

Random numbers, their generation, and their peculiarities

PHYS 250 (Autumn 2025) – Lecture 2

David Miller

Department of Physics and the Enrico Fermi Institute
University of Chicago

October 2, 2025

Outline

1 *Quick **git**/**GitHub** tutorial*

- Basics of **git**
- **git** workflow
- Our usage of **git** and **GitHub** resources

2 *Plan for homework*

- Using **GitHub** Classroom

3 *Physics*

- Random numbers: Introduction and motivation
- Types of random numbers
- Hypothesis testing and random numbers

Version control reminders



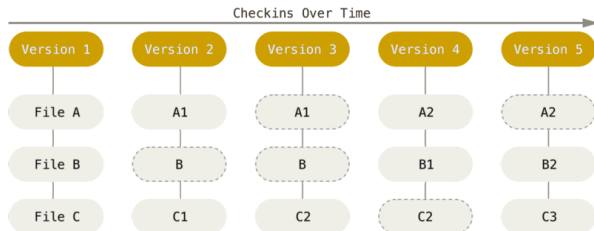
We will be using these tools for the course. There are **many** awesome tutorials out there, and here are a few of my favorite bookmarks:

- “Hello World” from **GitHub**
- An Intro to Git and GitHub for Beginners (Tutorial)
- A Visual Git Reference
- An Intro to Git and GitHub

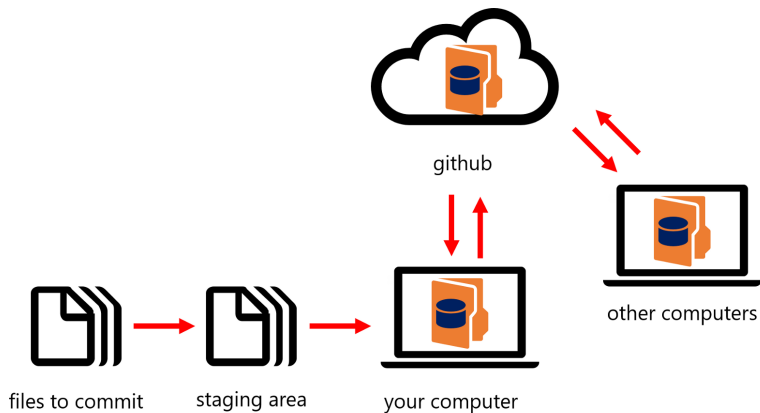
A basic summary of the concept of **git**, in my words, is:

git takes a snapshot of the entire “project” and saves it, kind of like running Time Machine (or any disk backup) on your entire hard drive every time.

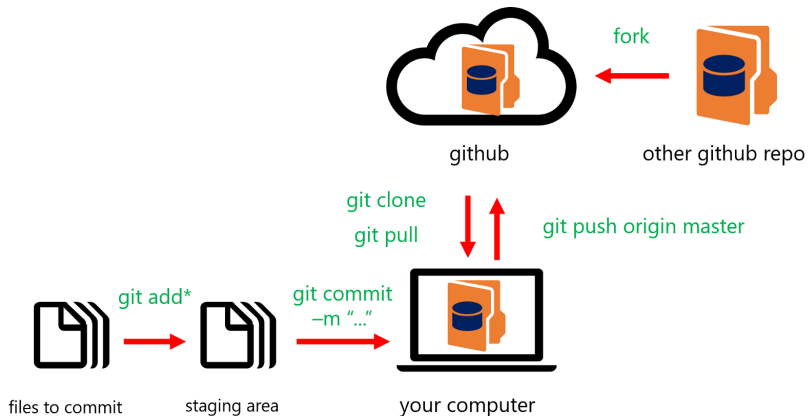
git concept in pictures



What does a **git** work flow look like?



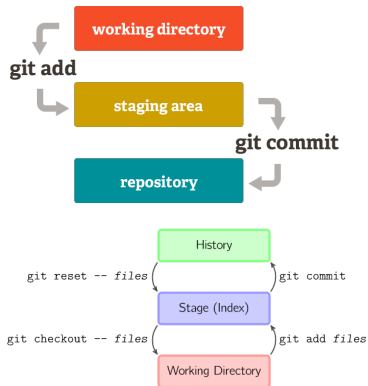
What does a **git** work flow look like?



