SSH For Fun and Profit

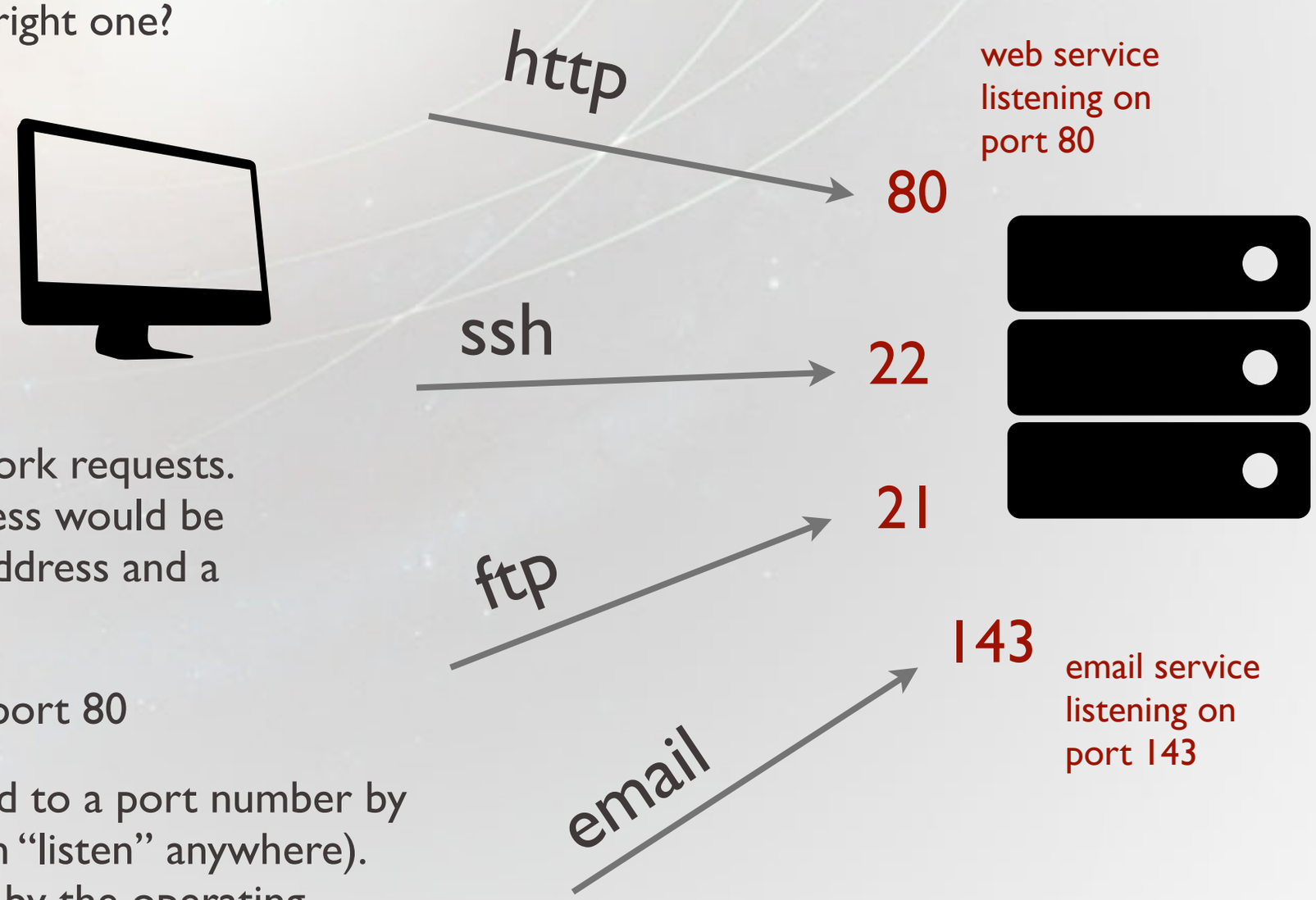
or,

Far More Than You Ever Wanted to Know About SSH

Demitri Muna

But First... Any Port in a Storm

Each request to a server is handled by a different program ("service"). How is each request sent to the right one?



A *port* is location for network requests. A complete network address would be the combination of an IP address and a port number, e.g.

104.196.243.211 port 80

Most protocols are mapped to a port number by default (but any service can "listen" anywhere). Ports 0-1023 are reserved by the operating system, any value up to 65535 is allowed.

Server icon by andriwidodo from the Noun Project.
Computer icon by Vladimir Belochkin from the Noun Project

Secure Shell (SSH)

SSH allows you to connect to a remote machine. You probably already use this, e.g.

```
% ssh muna@trillianverse.org
```

(And then you enter your password.)

Problems:

- You connect to several different servers.
- You have a different username on different servers.
- It's not easy to remember different passwords.
- You're using passwords.
- For security purposes, some machines require you to connect on different ports.
- You may have to connect to an intermediate server, then again to your server.

(Who knew there were so many problems?)

SSH Config File

The SSH configuration file allows you to bookmark each server you connect to and customize the connection parameters for each one. Create a file called "`config`" in a directory called "`.ssh`" in your home directory. It *must* have restricted permissions to work – only read/writable by your user:

```
% mkdir .ssh
% chmod 700 .ssh
% cd .ssh
% touch config
% chmod 600 config
```

Open the config file and enter the details of an SSH server you connect to:

```
Host trillian
    HostName trillianverse.org
    User demitri
    Port 22
```

full host name

if using the default port 22, this is optional

Now, you can connect to the server with the bookmarked name:

```
% ssh trillian
```

SSH Autocomplete

"UGH!" I hear you cry. "I don't have time to type in that *whole* bookmark name!" (Who does!?)

