



# Welcome to SciCoder!

# SciCoder 6

## Invited Speakers

Ivelina Momcheva (Yale)

John Parejko (Yale)

Adrian Price-Whelan (Columbia)

Erik Tollerud (Yale)

# SciCoder Introduction

Demitri Muna

8 June 2015

# Optimize Your Code

“We should forget about small efficiencies, say  
about 97% of the time:  
premature optimization is the root of all evil.”

Donald Knuth

↖ the guy who  
invented TeX

So important, I made this the first slide.



# Software Usability (UX)

“Don’t make me think.”

Steve Krug

```
plot(x, y, 'r--')
```

← unless you’re familiar with this, it’s not clear;  
even if you *are* familiar, your mind has to parse it

```
plot(x, y, color='red', linestyle='--')
```

← extremely clear, readable by anyone who’s never  
used this code

# Aims of this Workshop

- Improving your work environment – introducing you to (hopefully!) new tools that will make your work easier.
- Introduce you to good programming practices.
- Show you how to design and program against a database.
- Start to separate “programming” from the details of “syntax”.
- Apply what you are learning to a real research project.

# Aims of this Workshop

- Feel free to ask questions!
- Much of Day One will be laying the foundation for the rest of the week. You will get the most from this workshop if you get today and tomorrow's material, so stop me if you don't get something.
- We are not tied to a specific timetable, so a discussion or clarification will not throw us off schedule.

# Languages

## Scripted

- The code you write is run by an interpreter, line by line.
- Syntax errors are found when the interpreter hits that line.
- The text file is the program.
- Much faster to write and experiment in (just type, then run).
- Slower than compiled languages, but modern techniques and computers have vastly narrowed this gap.
- Examples: Python, Perl, Javascript, shell scripting

## Compiled

- The code you write must be compiled, e.g. turned into machine code.
- Syntax errors will prevent the program from being compiled.
- Slower to develop with (must compile, link, run).
- Programs are faster (but a poorly written C program can be slower than a well-written Python script!).
- Syntax is typically not very clean.
- Good for low-level programming when you need fine control of memory or direct access to hardware (and you don't).
- Examples: C, C++, Fortran, Java



# So, What Language Should I Use?

- There is no one answer – it depends on what you are doing.
  - Aim to minimize:
    - your development time
    - complexity of code
- } The most important asset is your time!  
(Or anyone using/modifying your code.)
- I'd rather spend one day writing code in Python that takes 10 hours to run than one week writing the same thing in C++ that takes 1 hour to run.
  - Even for things like Monte Carlo simulations, typically there are one or two routines that take 90% of the time. You save nothing by optimizing the other routines in your program. And you probably don't know which routines they are!
  - Always work at the highest possible level.

# C

- Don't use it. Seriously.
- C is a bare-bones language. It doesn't come with anything – you have to build everything from scratch.
- C doesn't even know what a string is. You deal with strings *a lot*.
- You have to manage memory yourself.
- C happily lets you read beyond arrays. This is Bad. This is a deal breaker. Don't use arrays.
- C is universal though – just about anything can read a C library. C is then good for writing libraries that will be widely used. But you will pay for it in development time.
- C is highly portable – it will run just about anywhere.
- **Bottom line:** Only use C to write very portable code, and even then only in the form of libraries. You will rarely need to do this.

# C++

- A true object-oriented (OO) language.
- If you don't know the concepts of OO, you can still write a C-like program. This is not a good thing (no gain over using C).
- The syntax is unfriendly and confusing.
- C++ is *strongly typed*.
- The language hasn't been 'updated' in over ten years (and don't hold your breath).
- You have to manage memory yourself. One of the most common types of bug in writing C++ (and C) are related to memory management (leading to increased development time).
- C++ libraries can be linked from other languages (e.g. Python), so also a good choice for portable libraries (but not as universal as C).
- Well-written code will produce programs about as fast as the hardware can run.
- **Bottom line:** Best language for time-critical code, but at the cost of higher development time and time spent debugging.

# Python

- A true object-oriented (OO) language.
- Easy to learn, the syntax is clean and natural.
- Python is *weakly typed*.
- Memory is managed for you.
- The language has *many* built-in features so you don't have to keep reinventing the wheel, and hundreds of more specialized libraries are freely available.
- The language is continually being updated with more functionality.
- For most of what you'll need to do, the code is functionally as fast as C/C++.
- Can link to C/C++ libraries for the most CPU-intensive code.
- **Bottom line:** Best language for the vast majority of your programming tasks, if for no other reason the short development time (remember, *your* time is more valuable than writing code that's 10% faster!).



# Perl

- Another scripted language, very popular on the Internet.
- Largely supplanted by Python due to the fact that the syntax isn't as clean.
- More external libraries than Python (but Python catching up every day).
- Very similar to Python – you can easily learn Perl after knowing Python (or vice versa).
- Best implementation of regular expressions in a programming language.
- Has OO features, but to use true OO the syntax is awkward (added on later, not part of initial design).
- Perl 6 update has taken over 10 years to arrive (and hasn't yet); people just moved to Python.
- **Bottom line:** Use Python. However, so much code was written in Perl that you'll probably come across some. Worth learning the basics to at least be able to read Perl code and maybe make some modifications.

# IDL

personal rant

- An anchor around the astronomical community's neck.
- Thin veil over C (but still has FORTRAN/VAX-isms!), and it doesn't hide nearly enough.
- The language has not been updated in *years*, and lacks now-common data structures such as dictionaries (a.k.a hashes, maps, associative arrays).
- ITT (the company) has no real interest in making the astronomical community happy.
- IDL costs *a lot*. Scientific computing should be *open*.
- The features that made IDL so appealing in the late 70s and 80s (visualization without complex programming, manipulating arrays as first-class objects) have been far surpassed by modern languages.
- The syntax is awkward. Graphics are ugly.
- IDL will frequently silently fail, so a lot of IDL code is checking that things worked as you expected them to.
- A large number of astronomical libraries exist in IDL.
- **Bottom line:** Move away from IDL when you can. As a community, we should move to convert the libraries we have into Python (people are doing this, but we need to be better coordinated). As best you can, don't write any new IDL code. When converting code, use OO concepts rather than a one-to-one conversion of routines.

Your professors used to use VAX when they were students. 'Nuff said.

# Examples of Why IDL is Bad

```
openr, lun, 'data.txt', /get_lun
```

I have to keep track of file numbers? You're a computer, you do it!

```
print, a
```

Why can't I type the variable alone to print out the value? Typing "print," wastes time.

Can't create an empty array. There is such a thing as an empty set.

If you modify a source file, you have to remember to recompile it. You're a computer, notice it yourself.

When you resize a plot window, the plot doesn't redraw. This is something the Mac can do. Since 1984.

Objects (as in OO) are possible, but the syntax and handling is awkward at best. And no one does it anyway.

No database connectivity.\*

\*Unless you pay more. And use ODBC (a niche Windows technology). This is a deal breaker - database access is not an advanced feature!

So as not to be wholly negative, here are two tips. Always place this at the beginning of every IDL procedure you write.

```
compile_opt idl2
compile_opt logical_predicate
TRUE = 1
FALSE = 0
```

- Defaults ints to 32-bit instead of 16.
- Does not allow use of () to access arrays.
- Non-zero values evaluate to false, zero evaluates to true.
- Use the TRUE and FALSE variables for readability.

```
IF (NOT TRUE) ...
IF (~TRUE) ...
```

← If this is in your code, it doesn't do what you think it does (bitwise operator).

← This is the boolean NOT operator.

And that's all I'll say about IDL this week.

# Optimize Your Code

- Your computer is not the computer your advisor grew up with.
- Computers are *fast* today. You don't have to worry about a few tens of bytes here and there – people still do this.
- Code readability and reuse is *far* more important than running time for nearly everything you'll do.
- Learn to use profiling for time-critical or CPU intensive code – let the computer tell you the bottleneck, don't guess (we'll come back to this later).

