

# Computational Physics

Prof. J.L. Sievers  
[sieversj@ukzn.ac.za](mailto:sieversj@ukzn.ac.za)

# Course Goals

- If you go on in physics, most work likely to be at computer
- The computer will do work for you. It will *\*not\** think for you.
- The only way to learn how to write code is by writing code.
- The goal of this course is to be practical. Focus on tools you'll (likely) need to get things done.
- Lectures/tutorial will be in lab. Try out commands right away.
- What do you want to learn about?

# Programming

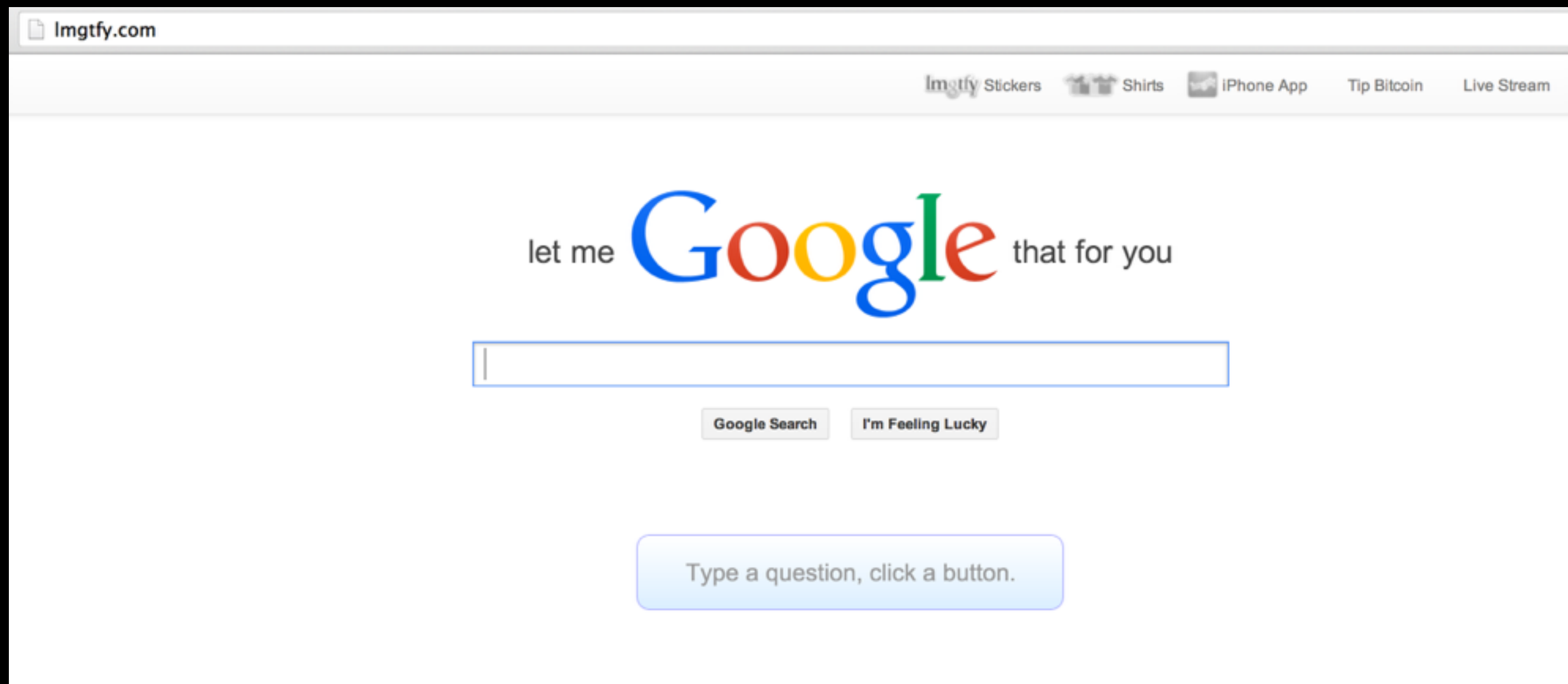
- There's a reason they're called programming languages
- You learn by practicing
- Will you be Shakespeare by the end of the course? No.
- But that's OK - even being able to find the bathroom in a foreign language can come in very handy.
- Don't be scared - just a few basics can let you do many useful things.

# Marking

- 60% practical - half tutorial problems, half final project.
  - Projects to be announced later, If there's something you would like to do for it, let me know.
  - would you prefer shorter tutorials due every class, or longer tutorials due once per week?
- 40% theory - test at end of term.

