mode, but the server expects to operate in active mode (or vice versa), weird things will happen
+ symptom: it takes a looong time for the server to respond
```

```
Workaround:
+ type ^C to break the current command
+ type active to tell your local system to use active mode transfers and try again
+ type passive to tell the local system to do passive mode transfers
+ record the results of this experiment in your notebook so the next time you connect to this server you can enter the correct mode
```
Ruby FTP Library

- Unlike String, Array, File, Dir, and other standard classes, the FTP class is part of a library
- libraries are distributed with Ruby
- the classes and methods in a library are always available, you just have to load them first
- When a program uses a library module, the convention is to put a require command at the front of the program

```ruby
#!/usr/bin/env ruby
# This program downloads a genome file from NCBI
require 'net/ftp'
```

- The argument to require is a string specifying the name of a library
- Documentation (e.g. the Thomas book) should show the name to use in a require statement
  - Note: require and load are similar -- load (re)loads a file every time it is called, but require makes sure a library is loaded just once

Aside: Name Confusion

- We now have three different things that are named “FTP”
  - an internet protocol (along with UDP, HTTP, SMTP, and a zillion others)
  - a Unix command line program
  - a Ruby class
- To keep them straight, I will use the following conventions:
  - the protocol will be written in plain font and all caps: FTP
  - the command line program will be written in fixed width font and lower case: ftp
  - the Ruby class will include the module name: Net::FTP
- Net::FTP, the full name of the class, using the module name, is used in Ruby programs
  - there are other other classes in the Net module, e.g. Net::HTTP

Net::FTP Methods

- The two main ideas to remember about using the Net::FTP class:
  - to have your program carry on a conversation with another machine create an object to represent the server on the other machine
  - methods of the object correspond to the commands of the ftp program (ls, cd, etc)
- To make a Net::FTP object call new:
  ```ruby
  f = Net::FTP.new
  #=> #<Net::FTP:0x46dc0 @resume=false, @debug_mode=false, @mon_entering_queue=[], @mon_count=0, @passive=false, @mon_owner=nil, @binary=true, @mon_waiting_queue=[]>
  ```
- Notes:
  - the string printed by this assignment shows the state of the server; you can ignore this for now
  - there is a Net::FTP method for most ftp commands, but the method name might be different (e.g. chdir instead of cd)

Net::FTP Methods

- To interact with a server, call the connect method, passing the name of the server
  ```ruby
  f.connect("ftp.ncbi.nih.gov")
  #=> nil
  ```
- Note: even though the return value is nil, the connection was made
  - if the connection fails, the connect method will raise an error
    ```ruby
    f.connect("ix.cs.uoregon.edu")
    Net::FTPTempError: 421 Our ftp server is ftp.cs.uoregon.edu
    from /opt/local/lib/ruby/1.8/net/ftp.rb:241:in `getresp'
    ...
    ```
- To terminate the session call the close method
  ```ruby
  f.close
  ```
Like the File class, Net::FTP has an open method that can be passed a block. open combines the new, connect, and close operations into one neat package.

Example: a session with the FTP server at NCBI might look like this:

```ruby
Net::FTP.open("ftp.ncbi.nih.gov") do |f|
  ...
  <call methods of f to list directories at NCBI, get files, ...>
  ...
end
```

Once a connection is made, call the login method to gain access:

- the two parameters to the method are strings containing a username and password
- this can be a problem if you're writing a script!
- it's very bad form to include passwords as plain text in a program
- figure out some way to read a password from a private file, or set up a public key authentication scheme on the remote system
- You can also call login with no parameters, in which case the method will try to do an anonymous login
  ```ruby
  f.login
  => "230 Anonymous access granted, restrictions apply.\n"
  ```
- The result of this call (and others) is the response from the server
- as with connect, the method raises an error if the username/password combination is invalid or if anonymous logins are not allowed.

Net::FTP Methods

- **ls**
  - return an array with one item for each entry in the current directory on the server
  ```ruby
  f.ls
  => ["drwxr-xr-x  30 ftp      www           512 Jan 16 22:24 pub"]
  ```
  - note these are "long" listings, what you get when you type "ls -l" in a shell
  - the first column shows the access modes; directory names start with "d"
  - the name of the entry is the last item in the string

- **chdir**
  - change to directory s on the server
  ```ruby
  f.chdir("pub")
  => nil
  ```
The following example is a summary of how to use Net::FTP to carry out a conversation with NCBI:

The goal is to download all the .gbk files for a species of bacteria:

```
% download.rb Escherichia_coli_K12
% download.rb Agrobacterium_tumefaciens_C58_Cereon
```

A little bit of exploration at NCBI's FTP host reveals:

- There is a top-level directory named genomes.
- Under that directory are directories for model organisms and major groups of organisms.
- In the directory named bacteria there are directories for individual species.

```
ftp> cd genomes/Bacteria
ftp> ls A*
```

The actual program, of course, has lots of comments.....

```ruby
#!/usr/bin/env ruby
require 'net/ftp'
def fetch(species)
  Net::FTP.open('ftp.ncbi.nih.gov') do |ftp|
    ftp.login
    ftp.chdir("genomes/Bacteria/#{species}")
    ftp.ls.grep(/\.gbk/).each do |entry|
      file = entry.split[-1]
      puts "#{file}...
      ftp.get(file)
    end
  end
end
species = ARGV.shift or abort "Usage: download.rb species"
fetch(species)
```

What happens if the user types the wrong species name?

```
% download.rb E.coli
```

```
/opt/local/lib/ruby/1.8/net/ftp.rb:243:in `getresp': 550 genomes/Bacteria/E.coli: No such file or directory
```

A runtime error occurred:

- The string after the : is the error message returned by the FTP server.
- The first line says the error was in a method named getresp.
- The second line says getresp was called from voidcmd.
- Yadda yadda
- Yadda yadda
- The last line (finally) says the problem was caused by a call in line 23 of our program

Yecch. There is no reason end users need to see this mess.
The solution to this problem: **catch the exceptions**

In C++, Java, and other languages the terminology is:
- when a method detects an error it calls a system-level function named `throw`
- the calling program uses a construct named `catch` to process the exception

In Ruby the construct is named `rescue`

```ruby
begin
  fetch(species)
rescue Exception => msg
  puts "#{$0}: #{msg}"
end
```

This expression means: the error is represented by an object of the `Exception` class; store this object in a variable named `msg` and execute the following block of code.

When our user runs the new version of the program:

```
% download.rb E_coli
```

```text
download.rb: 550 genomes/Bacteria/E_coli: No such file or directory
```

Much better!
Aside: Ruby Gems

- Learn about a program named gem, which manages these libraries (known as “Ruby Gems”)
- To get a list of currently installed gems:
  ```bash
  % gem list
  *** LOCAL GEMS ***
  actionmailer (2.0.2, 1.3.3)
  actionpack (2.0.2, 1.13.6, 1.13.3)
  actionwebservices (1.2.6, 1.2.3)
  ...
  ```
- To download and install a gem:
  ```bash
  % gem install ddate
  Successfully installed ddate-1.0.0
  1 gem installed
  Installing ri documentation for ddate-1.0.0...
  Installing RDoc documentation for ddate-1.0.0...
  ```

Other libraries

- mail
- xml
- ??

Aside: Ruby Gems

- RubyForge is a public repository of libraries contributed by the user community

Web Services

- soap
- rest
- examples