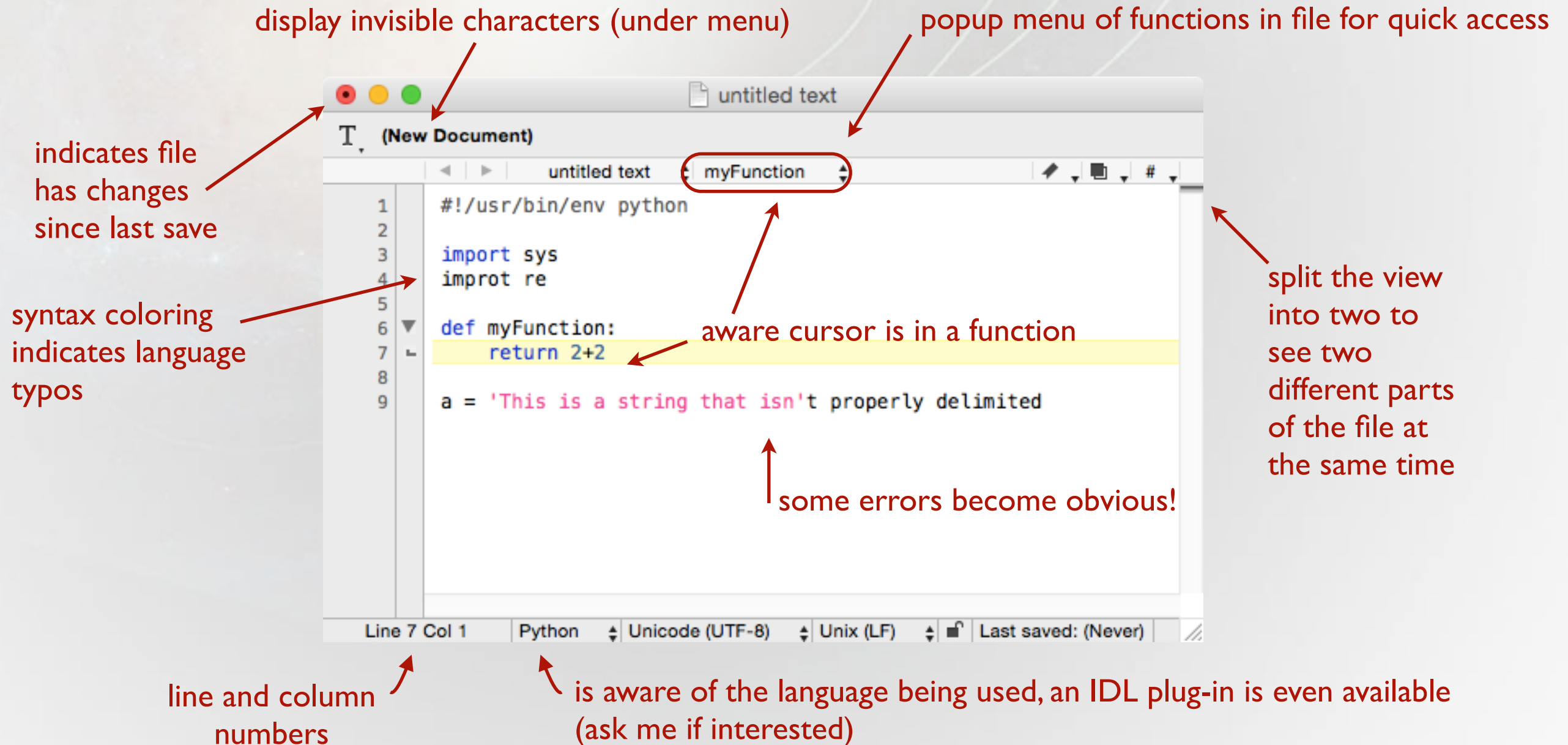


# Choose a Good Text Editor

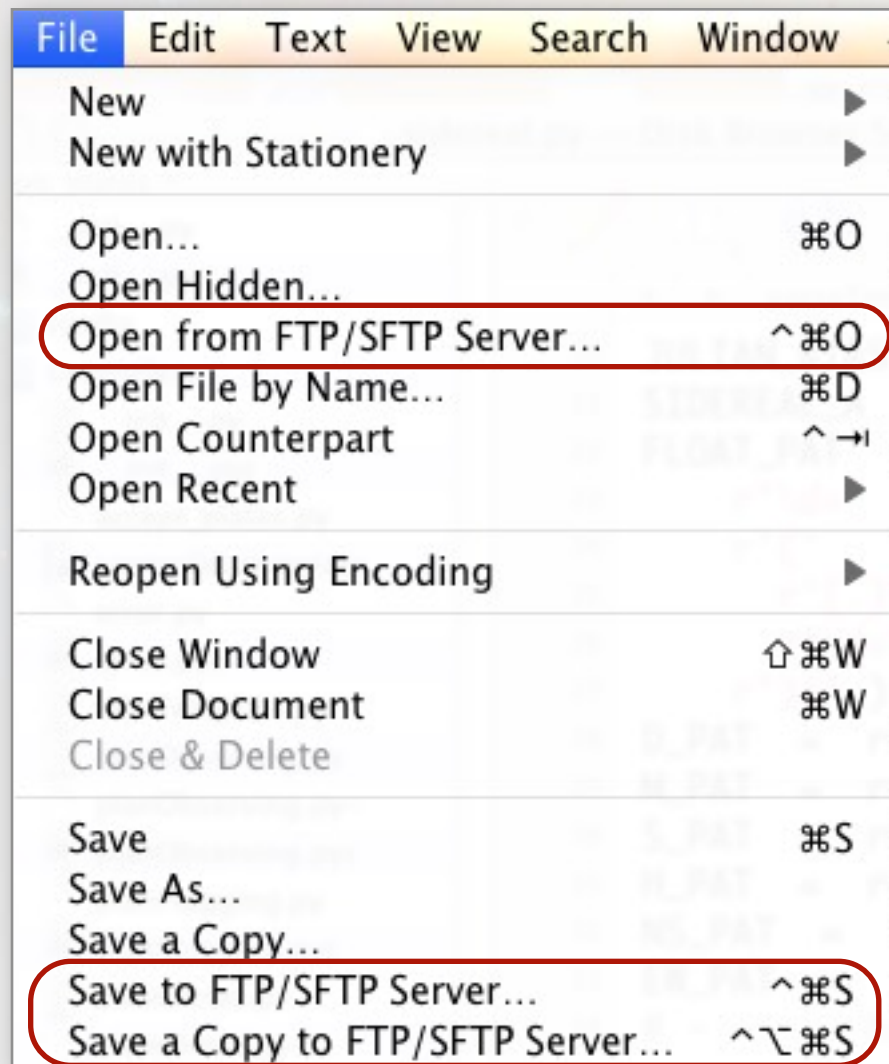
- You spend most of your time coding in a text editor – pick one that is easy to use and can help you.
- The choice of a text editor for some is like a religion. If you've picked yours, at least see what's available.
- Features to look for:
  - syntax highlighting (must have!!)
  - line numbers
  - can execute code directly from the editor (very handy)
  - aware of functions
- I recommend not using terminal editors such as emacs/vi for day to day coding. You *should* be very comfortable in one or the other, but there are simpler tools available.