# Course Rule

- If you continue in school or get a nice job, reason you will be getting paid is for you to figure things out.
- Do your results make sense? Computer cannot decide that for you. If they don't, say so - grading will be much kinder.
- Feel free to email/ask me questions *\*after\** you have tried to figure it out on your own. Include the 3 things you searched for. Otherwise...



90% chance that's all I'll be doing anyways

# Corollary

- First rule of coding: Don't!
- Someone has probably spent their career doing (almost) exactly what you're trying to do. Go use what they did!
- Your code will (almost certainly) be slower, buggier, less flexible, and more painful to use.

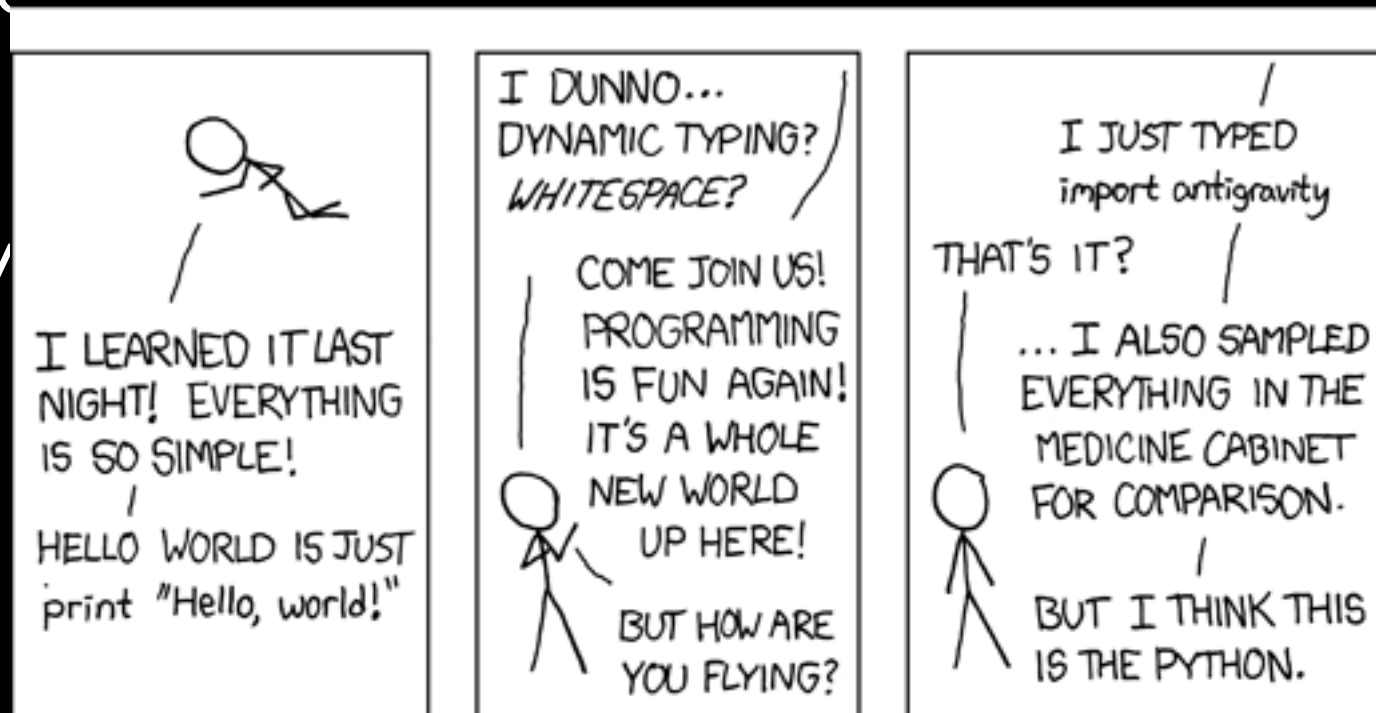
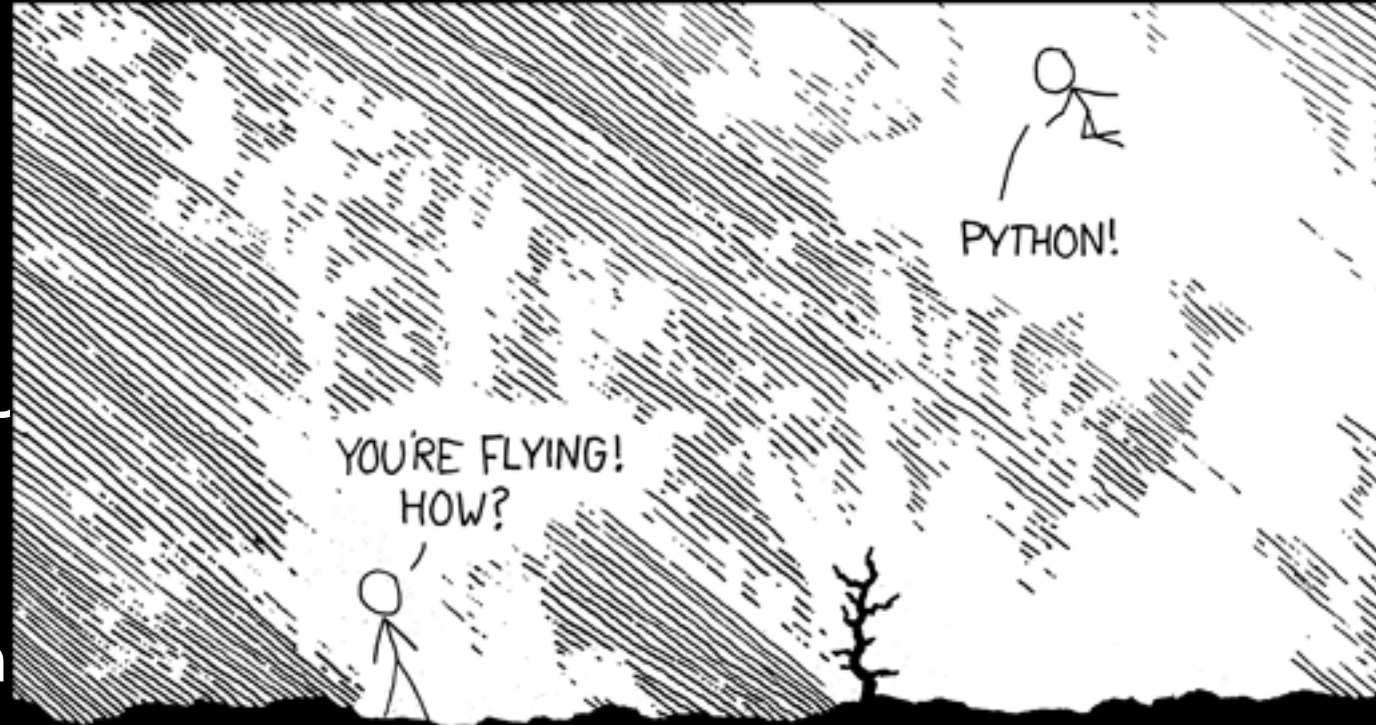
# Preliminary Outline