Create a new file in your home directory called `.autocomplete.sh` with this magic:

```
# SSH
function _ssh_completion() {
egrep -o '^Host [a-zA-Z]+' $HOME/.ssh/config | awk '{ print $2 }'
}
complete -W "$(_ssh_completion)" ssh
```

In your `.bashrc` (or `.bash_profile` on macOS), add this line:

```
source $HOME/.autocomplete.sh
```

Open a new terminal window, type `ssh` and the first letter or two of your server bookmark name, then hit tab to complete the name. You're now 27% more productive.

Intermediate Server

Sometimes you have to connect to one externally-facing server first, then ssh again into the server you want, e.g.:

```
% ssh -p 2222 muna@gatekeeper.zuul.io
Last login: Fri Jul 21 11:19:06 2017 from 123.45.67.8
gatekeeper% ssh demitri@danasfridge.zuul.io
Last login: Fri Jul 21 12:54:15 2017 from 89.76.54.1
danasfridge%
```

note different ports and usernames!

In your `.ssh/config` file, create a bookmark for both servers:

```
Host gatekeeper
    HostName gatekeeper.zuul.io
    User muna
    Port 2222

Host fridge
    HostName danasfridge.zuul.io
    User demitri
    ProxyCommand ssh -q gatekeeper -W %h:%p
```

For the internal server, add a `ProxyCommand` directive using the bookmark name of the external server.

To connect straight to the internal server, simply enter:

```
% ssh fridge
```

(or if you're on a deadline, "ssh fr<tab>"...)

scp, Too

If you're connecting to a server behind a gateway, have you ever copied files from your computer to the gateway, then again to the internal server you want? The bookmarks work with `scp`:

```
% scp proton_pack.pdf fridge:
```

You can use the bookmark name with nearly every program that uses SSH, and the program will read the connection details from your config file.

X11 Forwarding

Do you run X11 programs from the server on your desktop?
Normally to enable this you connect with:

```
% ssh -X trillian
```

Add this command to your server entry to enable X11 forwarding:

```
ForwardX11 yes
```

SSH Keys

Some servers are configured to accept encryption keys in addition to (or instead of!) requiring a password. This can be more secure since the account cannot be compromised by someone guessing passwords.

SSH keys are comprised of a *public* and a *private* key. The public key can be given to anyone (hence the name). If you connect to a server that has your public key and you can provide your private key, it will let you in. (Consequently, if your private key is stolen, someone else can log into your account!)

Generating SSH Keys

```
% cd .ssh
% ssh-keygen -t rsa -b 4096
% Generating public/private rsa key pair.
Enter file in which to save the key (/Users/demitri/.ssh/id_rsa): id_rsa_4096
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
% ls
id_rsa_4096
id_rsa_4096.pub
% chmod 600 id_rsa*
```

encryption algorithm

key strength - 4096 is unbreakable

private key

public key

If you enter a passphrase, you will need to type that password every time you use the ssh keys (e.g. when connecting to a server). It's common to not create a password, but know that if the private key is lost, anyone can use them. (But they would have to know which server to connect to, which the "config" file will provide.)

SSH Key Authentication

You need to do two things to authenticate with a server using SSH keys.
(This is assuming the server supports this, which would be unusual if not.)

1. Update your .ssh/config file entry for the host.

```
Host trillian
    HostName trillianverse.org
    User demitri
    IdentityFile ~/.ssh/id_rsa_4096
    IdentitiesOnly yes
```

private key name

do not attempt to ask for a password (SSH keys only)

2. Put the *public* key on the remote server.

Open this file on the remote server you are connecting to (create it if it doesn't exist):

```
% mkdir .ssh
% chmod 700 .ssh
% cd .ssh
% touch authorized_keys
% chmod 700 authorized_keys
% vi authorized_keys
```

Open the public key in a text editor and copy it. Paste it on one line in the `authorized_keys` file. You can add multiple keys (for different people connecting to the account), one on each line. Now you can connect without a password:

```
% ssh trillian
```

How Paranoid Are You?

By default, when you connect to a server over SSH, your side says:

```
I have three SSH keys:
    - Do you accept public key 1? No?
    - Do you accept public key 2? No?
    - Do you accept public key 3? Yes?
```

the server now has three of your public keys

Then the server encrypts a message with your public key and you return a response based on your private key to validate that it's you.

It is not insecure for anyone to get your *public* key. Someone who is collecting SSH keys from servers can use that collection of SSH keys to "fingerprint" you. This is less of a concern on your department servers, but more when you use SSH keys on services like GitHub where they are open to the public. (For example, my GitHub SSH keys are visible here: `https://github.com/demitri.keys`.) It's easy to collect all public SSH keys on GitHub. Then, if someone can get you to *try* to ssh into their server, they will know who you are.

You can generate an SSH key pair for every server (service) you want to use. For example, you can use the same one for all servers at your university, and one specifically for GitHub. When you use the `IdentityFile` directive, it tells your SSH client to offer *only* that key to the remote server, and nothing else.

It depends on how paranoid you are, but I recommend this practice.