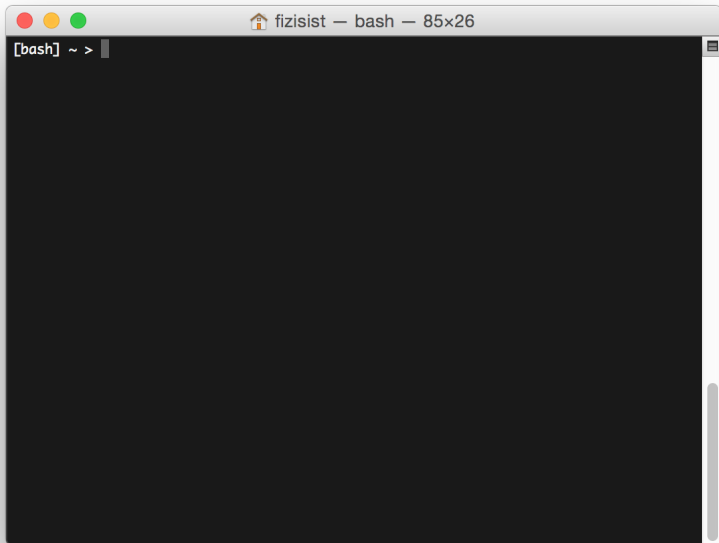
What does a **git** work flow look like?

The five commands you use the most are:

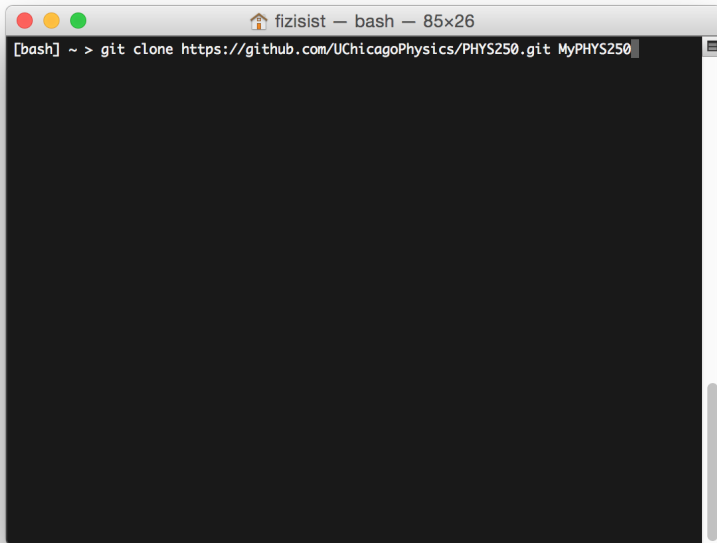
- `git add files`: copies *files* (at their current state) to the stage.
- `git commit -m "MESSAGE"`: saves a snapshot of the stage as a commit.
- `git reset files`: copies *files* from the latest commit to the stage.
 - Use this command to “undo” a `git add files`. You can also `git reset` to unstage everything.
- `git checkout files`: copies *files* from the stage to the working directory. Use this to throw away local changes.
- `git push`: send everything in your local repository back to the master