- Lecture 1: Linux/unix intro, filesystems, git
- Lecture 2: Python intro
- Lecture 3: Numpy/matplotlib/ more python
- Lecture 4: FFTs
- Lecture 5: Classes in python, complex numbers, overloading
- Lecture 6: N-body
- Lecture 7 Linear algebra/model fitting/MCMC
- Lecture 8: Advection/diffusion, 1-d PDEs
- Lecture 9; Fluid simulations, multi-d PDEs
- Lecture 10: Programming style, testing
- Lecture 11: Speeding up python - cython/C library interfaces
- Lecture 12: Test



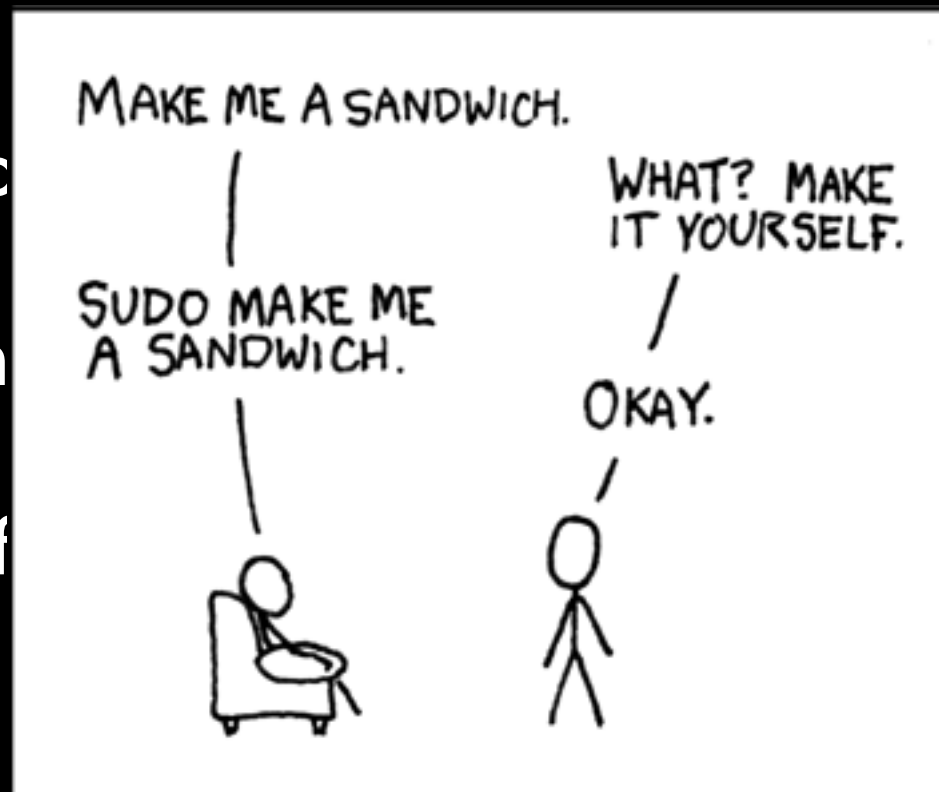
# Unix & Python

- Course will be in a Linux (a variety of Linux distributions) environment
- Course will mainly use python programming language
- Why python? Flexible language, open source, lots of things already written for it.
- Open source means wherever you go, you can find it and follow.