More detail for the curious: `https://utcc.utoronto.ca/~cks/space/blog/tech/SSHKeysAreInfoLeak`

Reusing the Same Connection

It takes time to make an SSH connection, and often you will open several terminal windows. You can configure SSH to reuse the first connection when you open other terminal windows (or other connections). This will result in faster connections, especially useful if several windows and/or programs are sending requests over SSH.

```
Host gatekeeper
    HostName gatekeeper.zuul.io
    User muna
    Port 2222
    ControlMaster auto
    ControlPath ~/.ssh/connections/%C
    ControlPersist 1m
```

When a new connection is open, a file is created where you specify using the `ControlPath` directive. (Make sure you've created the directory first!)

```
% cd ~/.ssh
% mkdir connections
% chmod 700 connections
```

Let's say you open two terminal windows and `ssh gatekeeper` from each. If you close the first connection, the command will "hang" since the second connection is using it.

The `ControlPersist` directive keeps the connection open in the background so that one window's connection doesn't depend on another. The value of "1m" means that the background connection is kept open for one minute after the last connection has closed.

Universal Settings

There are certain settings that you might want to set for every SSH connection you make. You can do this with a special `Host *` section:

```
Host *
    ControlMaster auto
    ControlPath ~/.ssh/connections/%C
    ControlPersist 1m
    ServerAliveInterval 90
    ServerAliveCountMax 10
    XAuthLocation /opt/X11/bin/xauth
```

`ServerAliveInterval` If your connection is idle (you went for coffee), send message to the server every *n* seconds so it doesn't disconnect you for being idle.

`ServerAliveCountMax` If the server disappears, keep trying *n* times to reconnect.

`XAuthLocation` You might see this error on macOS: "`No xauth data; using fake authentication data for X11 forwarding`." The ssh program doesn't know where your X library (specifically the `xauth` program) is located. If you have XQuartz installed, put the directive and path above in your `Host *` section to get rid of the error.

Remote Commands

You can execute commands on a remove server without directly logging in.

instead of this:

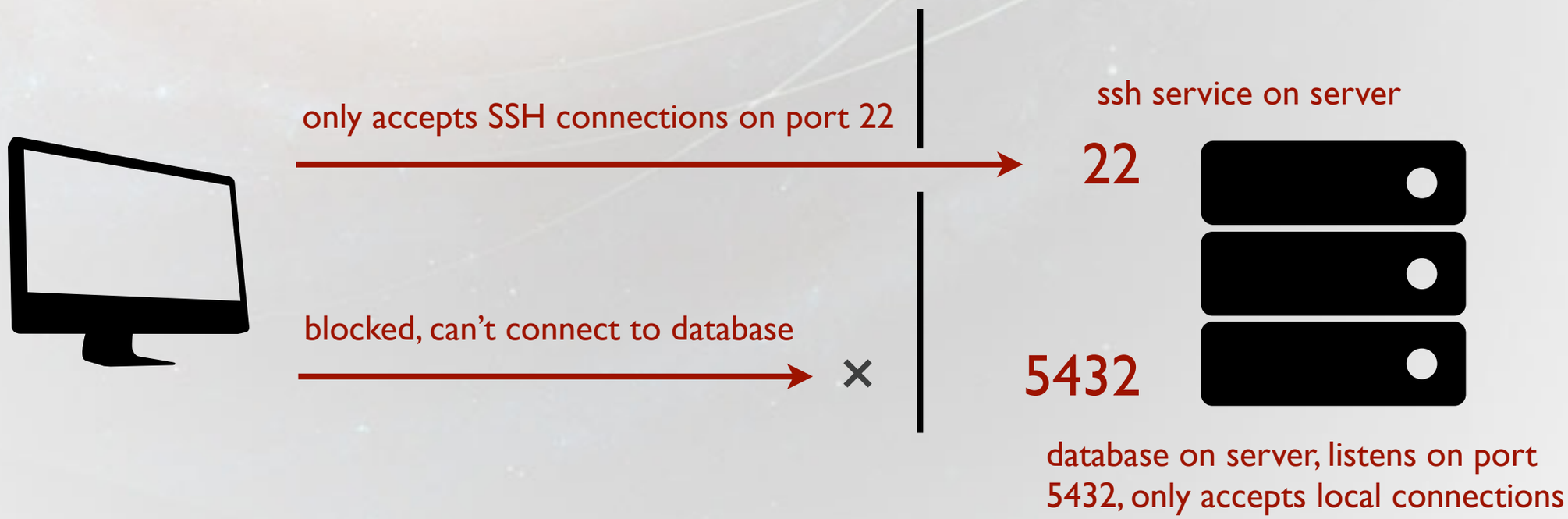
```
% ssh trillian
Last login: Fri Jul 21 11:19:06 2017 from 123.45.67.8
trillian% ls -l
```