# Mac Text Editor

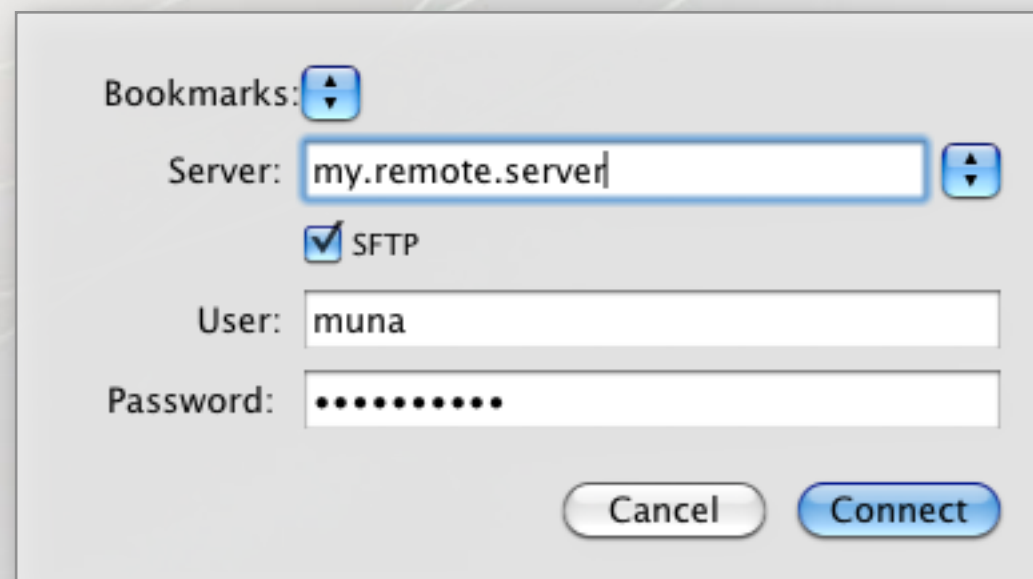
## Text Wrangler <http://barebones.com>



# Edit Remote Files



Open a file from your linux server, either locally or anywhere else. Edit it as if it's on your machine. When you save the file, it saves back to the server.

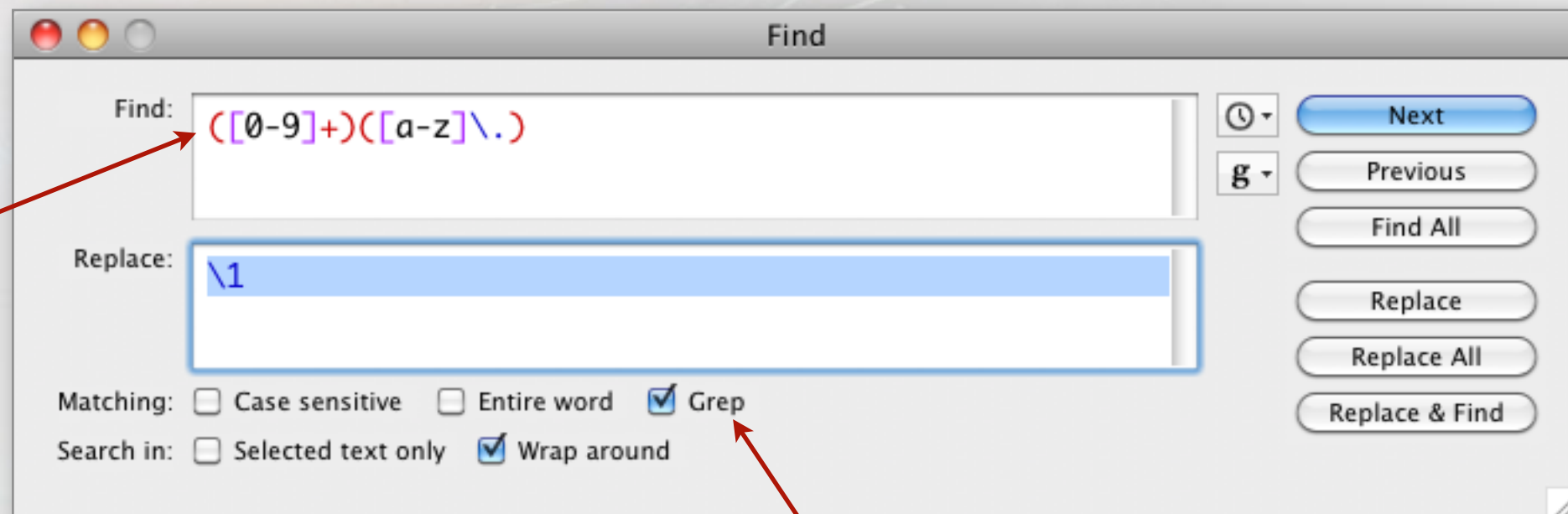


You shouldn't have to download a remote file, edit it locally, and upload it back again.

# Regular Expressions in the Editor

A good editor will handle regular expressions in the search and replace panel. When you learn these (which we will later cover), you can often perform certain tasks with this instead of having to write a program.

syntax highlighting will indicate whether the regular expression is valid



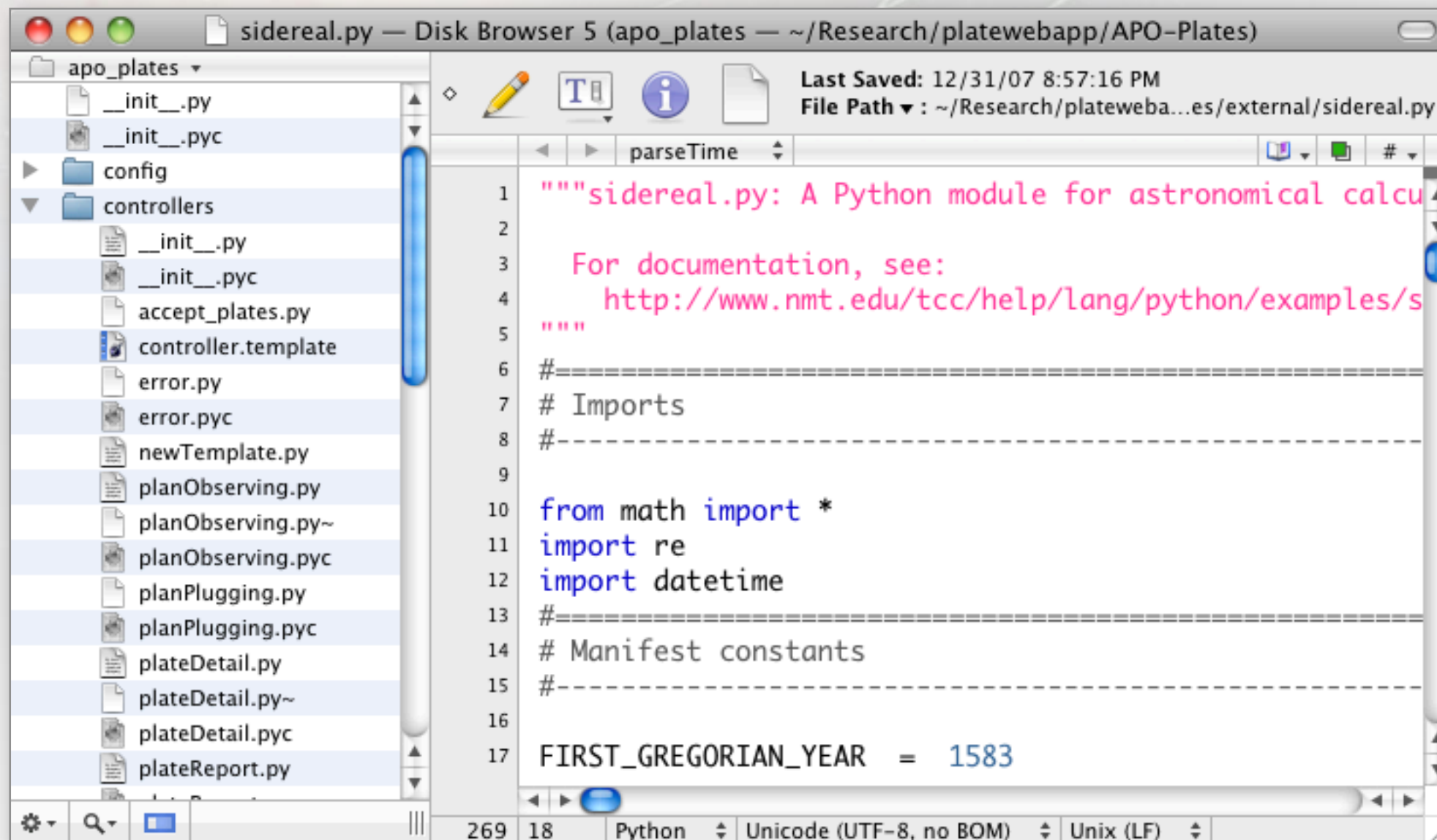
can use grep (regular expressions) or not

This just scratches the surface – there are many more features we will gradually cover.