# Unix Intrinsically Multi-User

- Individual people have accounts in unix, with identifying user names. Each account has a password with it.
- Guiding principle - you aren't allowed to muck up someone else's account
- Every file has an owner. You own your own.
- Once you have a user name, you choose wisely...
- root is a special, all-powerful user. Be very careful when root. Don't break the OS. Be
- sudo (super-user do) executes commands as root. You will not have root/sudo access on these computers.



# Let's Get Everyone Accounts

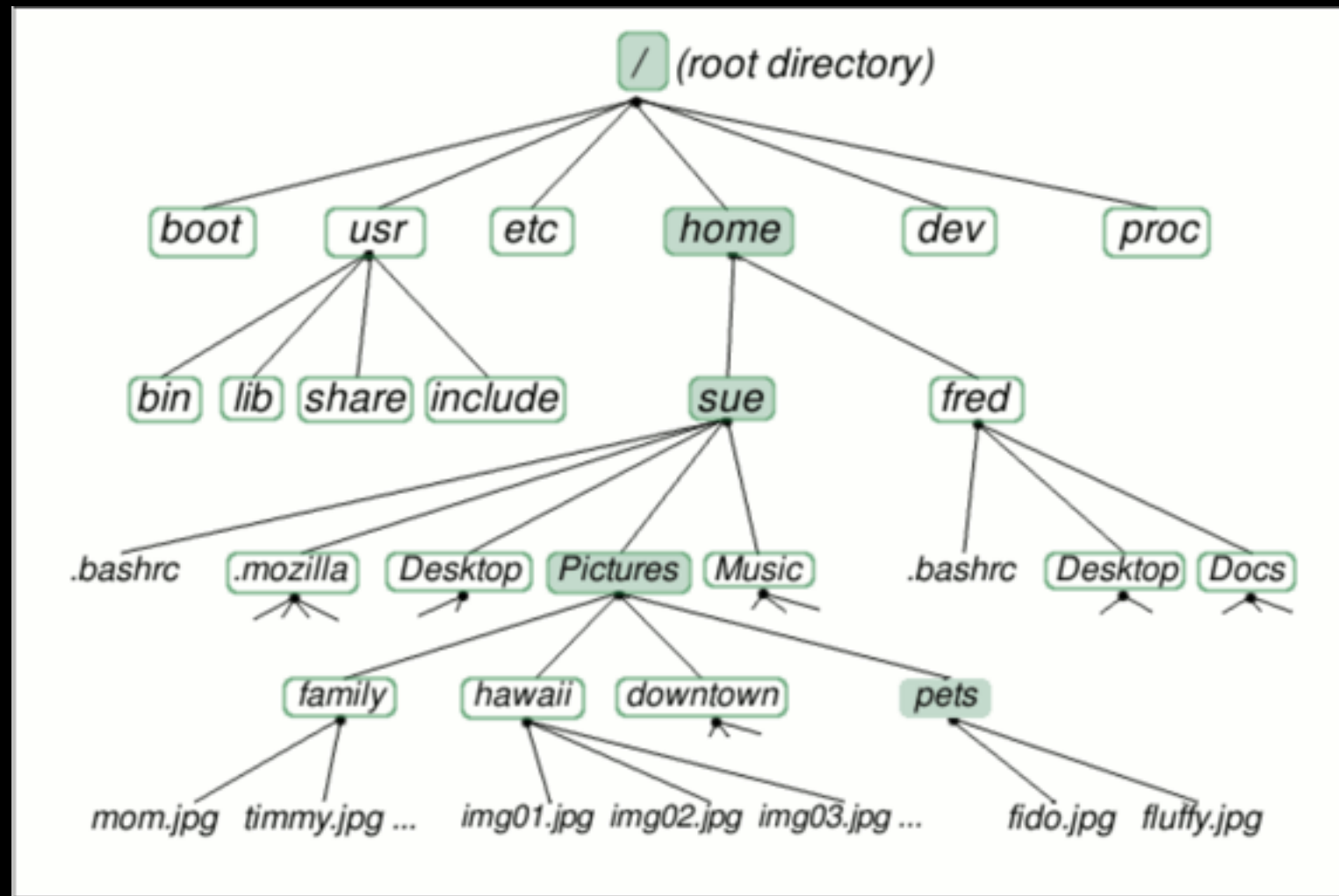
- Think of a user name and password.
- Let me know when you're ready and I'll make the account.
- Since accounts only on one machine, you should expect to sit at the same computer during the term.
- We'll wait until everyone is logged in as themselves.