you can do this:

```
% ssh trillian 'ls -l'
```

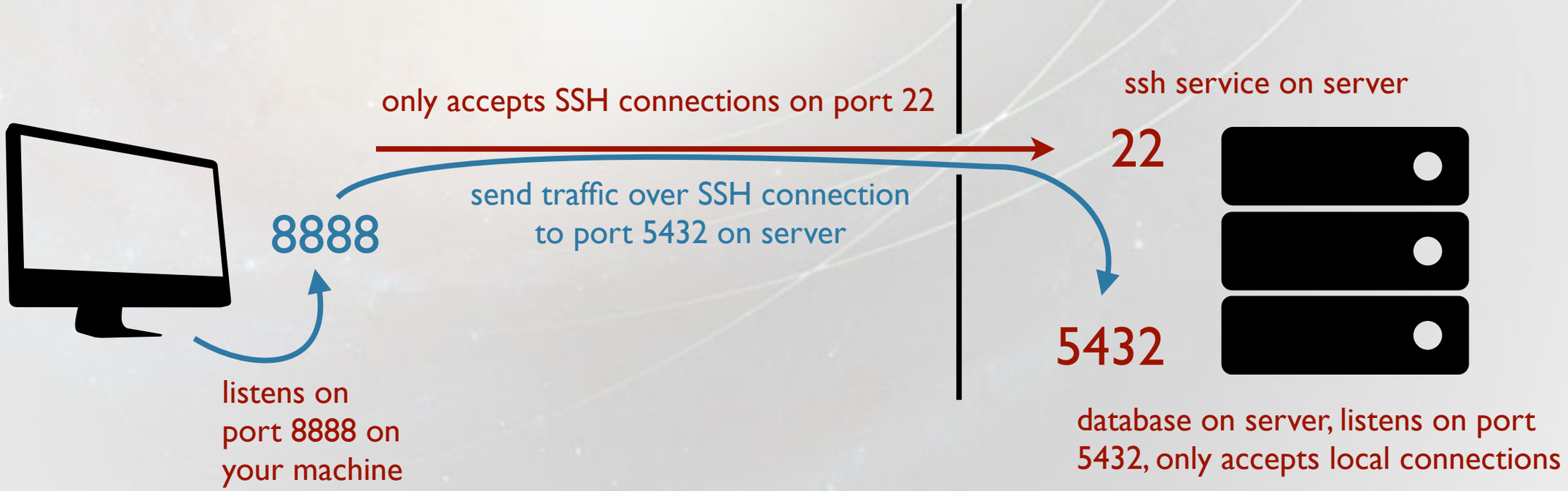
SSH Tunnels

Most servers only allow you to connect via SSH. There might be times when you want to connect to another port, e.g. a database or a development server you are writing.



SSH Tunnels

An SSH tunnel reroutes traffic to a port you specify on your machine, through (tunneled) the SSH connection, then sent to the port on the remote server you want it to go to:



Create a tunnel using this command:

```
% ssh -L 8888:localhost:5432 trillian
```

The local port number is arbitrary, just select something that is not already being used (10000-65535 is safe).

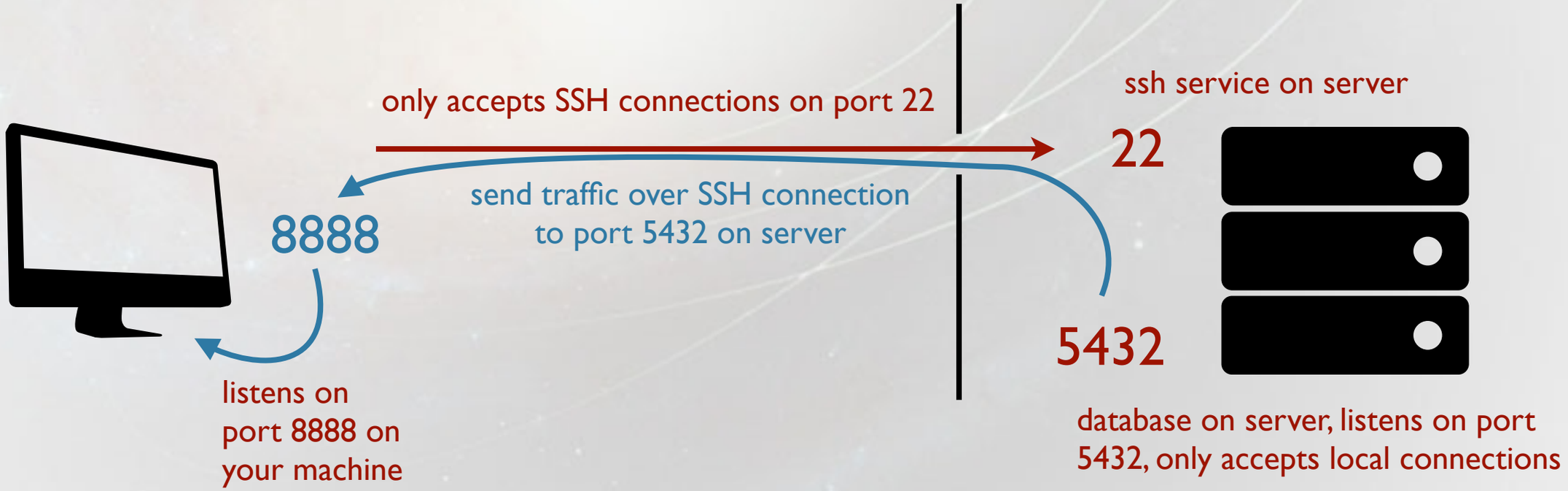
Or always create a tunnel on any connection:

```
Host trillian
    HostName trillianverse.org
    User demitri
    LocalForward 8888 localhost:5432
```

Any data that is sent to localhost:8888 (or 127.0.0.1:8888) is sent to trillian:5432.

SSH Tunnels

The reverse situation is possible – have any traffic sent to a port on the remote server be forwarded to your machine:



Create a reverse tunnel using this command:

```
% ssh -R 5432:localhost:8888 trillian
```

Any data that is sent to trillian:5432 is sent to port 8888 on your computer.