# Project Editing

Several editors offer “disk browsers” that allow you to access several files at one time. This is similar to an IDE (integrated programming environment) such as Xcode and Eclipse. This eases work in larger projects and offers features such as multi-file search and replace.



# Linux Text Editors

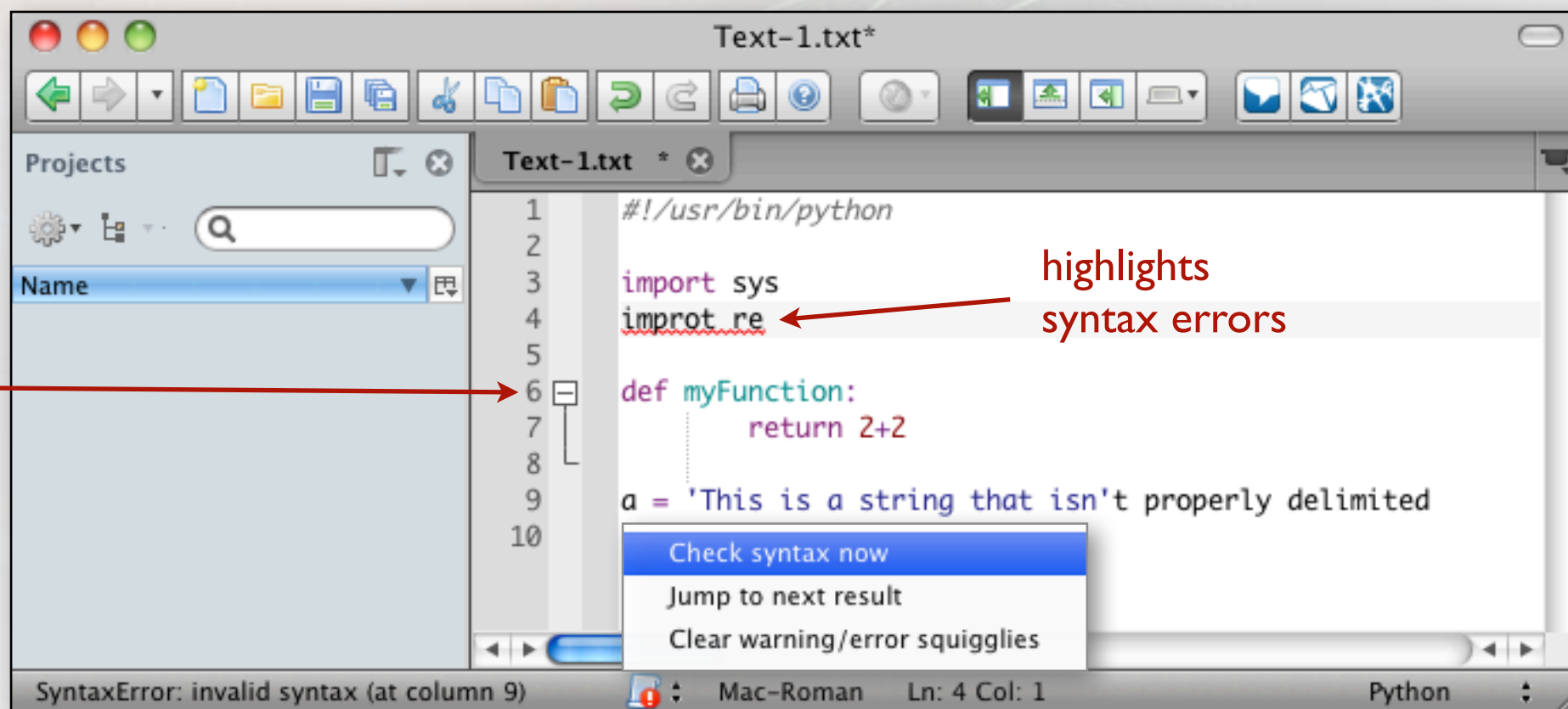
For Linux users (obviously) but also for Mac/Windows users when they are working on a remote server. These applications can be launched over X11 (but I prefer to use a local editor whenever possible).