# Terminal Time

- Unix's power lies in the command line. You can tell the computer to do most anything there.
- Let's all get a terminal started:
- Click on upper left ("dash"). Should get a search box.
- Type in "terminal" in the search box. Click to start.
- You might want to pin terminal to the sidebar.



# File Hierarchy



- All regular files live in directories.
- Directories arranged in trees.
- Every directory has a parent directory (except for the root directory)
- Directories help keep you organized. Use them!

# First Commands

- Your command line always has a location in the directory tree. Type “pwd” (print working directory) to find out where it is.
- See what files live in the directory - “ls” (list).
- Change directory of command lines - “cd <arg>” (change directory) where <arg> is the desired new directory.
- “..” always points to the parent directory. “cd ..” says “go up one level in the directory heirarchy.” “.” always points to the current directory.

# Commands are Flexible

- Many commands can do a variety of tasks through *options*.
- example: “ls -l” lists long-form information on files in directory
- “ls -a” lists hidden files.
- Commands can (usually) be combined: “ls -alh” says “list files (including hidden ones) in long-form. Report the sizes in human-readable format.
- Can you remember all the magic options? Me neither. man (manual) pages are installed on the system. Just type “man <command>” to learn about it.
- So, to learn more about the “man” command, type “man man”

# Looking at Files

- “cat” will dump the contents of a file to the screen.
- “more” will do the same thing, but one page at a time.
- “head” and “tail” will dump the first (or last) 10 lines of a file to the screen.



# Copying/moving/deleting

- “rm” (remove) will delete a file `rm foo.txt`
- “cp” (copy) will make an identical copy of a file: `cp foo.txt fwee.txt`
- `mkdir` will create a directory: `mkdir fwee`
- `rmdir` remove an empty directory.