Or always create a tunnel on any connection:

```
Host trillian
    HostName trillianverse.org
    User demitri
    RemoteForward 5432 localhost:8888
```

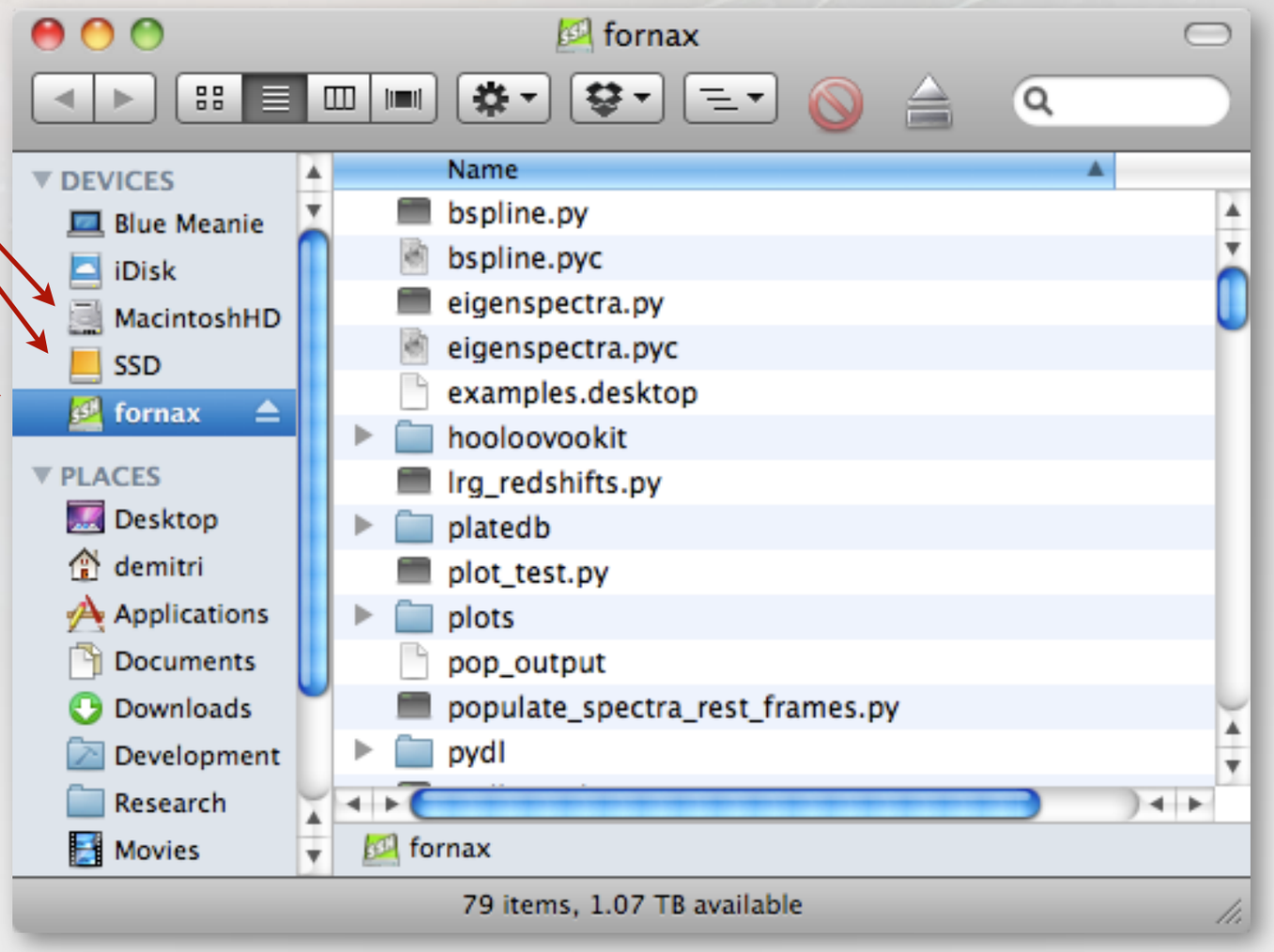
FUSE

We interact with files on remote machines on a daily basis. To read or edit these files, people typically ftp (or scp) files back and forth to your local machine.

A better way is to use FUSE. This allows you to mount any disk you can ssh to as if it were directly plugged into your machine.

Local disks

Remote Linux server



Files act as if they are on my computer (local programs can open them directly), but they are actually sitting on the remote server.

Installing FUSE – Mac

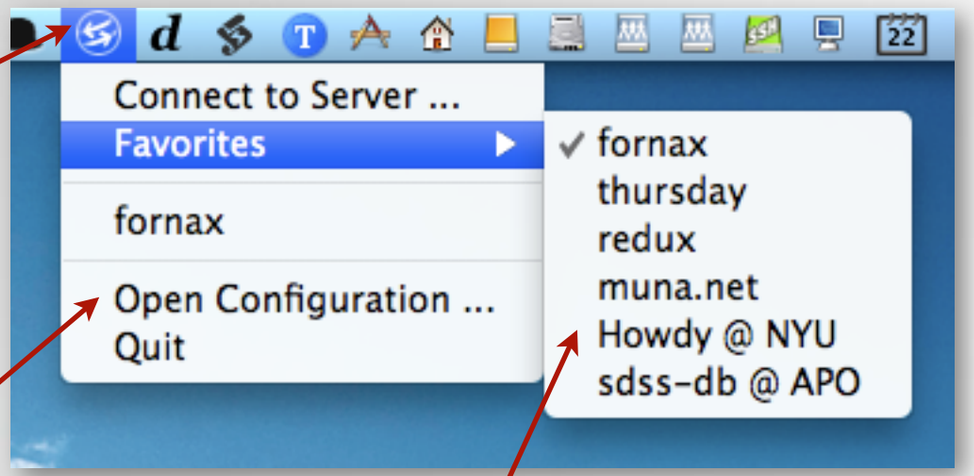
- First install FUSE for macOS & SSHFS (with "compatibility layer" checked):