## Komodo Edit <http://www.activestate.com/komodo-edit/downloads>

features code completion

code folding: collapse long functions for better readability

it even tells you the problem

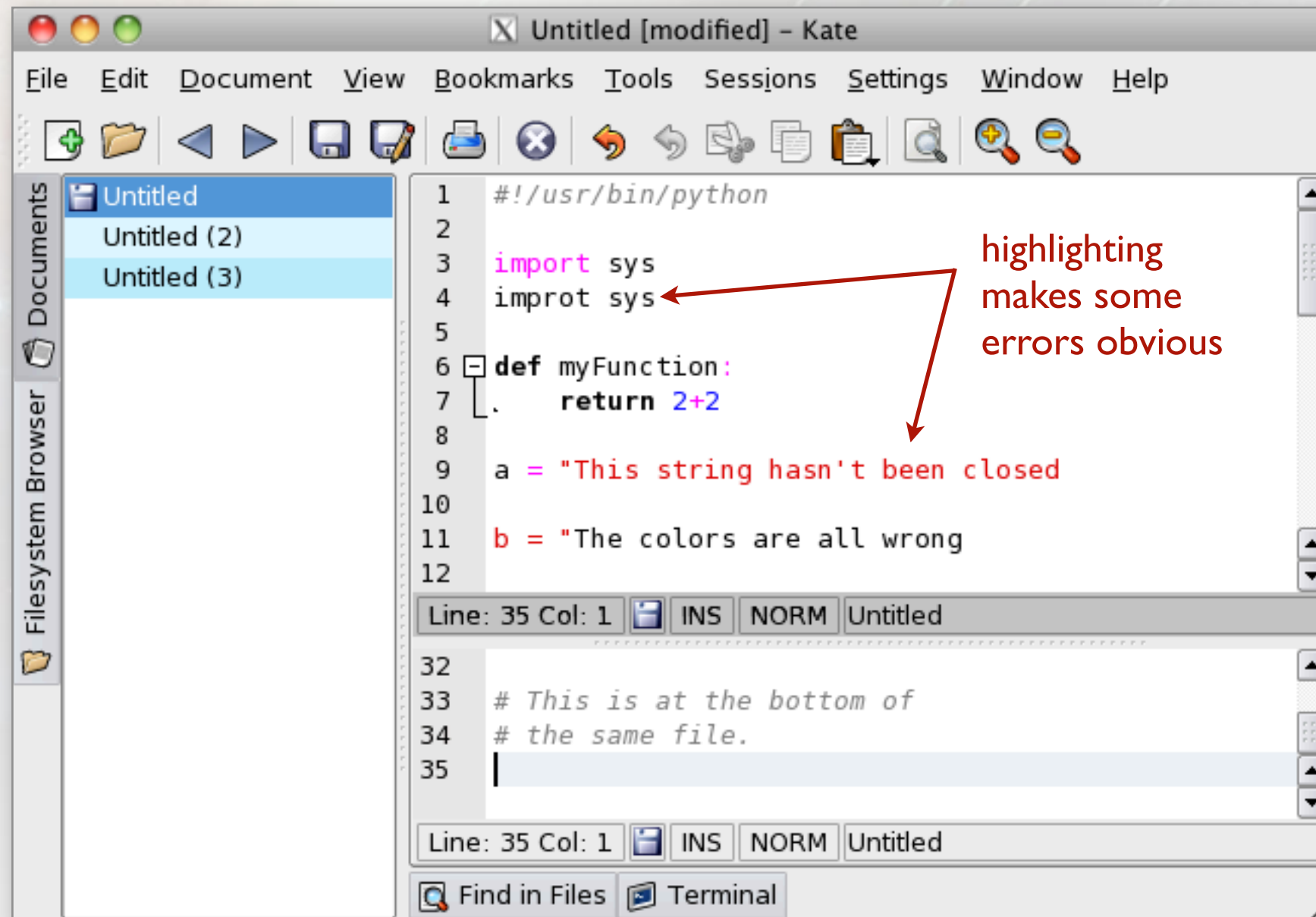


Available for free on Linux/Mac/Windows.

select language here

# Linux Text Editors

**Kate** (part of KDE, install package “kate” in Ubuntu)



work on  
multiple files at  
once

highlighting  
makes some  
errors obvious

split view of  
same file

# Your “Everything Box”

We gather tons of information that takes the form of small snippets of data: web bookmarks, locations of data or servers that we use, frequent flier numbers, etc. It doesn't make sense to save each of these things in a text document on our hard drive — that solves the problem of saving it, but not finding it quickly or easily. And it generates more work to keep it organized (which most people don't do, so data gets lost).

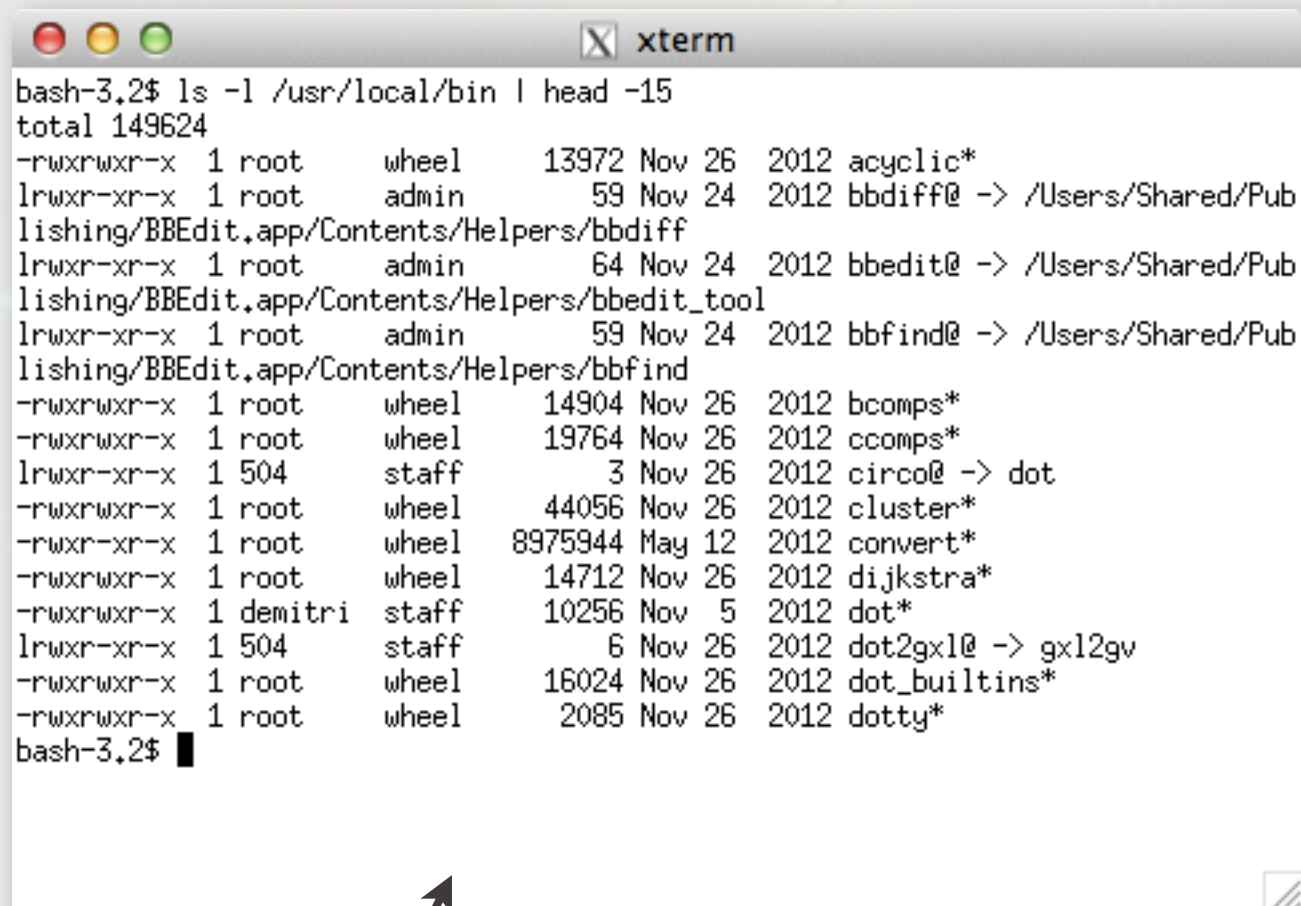
An “everything box” is a program that accepts all kinds of data like this — text, pictures, passwords, etc.

You can tag the information so you can find things in a way that makes sense to you (even using multiple tags).

Evernote is what I currently recommend.



# Your Terminal & You



```
bash-3.2$ ls -l /usr/local/bin | head -15
total 149624
-rwxrwxr-x 1 root    wheel    13972 Nov 26  2012 acyclic*
lrwxr-xr-x 1 root    admin     59 Nov 24  2012 bbdiff@ -> /Users/Shared/Pub
lishing/BBEdit.app/Contents/Helpers/bbdiff
lrwxr-xr-x 1 root    admin     64 Nov 24  2012 bbedit@ -> /Users/Shared/Pub
lishing/BBEdit.app/Contents/Helpers/bbedit_tool
lrwxr-xr-x 1 root    admin     59 Nov 24  2012 bbfind@ -> /Users/Shared/Pub
lishing/BBEdit.app/Contents/Helpers/bbfind
-rwxrwxr-x 1 root    wheel    14904 Nov 26  2012 bcomps*
-rwxrwxr-x 1 root    wheel    19764 Nov 26  2012 ccomps*
lrwxr-xr-x 1 504     staff      3 Nov 26  2012 circo@ -> dot
-rwxrwxr-x 1 root    wheel    44056 Nov 26  2012 cluster*
-rwxr-xr-x 1 root    wheel   8975944 May 12  2012 convert*
-rwxrwxr-x 1 root    wheel    14712 Nov 26  2012 dijkstra*
-rwxrwxr-x 1 demitri staff    10256 Nov  5  2012 dot*
lrwxr-xr-x 1 504     staff      6 Nov 26  2012 dot2gxl@ -> gxl2gv
-rwxrwxr-x 1 root    wheel    16024 Nov 26  2012 dot_builtins*
-rwxrwxr-x 1 root    wheel     2085 Nov 26  2012 dotty*
bash-3.2$
```