# Wildcards

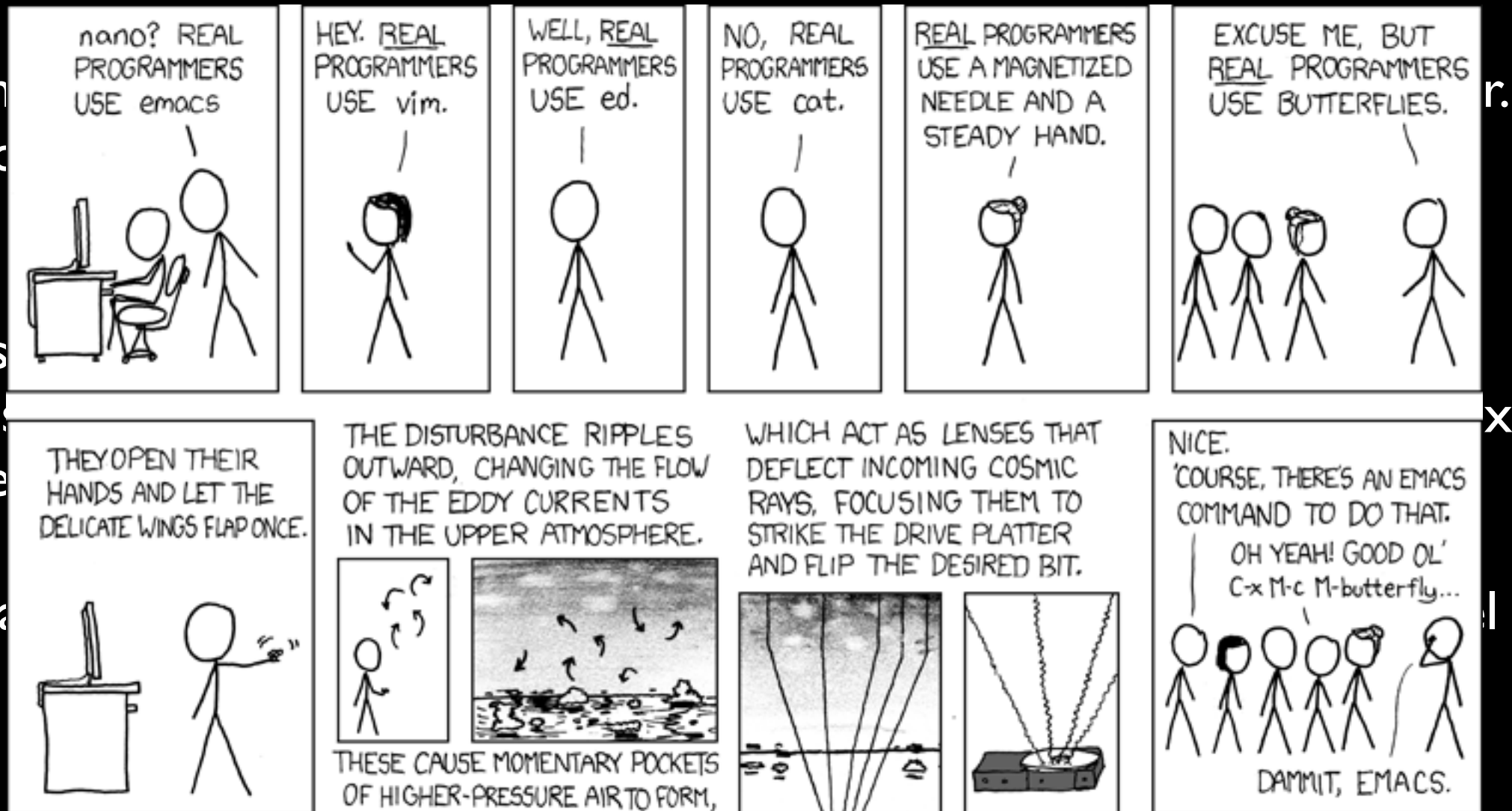
- The command line will let you specify subsets of files.
- “\*” matches any set of characters: “ls \*.txt” lists all files whose names end in “.txt”
- “?” matches any single character: “ls crud.?” will list all files that start with crud. followed by a single character. crud.9 would match, crud.l0 would not.
- [] matches any character inside the brackets. “ls set\_[0123].txt” would match set\_0.txt and set\_3.txt, but *\*not\** set\_4.txt.
- {} lists comma separated tags that will match. “ls \*.{txt,jpg}” will list all files whos names end in .txt *or* in .jpg

# Redirection

- A magic power of unix is the ability to redirect output into files or other commands.
- “|” (pipe) sends the output of one command into the next. “wc” (word count) will count the # of words in something. So, “ls -l \*.txt | wc” will tell you how many .txt files you have in a directory.
- “>” redirects output into a file. So, “ls -l > foo.txt” will list extended info about each file into the file “foo.txt”, which you can look at.
- “>>” is similar to “>”, but puts things at the end of the file. “echo” just repeats back its input. So, what would “echo thats all folks >> foo.txt” do?

# Text Editors

- At some point, one person says to others, "emacs changed, followed by..."
- There are a lot of people free to...



# Environment Variables

- Normally you will have variables defined that guide what happens at the command line.
- “env” or “printenv” will normally print them to the screen.
- Some important ones include :“PATH” - tells the command line where to search for commands to execute. “HOME” - tells you where your home directory is - this is where your files live. (Note that environment variables are case sensitive.)
- values of variables can be accessed with a “\$” sign. “echo” is like a print command, so to see the location of your home directory, type “echo \$HOME”.
- At UKZN, “http\_proxy” is another important one - this tells the system how to get through the proxy server.

# Let's Make a (not) Useful File

- Whenever you start a new shell, a resource file is used. For us, that will be “\${HOME}/.bashrc” (bourne-again shell resources).
- Let's make it so we could get through the proxy. Let's set “http\_proxy” so we can get out from the command line.
- “export http\_proxy=http://<username>:<password>@bcproxy.ukzn.ac.za:8080”.
- NEVER write your password in plain text. Put something fake in for now, but you would have to use real password.
- You can put this in your .bashrc. How would you check if new terminals register this?
- ftp\_proxy and https\_proxy should also be set to the same value.