git workflow example: edit HelloGaussian.py



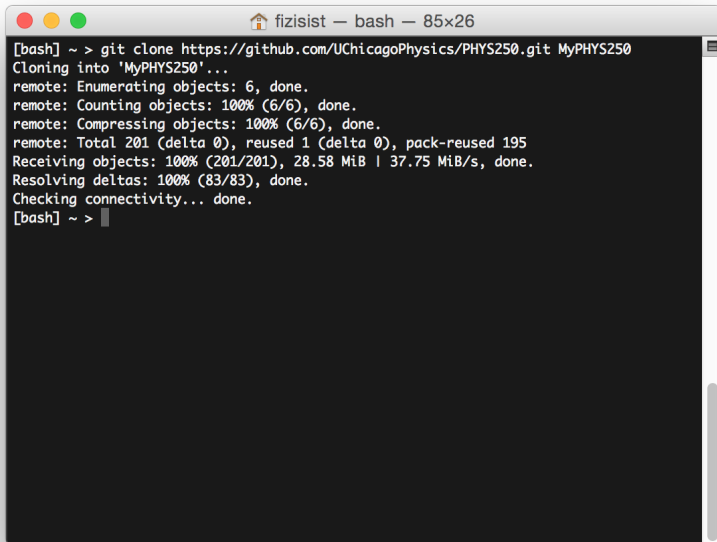
git workflow example: edit HelloGaussian.py



A terminal window with a light gray title bar containing three colored window control buttons (red, yellow, green) on the left and a title "fizisist — bash — 85x26" in the center. The terminal area has a black background with white text. The prompt "[bash] ~ >" is followed by the command "git clone https://github.com/UChicagoPhysics/PHYS250.git MyPHYS250". A white cursor is positioned at the end of the command line. A vertical scrollbar is visible on the right side of the terminal window.

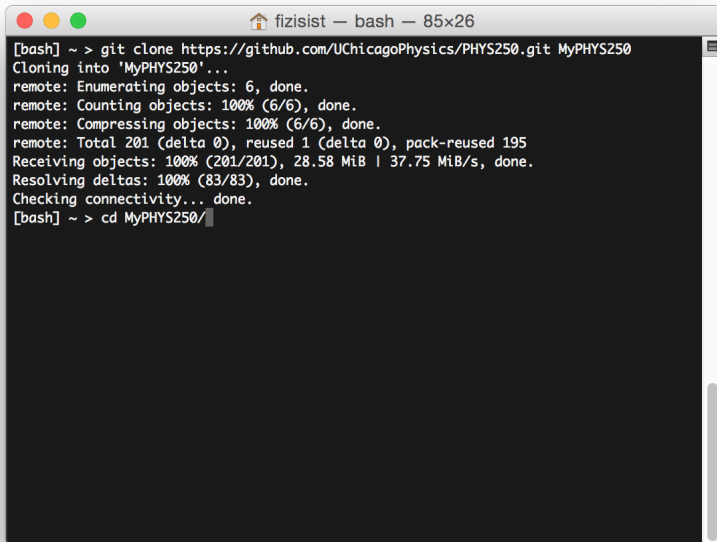
```
[bash] ~ > git clone https://github.com/UChicagoPhysics/PHYS250.git MyPHYS250
```

git workflow example: edit HelloGaussian.py

A terminal window titled 'fizisist — bash — 85x26' with standard macOS window controls (red, yellow, green buttons). The terminal shows the execution of a git clone command to create a local repository from a GitHub repository. The output shows progress for enumerating, counting, and compressing objects, as well as receiving and resolving deltas. The process completes successfully, and the prompt returns to the user.

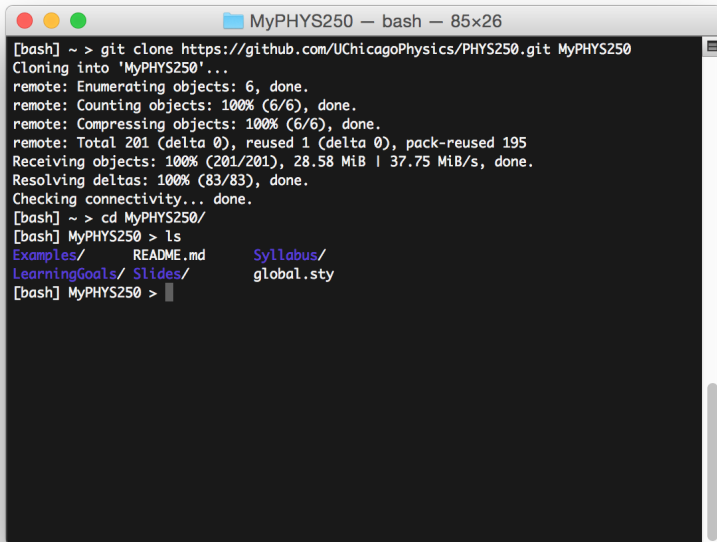
```
[bash] ~ > git clone https://github.com/UChicagoPhysics/PHYS250.git MyPHYS250
Cloning into 'MyPHYS250'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 201 (delta 0), reused 1 (delta 0), pack-reused 195
Receiving objects: 100% (201/201), 28.58 MiB | 37.75 MiB/s, done.
Resolving deltas: 100% (83/83), done.
Checking connectivity... done.
[bash] ~ > 
```