  http://osxfuse.github.io

- Then install Macfusion, a graphical interface to FUSE (get latest):

  https://github.com/ElDeveloper/macfusion2/releases

In Macfusion, "Start Macfusion Menuitem" to add shortcut menu to the menu bar.



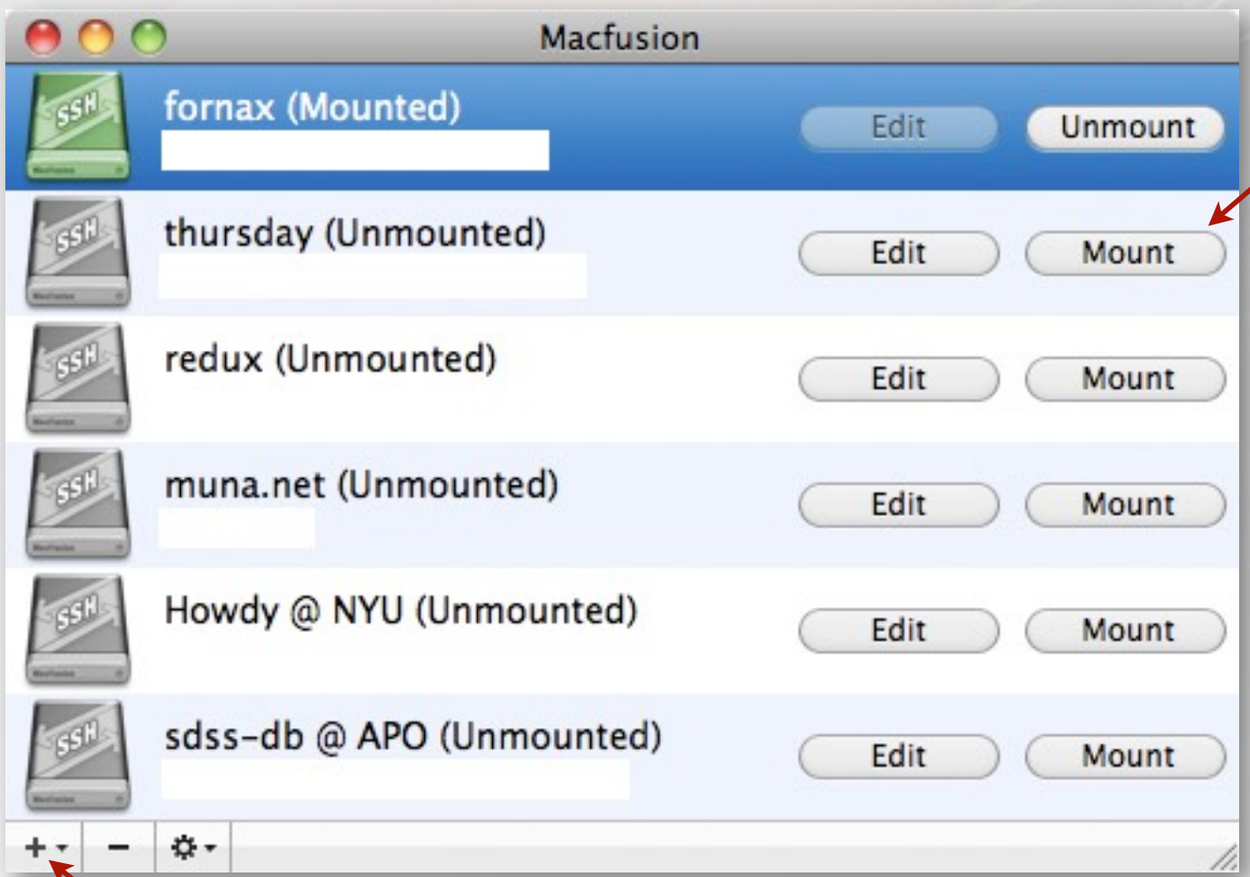
Create new connections.

Just select a pre-configured server to mount on desktop.

Macfusion

We interact with files on remote machines on a daily basis. To read or edit these files, people typically ftp (or scp) files back and forth to your local machine.

A better way is to use FUSE. This allows you to mount any disk you can ssh to as if it were directly plugged into your machine.