# Finally, Version Control

- Let's role play.

# Version Control ctd.

- What did we just learn? It's important to be able to undo changes
- Merging changes from different places/different people important. Should be done in a systematic way. If I change one part of a file while you change another, would like to combine those changes automatically.
- What happens if someone steals your computer? The more places code lives, the safer you are.



# Version Control 3

- Several version control (VC) systems exist, 3 most common are CVS, SVN, and git. We will use git.
- VC lets many people edit files. A set of files being tracked is called a “repository.” A VC system can keep your local files in a repository synced with the central repository.
- When you are done making changes, you “commit” them to your local repository, then “push” them to the central repository. When someone else makes changes, you “pull” them from the central repository.
- If you both edited the same part of a file, the VCS will throw an error and tell you to fix the conflict by hand.
- VCSs usually require your local repository to be up-to-date with the central one before you’re allowed to push changes.

# Git

- git was developed by Linus Torvalds (the creator of Linux)
- very powerful
- Basic command `git add <files>` will add files to the repository
- When ready, `git commit -m "message"` will commit your repository
- “git history” will show the history of the repository
- “git revert” will revert the repository to a previous version

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.



Li” in linux)

empty repository. “git add  
ory.

>” “ will commit your changes to

a saved in your commits.

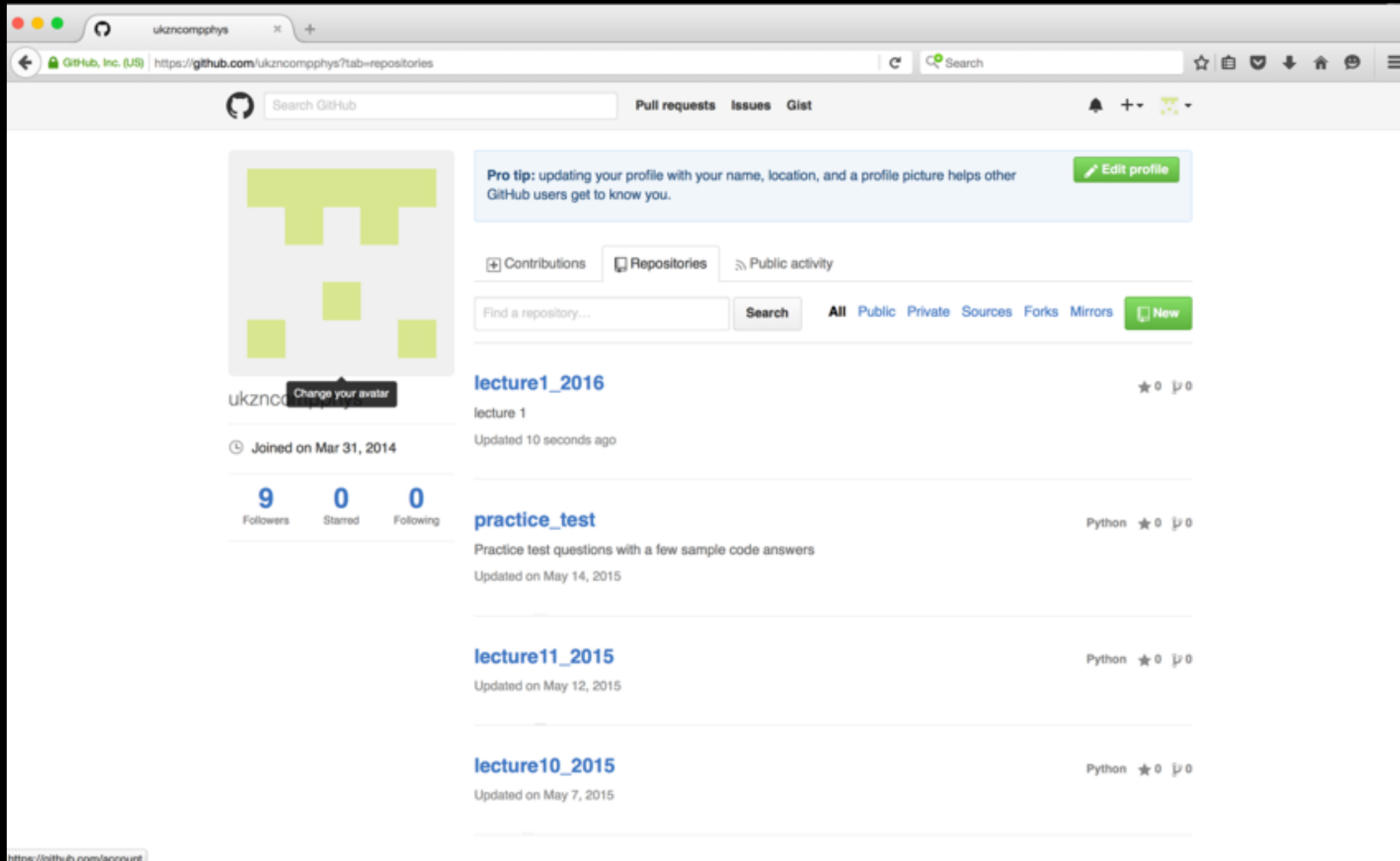
ersion of the repository.