git workflow example: edit HelloGaussian.py

A terminal window titled 'fizisist — bash — 85x26' with standard macOS window controls (red, yellow, green buttons). The terminal shows the execution of a git clone command to create a local repository from a GitHub repository. The output shows the cloning process, including object enumeration, counting, and compression, all completed successfully. The prompt then changes to the new directory.

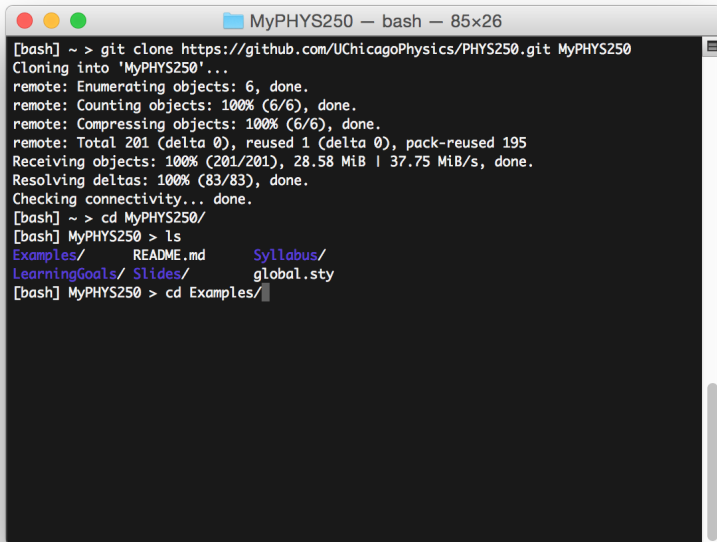
```
[bash] ~ > git clone https://github.com/UChicagoPhysics/PHYS250.git MyPHYS250
Cloning into 'MyPHYS250'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 201 (delta 0), reused 1 (delta 0), pack-reused 195
Receiving objects: 100% (201/201), 28.58 MiB | 37.75 MiB/s, done.
Resolving deltas: 100% (83/83), done.
Checking connectivity... done.
[bash] ~ > cd MyPHYS250/
```

git workflow example: edit HelloGaussian.py

A terminal window titled "MyPHYS250 — bash — 85x26" with standard macOS window controls (red, yellow, green buttons). The terminal shows the execution of a git clone command to create a local repository from a GitHub repository. The output shows the cloning process, including object enumeration, counting, and compression. After cloning, the user navigates to the new directory and lists its contents, showing a directory structure with subdirectories and files.