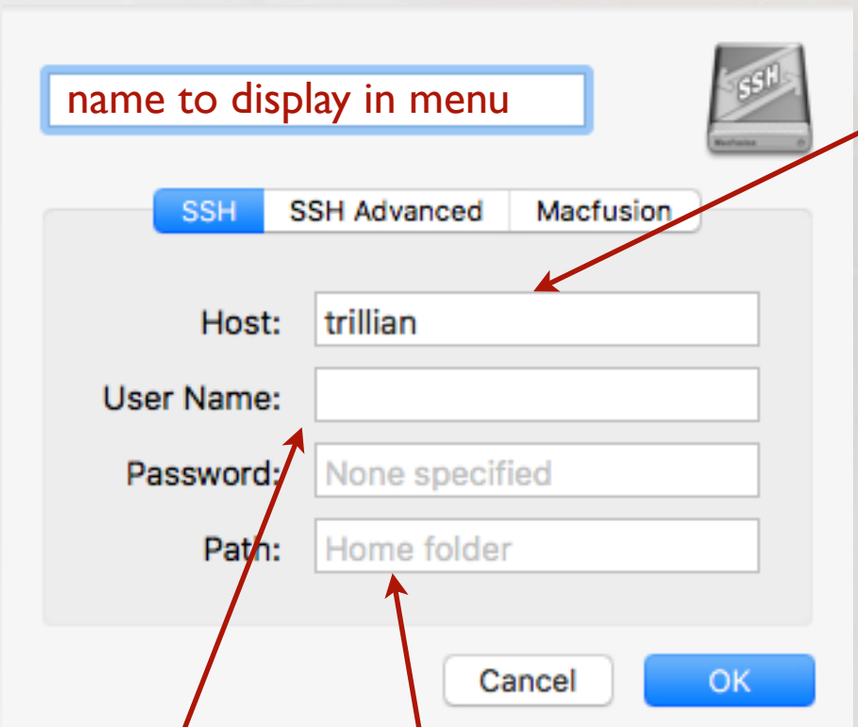
Mount a server (but easier from menu item)

Add a new (SSHFS) configuration

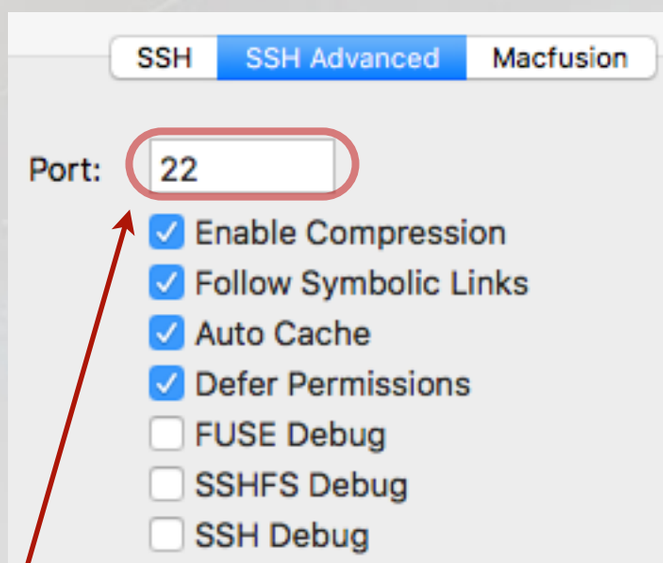
Macfusion

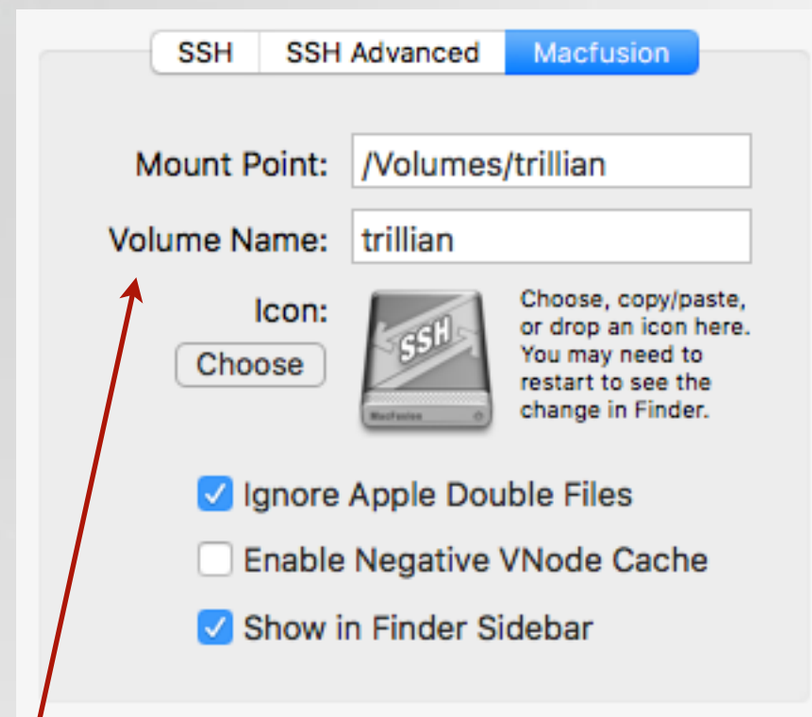
We interact with files on remote machines on a daily basis. To read or edit these files, people typically FTP (or scp) files back and forth to your local machine.

A better way is to use fuse. This allows you to mount any disk you can SSH to as if it were directly plugged into your machine.



can use same bookmark name in .ssh/config





remote server user name and password

the remote directory to mount (must have read access!)

if you ssh to a non-standard port

what you'd like it to be called on the your desktop

Directions When Macfusion is Broken

1. Create a directory where the remote server will appear (a.k.a. "mount point"). If you put it in /Volumes it can be visible in the Finder sidebar, but you need to adjust the permission to your user.

```
% cd /Volumes
% sudo mkdir trillian
% sudo chown demitri trillian
```

Can also leave folder as belonging to root and then run sshfs (below) as root.