If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of 'It's really pretty simple, just think of branches as...' and eventually you'll learn the commands that will fix everything.” <https://xkcd.com/1597/>

# Github

- Github provides free repository support. Much of the code released today is done on github.
- You can make an account, then github will walk you through the steps you need to do what you want.
- “git clone” will copy a repository. I have made a github page for this class - you can get the slides plus a linux tutorial cheat sheet there.
- In the browser, go to “<https://github.com/ukzncompphys>”

# Click on Repositories, then lecture 1\_2016



The screenshot shows a web browser window with the URL `https://github.com/ukzncompphys?tab=repositories`. The page displays the GitHub profile for user `ukzncompphys`. The profile includes a placeholder avatar, a bio, and statistics: 9 followers, 0 starred repositories, and 0 following. The 'Repositories' tab is selected, showing a list of repositories. The first repository is `lecture1_2016`, described as 'lecture 1', updated 10 seconds ago. Below it is `practice_test`, described as 'Practice test questions with a few sample code answers', updated on May 14, 2015. Further down are `lecture11_2015` and `lecture10_2015`, both described as 'Python' and updated on May 12, 2015 and May 7, 2015 respectively. A 'Pro tip' banner at the top right suggests updating the profile.

ukzncompphys

Pro tip: updating your profile with your name, location, and a profile picture helps other GitHub users get to know you. [Edit profile](#)

[Contributions](#) **Repositories** [Public activity](#)

Find a repository... [Search](#) [All](#) [Public](#) [Private](#) [Sources](#) [Forks](#) [Mirrors](#) [New](#)

**lecture1\_2016** ★ 0 🍴 0  
lecture 1  
Updated 10 seconds ago

**practice\_test** Python ★ 0 🍴 0  
Practice test questions with a few sample code answers  
Updated on May 14, 2015

**lecture11\_2015** Python ★ 0 🍴 0  
Updated on May 12, 2015

**lecture10\_2015** Python ★ 0 🍴 0  
Updated on May 7, 2015

<https://github.com/account>

# Clone URL Here

ukzncompphys / lecture1\_2016

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Wiki Pulse Graphs Settings

lecture 1 — Edit

2 commits 1 branch 0 releases 1 contributor

Branch: master New pull request

New file Upload files Find file HTTPS https://github.com/ukzn Download ZIP

sievers added slides. Latest commit 8902304 37 seconds ago

compphys_1.pdf	added slides.	36 seconds ago
linux_boot_camp.pdf	initial commit	4 minutes ago

Help people interested in this repository understand your project by adding a README. Add a README

© 2016 GitHub, Inc. Terms Privacy Security Contact Help

Status API Training Shop Blog About

# Then Clone the Repository

- “git clone <link from github>” will get you the slides.
- [https://github.com/ukzncompphys/lecture1\\_2016.git](https://github.com/ukzncompphys/lecture1_2016.git) is the link in case you can't get to it.
- If this doesn't work, you are probably having proxy issues.

You might want to stay at same computer during course.  
However, if you haven't, just can just “git pull/clone” from a different machine and be set up in seconds.

# Tutorial Problems

- How can you list all files in a directory, with the most recent one displayed last? (5)
- How can you display the first 25 lines of a file? How can you display just lines 16 through 25? (5)
- Sometimes there are multiple man pages with the same name. How can you display them all? (5)
- Make a new text file with your name, email address, what you're working on for your honours project, plus a few things you'd like to learn in the course. Initialize a git repository and commit this file. (5)
- Put your answers to the other tutorial questions into another text file. Add this to the repository also. (5)
- Make a github account and push the repository onto github. Email me your github name so I can have a look at your file/answers. (10)