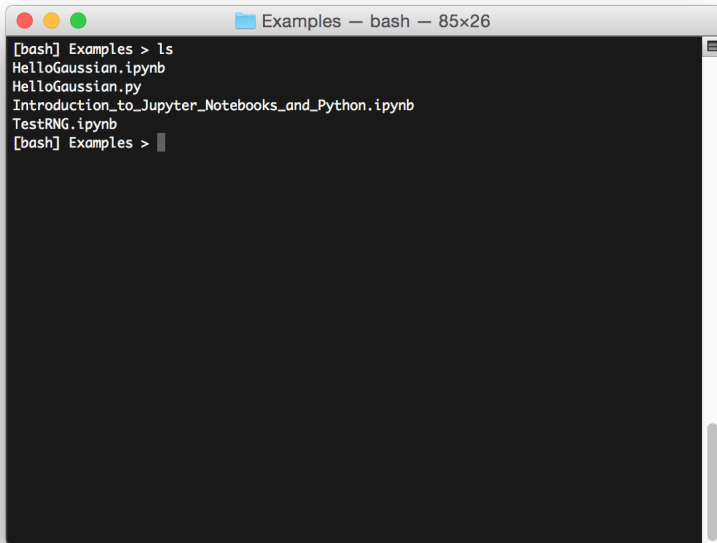
```
[bash] ~ > git clone https://github.com/UChicagoPhysics/PHYS250.git MyPHYS250
Cloning into 'MyPHYS250'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 201 (delta 0), reused 1 (delta 0), pack-reused 195
Receiving objects: 100% (201/201), 28.58 MiB | 37.75 MiB/s, done.
Resolving deltas: 100% (83/83), done.
Checking connectivity... done.
[bash] ~ > cd MyPHYS250/
[bash] MyPHYS250 > ls
Examples/      README.md      Syllabus/
LearningGoals/ Slides/        global.sty
[bash] MyPHYS250 > █
```

git workflow example: edit HelloGaussian.py

A terminal window titled "MyPHYS250 — bash — 85x26" with a standard macOS window header (red, yellow, green buttons). The terminal shows the process of cloning a repository from GitHub. The output of the commands is as follows:

```
[bash] ~ > git clone https://github.com/UChicagoPhysics/PHYS250.git MyPHYS250
Cloning into 'MyPHYS250'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 201 (delta 0), reused 1 (delta 0), pack-reused 195
Receiving objects: 100% (201/201), 28.58 MiB | 37.75 MiB/s, done.
Resolving deltas: 100% (83/83), done.
Checking connectivity... done.
[bash] ~ > cd MyPHYS250/
[bash] MyPHYS250 > ls
Examples/      README.md      Syllabus/
LearningGoals/ Slides/       global.sty
[bash] MyPHYS250 > cd Examples/
```

git workflow example: edit HelloGaussian.py

A terminal window titled "Examples — bash — 85x26" with a dark background. The window shows the output of a 'ls' command, listing four files: HelloGaussian.ipynb, HelloGaussian.py, Introduction_to_Jupyter_Notebooks_and_Python.ipynb, and TestRNG.ipynb. The prompt is currently at the end of the last line.

```
[bash] Examples > ls
HelloGaussian.ipynb
HelloGaussian.py
Introduction_to_Jupyter_Notebooks_and_Python.ipynb
TestRNG.ipynb
[bash] Examples > 
```

git workflow example: edit HelloGaussian.py

```
import numpy as np
import matplotlib.pyplot as plt

def p(x):
    return np.exp(-x**2)

#let's plot it
x = np.linspace(-3,3,100)
y = p(x)
plt.plot(x,y)
plt.show()

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

"HelloGaussian.py" [noeol] 11L, 166C

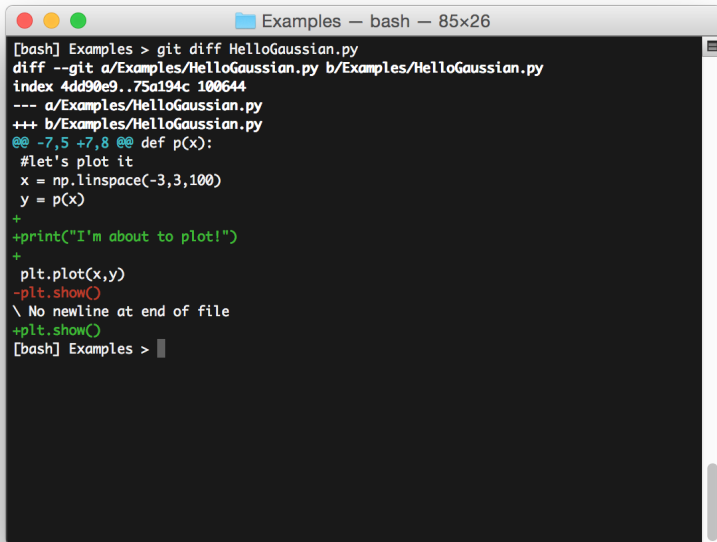
git workflow example: edit HelloGaussian.py

A screenshot of a terminal window titled "Examples — vim — 85x26". The terminal displays the following Python code:

```
import numpy as np  
import matplotlib.pyplot as plt  
  
def p(x):  
    return np.exp(-x**2)  
  
#let's plot it  
x = np.linspace(-3,3,100)  
y = p(x)  
  
print("I'm about to plot!")  
  
plt.plot(x,y)  
plt.show()  
  
~  
~  
~  
~  
~  
~  
~  
~  
~
```

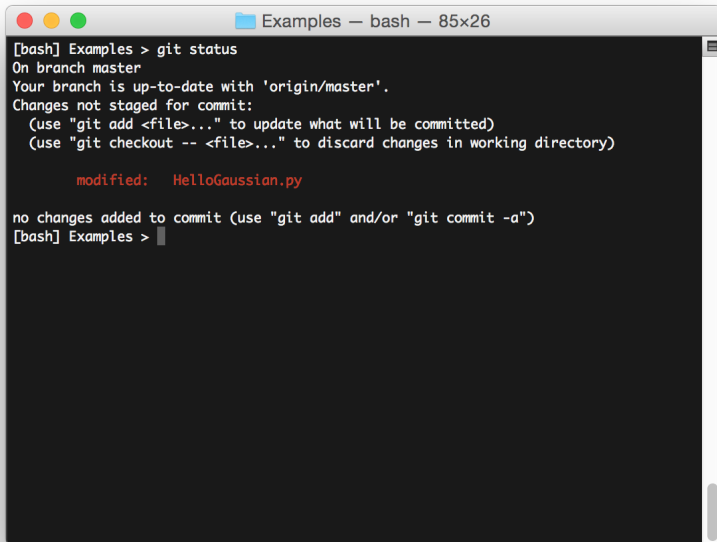
The prompt "`"HelloGaussian.py"`" followed by column numbers "14L, 197C" is visible at the bottom left.

git workflow example: edit HelloGaussian.py

A terminal window titled "Examples — bash — 85x26" with standard macOS window controls (red, yellow, green buttons). The terminal displays the output of the command "git diff HelloGaussian.py". The output shows a diff between the current file and a previous version, highlighting changes in a Gaussian function definition and its plotting. The changes include adding a print statement and a plt.show() call, and removing another plt.show() call. The terminal text is as follows:

```
[bash] Examples > git diff HelloGaussian.py
diff --git a/Examples/HelloGaussian.py b/Examples/HelloGaussian.py
index 4dd90e9..75a194c 100644
--- a/Examples/HelloGaussian.py
+++ b/Examples/HelloGaussian.py
@@ -7,5 +7,8 @@ def p(x):
    #let's plot it
    x = np.linspace(-3,3,100)
    y = p(x)
+
+print("I'm about to plot!")
+
    plt.plot(x,y)
-plt.show()
\ No newline at end of file
+plt.show()
[bash] Examples > 
```

git workflow example: edit HelloGaussian.py

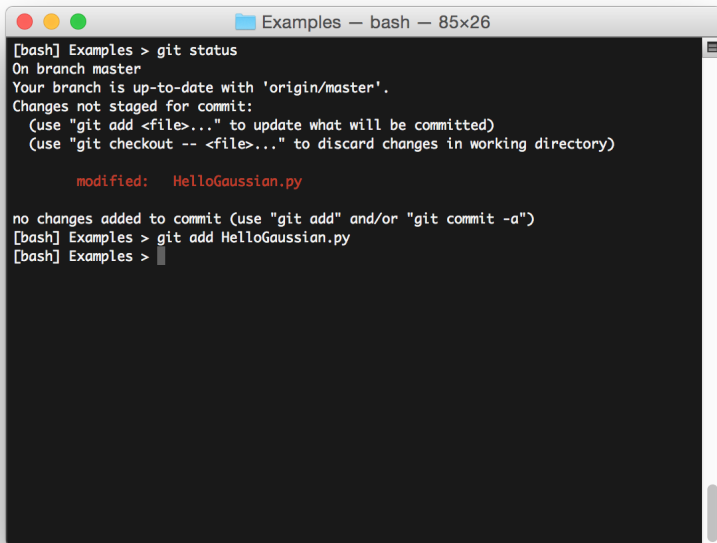
A terminal window titled "Examples — bash — 85x26" with a dark background. It shows the output of the 'git status' command. The output indicates the user is on the 'master' branch, which is up-to-date with 'origin/master'. It lists 'HelloGaussian.py' as a modified file. At the bottom, it says 'no changes added to commit' and shows the prompt '[bash] Examples >'.