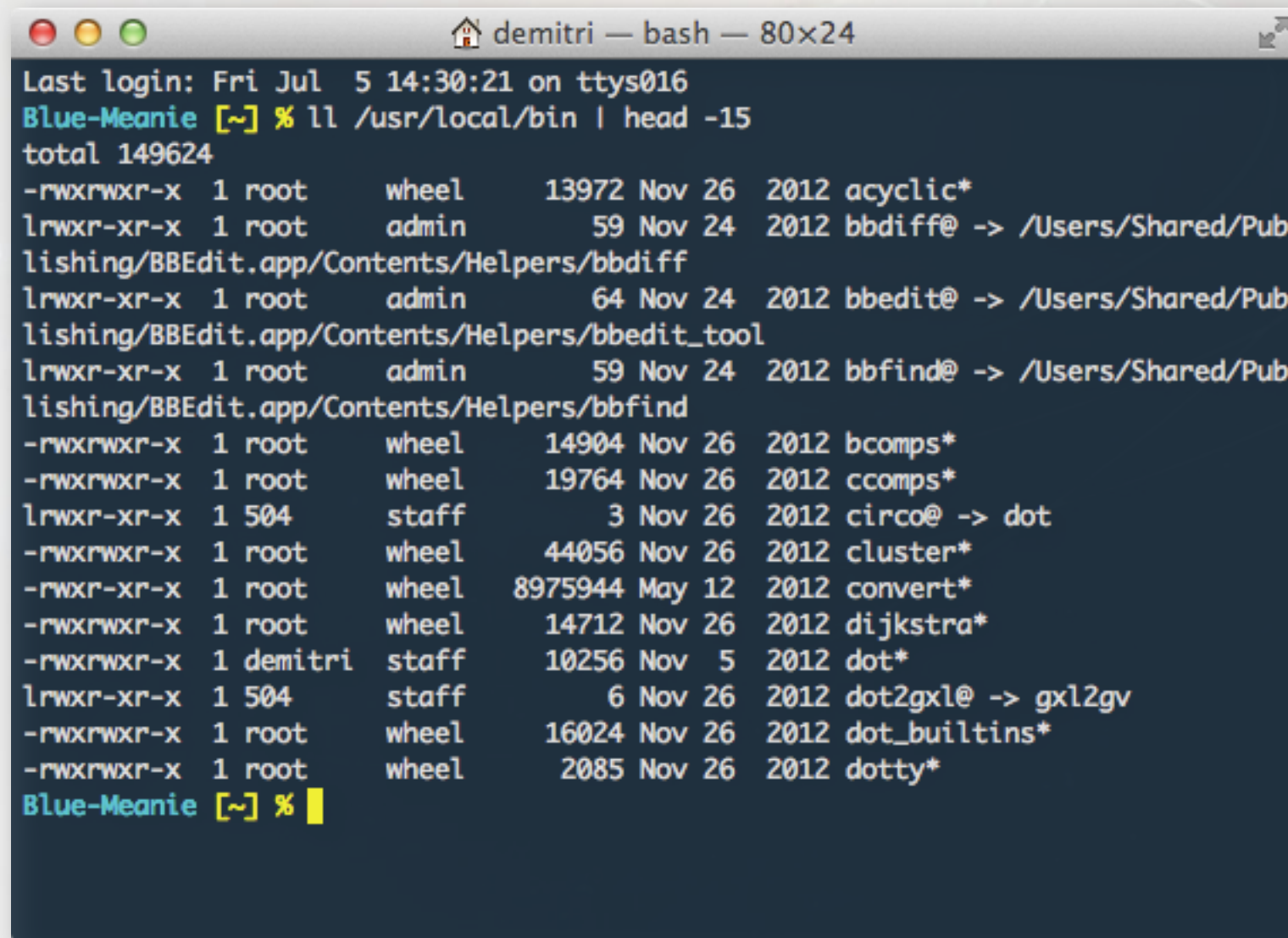
You don't want to look at this all day long. You have better standards than that.

## What's wrong with xterm?

- Boring
- Hard to read
- Can't paste text into window
- Pixelated
- Black and white
- Makes small children cry

There is NO good reason to use Xterm. Don't do it.

# Upgrading Your Terminal



```
demitri — bash — 80x24
Last login: Fri Jul 5 14:30:21 on ttys016
Blue-Meanie [~] % ll /usr/local/bin | head -15
total 149624
-rwxrwxr-x 1 root wheel 13972 Nov 26 2012 acyclic*
lrwxr-xr-x 1 root admin 59 Nov 24 2012 bbdiff@ -> /Users/Shared/Pub
lishing/BBEdit.app/Contents/Helpers/bbdiff
lrwxr-xr-x 1 root admin 64 Nov 24 2012 bbedit@ -> /Users/Shared/Pub
lishing/BBEdit.app/Contents/Helpers/bbedit_tool
lrwxr-xr-x 1 root admin 59 Nov 24 2012 bbfind@ -> /Users/Shared/Pub
lishing/BBEdit.app/Contents/Helpers/bbfind
-rwxrwxr-x 1 root wheel 14904 Nov 26 2012 bcomps*
-rwxrwxr-x 1 root wheel 19764 Nov 26 2012 ccomps*
lrwxr-xr-x 1 504 staff 3 Nov 26 2012 circo@ -> dot
-rwxrwxr-x 1 root wheel 44056 Nov 26 2012 cluster*
-rwxr-xr-x 1 root wheel 8975944 May 12 2012 convert*
-rwxrwxr-x 1 root wheel 14712 Nov 26 2012 dijkstra*
-rwxrwxr-x 1 demitri staff 10256 Nov 5 2012 dot*
lrwxr-xr-x 1 504 staff 6 Nov 26 2012 dot2gxl@ -> gxl2gv
-rwxrwxr-x 1 root wheel 16024 Nov 26 2012 dot_builtins*
-rwxrwxr-x 1 root wheel 2085 Nov 26 2012 dotty*
Blue-Meanie [~] %
```

- On Mac OS X, use the Terminal program. It will work seamlessly with any X application.
- Each Linux will have a native Terminal application.
- Copy/paste work with everything (except X applications).
- Set up a color scheme for each server you use on a regular basis – it's a quick visual indicator to see where you are.
- Text is easier on your eyes.
- Color will help guide your eye and find information more quickly.
- Can drag/drop text into window.

# Upgrading Your Terminal

```
log — bash — 98x24
Last login: Fri Jul 5 14:35:57 on ttys036
Blue-Meanie [~] % cd /var/log
Blue-Meanie [/var/log] % ll | head -15
total 5960
-rw-r--r--@ 1 root      wheel      12 Nov 14 2012 CDIS.custom
drwxrwx--- 24 root      admin        816 Jul  5 11:42 DiagnosticMessages/
drwxr-xr-x  2 root      wheel         68 Jun 20 2012 apache2/
-rw-r--r--  1 root      wheel      3439 May  6 16:54 appfirewall.log
drwxr-xr-x 49 root      wheel      1666 Jul  5 12:57 asl/
drwxr-xr-x  4 root      wheel       136 Apr  9 11:35 com.apple.launchd/
drwx----- 4 root      wheel       136 Mar 24 20:15 com.apple.launchd.peruser.0/
drwx----- 3 daemon    wheel       102 Apr 17 13:11 com.apple.launchd.peruser.1/
drwx----- 4 _softwareupdate wheel       136 Jul  5 12:57 com.apple.launchd.peruser.200/
drwx----- 3 Guest      wheel       102 Nov 15 2012 com.apple.launchd.peruser.201/
drwx----- 4 _coreaudiod wheel       136 May  6 16:54 com.apple.launchd.peruser.202/
drwx----- 4 _cvmsroot  wheel       136 Jun 19 23:23 com.apple.launchd.peruser.212/
drwx----- 4 _lp        wheel       136 Jun  8 10:25 com.apple.launchd.peruser.26/
drwx----- 4 nobody     wheel       136 Jul  5 12:35 com.apple.launchd.peruser.4294967294/
Blue-Meanie [/var/log] %
```

Discontiguous text selection!