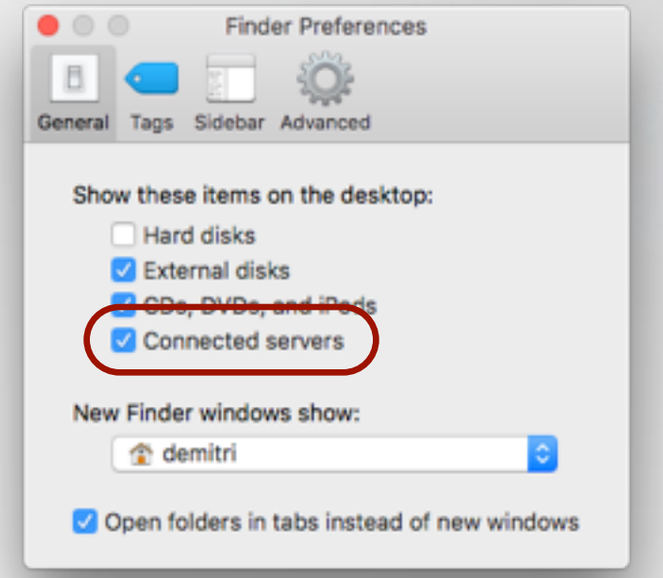
2. Mount the remote server to your desktop.

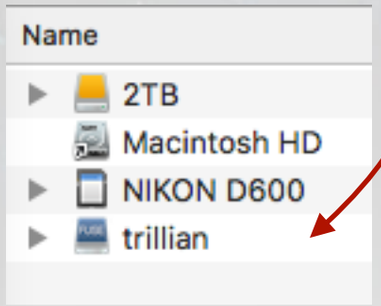
path on remote server, blank for home directory

```
sshfs -o volname=<name on desktop> -o local <ssh bookmark>:<remote path> <mount point>
```

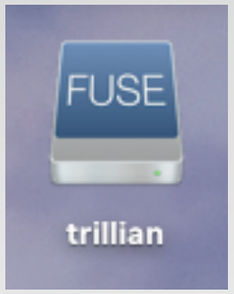
```
% sshfs -o volname=trillian -o local trillian: /Volumes/trillian
```

defined in .ssh/config



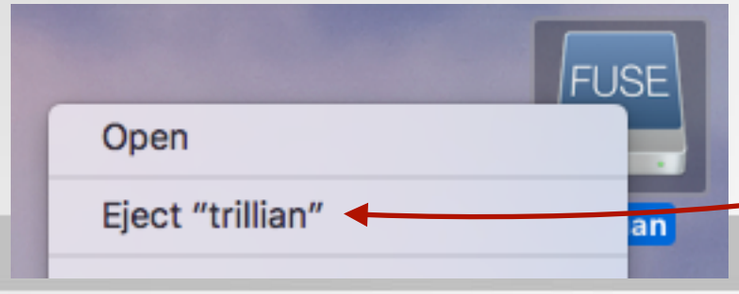


appears on desktop



3. Disconnect server (unmount).

```
% umount /Volumes/trillian
```



or right-click on icon, eject

Directions When Macfusion is Broken

Combine these steps into an alias to make things easier:

```
% alias mount_trillian='mkdir ~/Volumes/trillian;  sshfs -o
volname=trillian -o local trillian: ~/Volumes/trillian'
```

That will require one alias per server.

To make it easier, put this in a file (e.g. remotemount.sh) and source it from your BASH startup (or define the function there directly):

```
remotemount () {
    mkdir ~/Volumes/$1
    sshfs -o volname=$1 -o local $1: ~/Volumes/$1
}
```

Call like this:

same as name defined in .ssh/config

```
% remotemount trillian
```

FUSE – Linux

Install via apt-get (on Ubuntu) or similar:

```
% sudo apt-get install sshfs
% sudo adduser <your username> fuse
```

gives your user permission to use fuse

Log out and then log in. Next, create a directory where we will mount the remote disks:

```
% sudo mkdir -p /mnt/sshfs
% sudo chown <your username>:fuse /mnt/sshfs
```

If your account name on the remote server 'astro.state.edu' is 'abc', you can mount your home directory there in your local home directory to a folder called "astro" there as:

```
% sudo sshfs abc@astro.state.edu: /mnt/sshfs/astro
```

your remote files now appear here as if they were local

To mount another directory, use:

```
% sudo sshfs abc@astro.state.edu:/some/other/dir /mnt/sshfs/astro
```

To unmount the "drive": `% sudo fusermount -u /mnt/sshfs/astro`

FUSE – Linux

You can create aliases to make this simpler for commonly used hosts:

```
# astro
alias mount_astro="sudo sshfs abc@astro.state.edu: /mnt/sshfs/astro"
alias unmount_astro="sudo fusermount -u /mnt/sshfs/astro"

# traal
% alias mount_traal="sudo sshfs abc@traal.stat.edu: /mnt/sshfs/traal"
% alias unmount_traal="sudo fusermount -u /mny/sshfs/traal"
```

bring your towel

Other Topics

You can set up a SOCKS proxy over SSH. This will tunnel your web traffic over your SSH connection.

Benefits:

- All traffic is encrypted – good for when using public wifi.
- Your traffic will appear to be coming from the host you are connecting to – useful when trying to download journal papers when away from the university.

(The details are left as an exercise for the reader before we get too far off course...)