```
[bash] Examples > git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   HelloGaussian.py

no changes added to commit (use "git add" and/or "git commit -a")
[bash] Examples > 
```

git workflow example: edit HelloGaussian.py

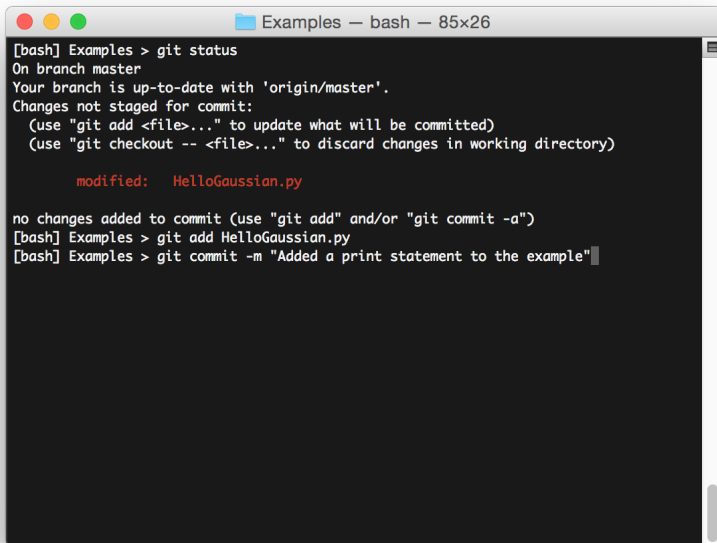


```
[bash] Examples > git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   HelloGaussian.py

no changes added to commit (use "git add" and/or "git commit -a")
[bash] Examples > git add HelloGaussian.py
[bash] Examples > 
```

git workflow example: edit HelloGaussian.py

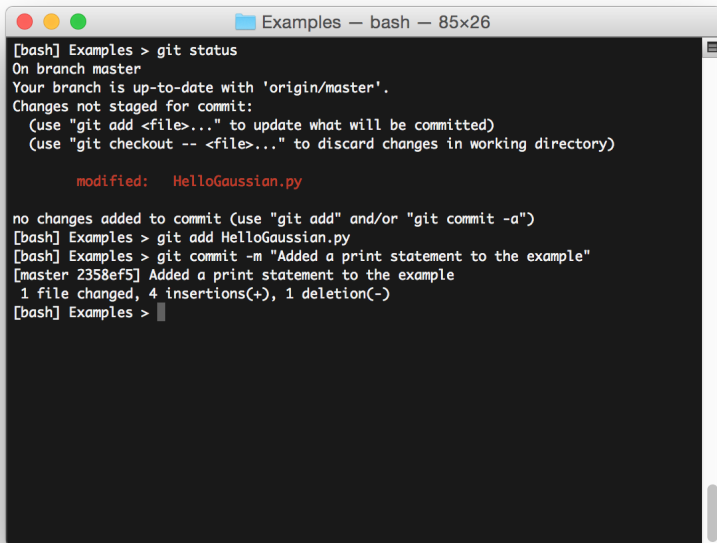
A terminal window titled "Examples — bash — 85x26" with standard macOS window controls (red, yellow, green buttons). The terminal output shows the execution of git status, git add, and git commit commands. The status output indicates that HelloGaussian.py is modified. The commit message is "Added a print statement to the example".