# Managing SSH

Standard way to connect via SSH:

```
% ssh muna@bender.osu.edu
```

SCP to the same machine:

```
% scp file.txt muna@bender.osu.edu:
```

Often servers are behind a gateway server, so to log into them from outside your institution, you have to SSH into two machines:

```
% ssh muna@gateway.osu.edu
```

```
% ssh muna@bender.osu.edu
```

This can turn into a lot of typing for common tasks, particularly if the names are long. You may also have to remember different user names or port numbers on different machines.



# Managing SSH

Create a file called `config` in the `.ssh` directory in your home directory:

```
Host bender
```

```
User muna
```

```
HostName bender.osu.edu
```

← set if the port is non-standard (i.e. not 22)

```
Port 1234
```

```
ForwardX11Trusted yes
```

← open any remote X11 application on your computer

```
ForwardAgent yes
```

```
Compression yes
```

← carry your SSH key authentication to other hosts  
you connect to

```
Host benderosu
```

```
User muna
```

```
Hostname bender.osu.edu
```

```
Port 1234
```

```
ProxyCommand ssh -q muna@gateway.edu nc %h %p
```

← first connect to the gateway server, then  
immediately SSH into the next machine

Now this works:

```
ssh bender
```

```
scp file.txt bender:
```

```
ssh bender
```

# A Nicer README

- It's always a good idea to put “readme” files with your code. They can describe the overall structure of your code or data and provide a good entry point to how to use the code and how it's organized.
- This is useful for your own code – you forget things.
- Plain text files still dominate – they can be easily read anywhere and are easy to modify by anyone (anywhere).
- But they're bland and boring...

# Markup Languages

- A markup language provides a means to indicate decoration (e.g. bold, italics, boxes) with plain text.
- Example languages: HTML, LaTeX

HTML

```
<h3>A header</h3>
<table border="1">
<tr><td align="center">
<b>Audentes</b> fortuna
iuvat.
</td></tr>
</table>
```

**A header**

**Audentes** fortuna iuvat.

LaTeX

```
\Psi =
\frac{e^{2/\theta_0}}
{(2\pi - 86)(i + 99)}
```

$$\Psi = \frac{e^{2/\theta_0}}{(2\pi - 86)(i + 99)}$$

plain text, but hard to read

easy to read, but not editable

# Markdown

- Markdown is a markup language that is easy to read, even with the markup “code”.
- Supports lists, styles, links, tables, images, inline code, math equations, more.
- GitHub renders markdown files automatically.
- Files are still plain text, but are indicated with a “.md” extension.
- There are editors that provide a split view – the text on one side, and the rendered view in the other.

## Editors

Linux: <http://uberwriter.wolfvollprecht.de>

Mac: <http://mouapp.com>

Web: <http://benweet.github.io/stackedit/>

Mac QuickLook plugin: <https://github.com/toland/qlmarkdown>

Emacs: <https://github.com/jamesnvc/emacs.d/blob/master/modes/multimarkdown-mode.el>



# Markdown Example

markdown\_example.md83 Words

### ###Welcome to my README file

Here are some things you can do

- \* An inline link to [SciCoder](http://scicoder.org)
- \* A raw URL: <http://xkcd.com>
  - \* nest list items
- \* inline code: try the `try` statement!

Indicate block code with four leading spaces:

```
for i in range(100):
    print i
```

Have a table!

First Header	Second Header	Third Header
Content Cell	Content Cell	Content Cell
Content Cell	Content Cell	Content Cell

Equations are surrounded by "\$\$":

$$\Psi = \frac{e^{2/\theta_0}}{(2\pi - 86)(i + 99)}$$

## Welcome to my README file

Here are some things you can do

- An inline link to [SciCoder](http://scicoder.org)
- A raw URL: <http://xkcd.com>
  - nest list items
- inline code: try the `try` statement!

Indicate block code with four leading spaces:

```
for i in range(100):
    print i
```

Have a table!

First Header	Second Header	Third Header
Content Cell	Content Cell	Content Cell
Content Cell	Content Cell	Content Cell

Equations are surrounded by "\$\$":

$$\Psi = \frac{e^{2/\theta_0}}{(2\pi - 86)(i + 99)}$$

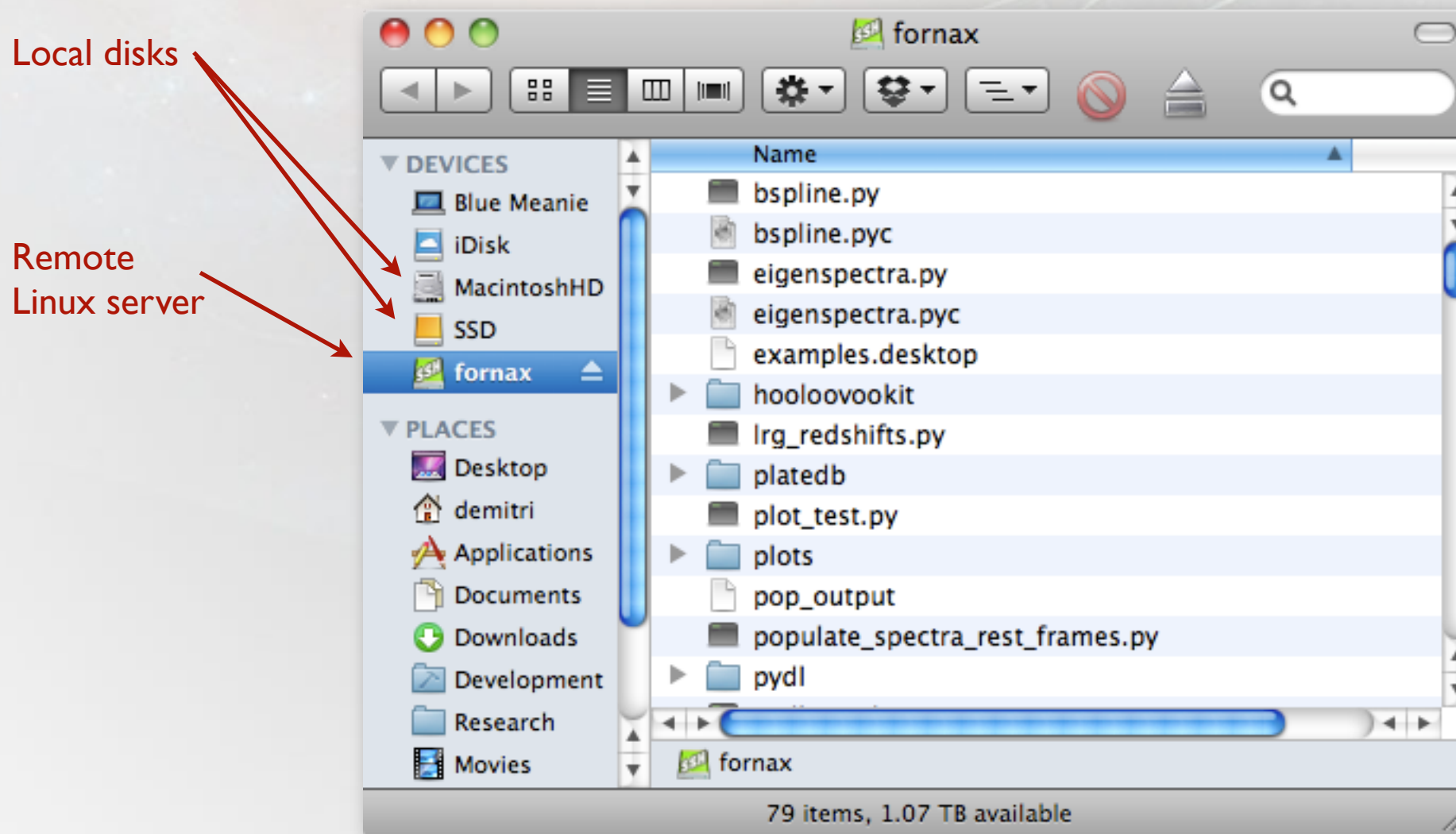
Edit this side

Live updates here

# 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.



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 OSXFuse:

<http://osxfuse.github.io>

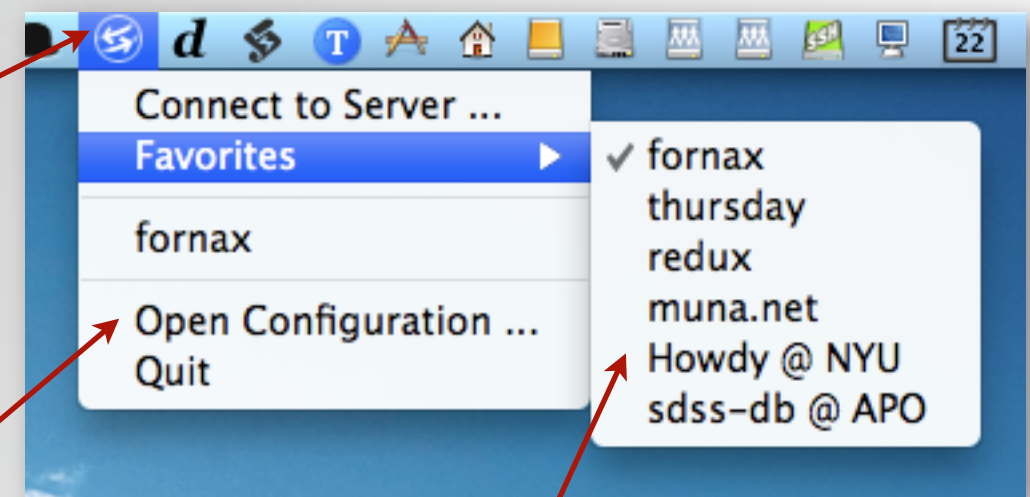
- Then install Macfusion, a graphical interface to mac-fuse:

<http://www.danielzhang.info/drupal6/system/files/Macfusion.app-built-by-Daniel-Zhang.zip>



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

Create new connections.



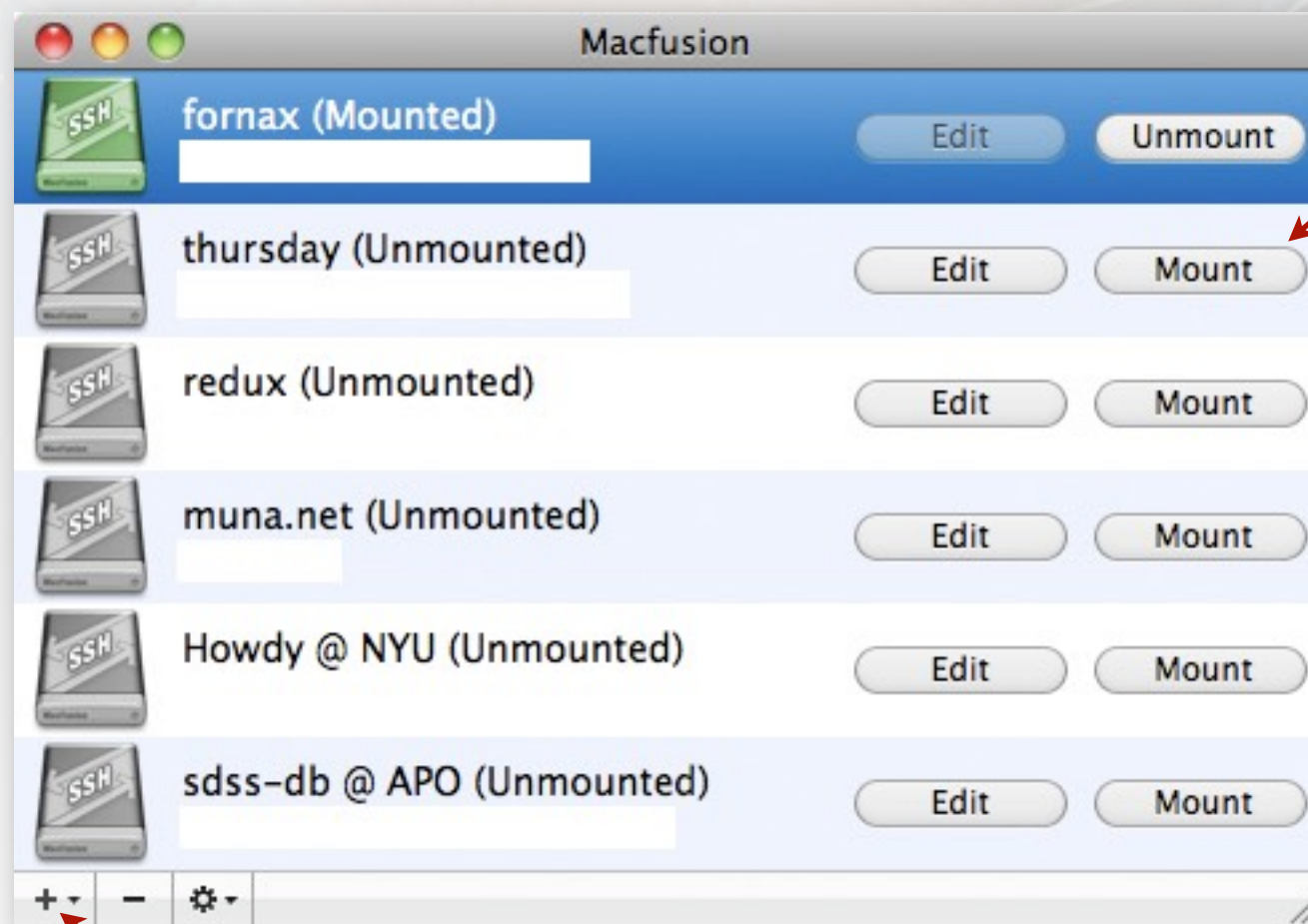
Just select a pre-configured server to mount on desktop.



# 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.

The image shows three sequential screenshots of the Macfusion application window, illustrating the configuration process. Red arrows and text provide context for the fields:

- SSH Tab:**
  - configuration name:** Points to the title bar text field.
  - Host:** Set to "None".
  - User Name:** Set to "demitri".
  - Password:** Set to "None specified".
  - Path:** Set to "Home folder".
- SSH Advanced Tab:**
  - Port:** Set to "22". A red circle highlights this field, with an arrow pointing to the text "can use same shortcut in .ssh/config".
  - Extra Options:** Set to "None specified".
- Macfusion Tab:**
  - Mount Point:** Set to "Default".
  - Volume Name:** Set to "Default".
  - Icon:** A custom SSH icon is selected.
  - Ignore Apple Double Files:** Unchecked.
  - Enable Negative VNode Cache:** Unchecked.

Additional explanatory text at the bottom:

- remote server user name and password:** Points to the User Name and Password fields in the SSH tab.
- the directory to mount (must have read access!):** Points to the Path field in the SSH tab.
- if you ssh to a non-standard port:** Points to the Port field in the SSH Advanced tab.
- what you'd like it to be called on the your desktop:** Points to the Volume Name field in the Macfusion tab.

# 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
```

← bring your towel

```
% alias mount_traal="sudo sshfs abc@traal.stat.edu: /mnt/sshfs/traal"
```

```
% alias unmount_traal="sudo fusermount -u /mnt/sshfs/traal"
```