```
[bash] Examples > git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   HelloGaussian.py

no changes added to commit (use "git add" and/or "git commit -a")
[bash] Examples > git add HelloGaussian.py
[bash] Examples > git commit -m "Added a print statement to the example"
```

git workflow example: edit HelloGaussian.py

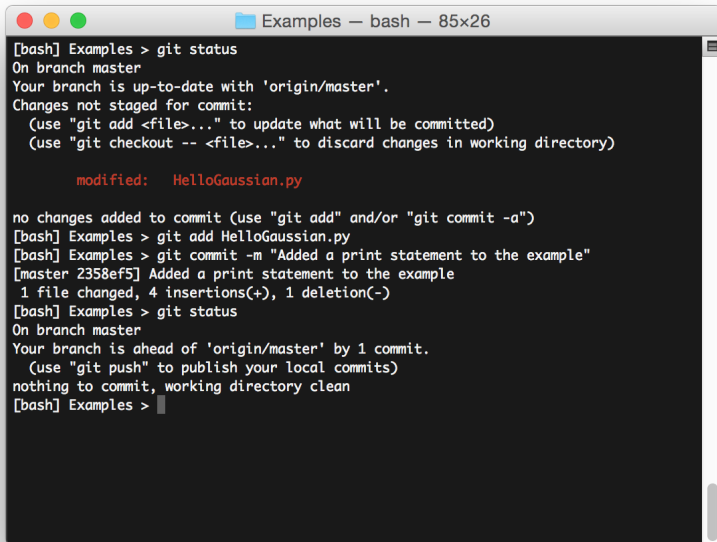


```
[bash] Examples > git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   HelloGaussian.py

no changes added to commit (use "git add" and/or "git commit -a")
[bash] Examples > git add HelloGaussian.py
[bash] Examples > git commit -m "Added a print statement to the example"
[master 2358ef5] Added a print statement to the example
 1 file changed, 4 insertions(+), 1 deletion(-)
[bash] Examples >
```

git workflow example: edit HelloGaussian.py

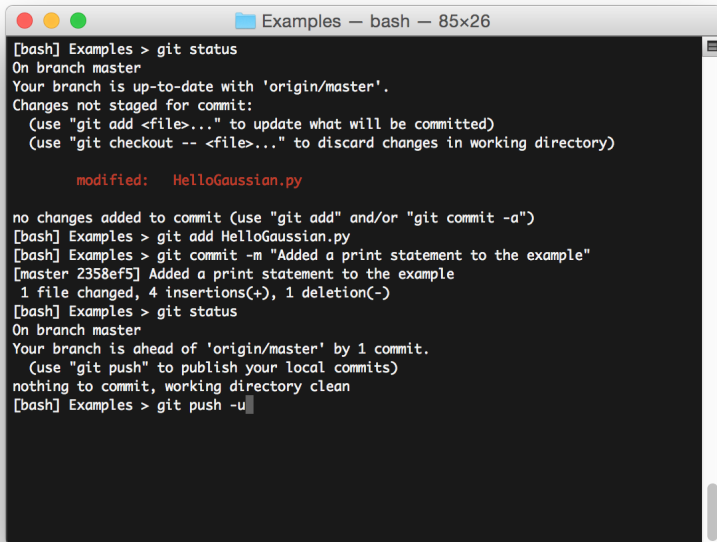
A terminal window titled "Examples — bash — 85x26" with a macOS-style title bar (red, yellow, green buttons). The terminal shows the following commands and output:

```
[bash] Examples > git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   HelloGaussian.py

no changes added to commit (use "git add" and/or "git commit -a")
[bash] Examples > git add HelloGaussian.py
[bash] Examples > git commit -m "Added a print statement to the example"
[master 2358ef5] Added a print statement to the example
 1 file changed, 4 insertions(+), 1 deletion(-)
[bash] Examples > git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
[bash] Examples > █
```

git workflow example: edit HelloGaussian.py

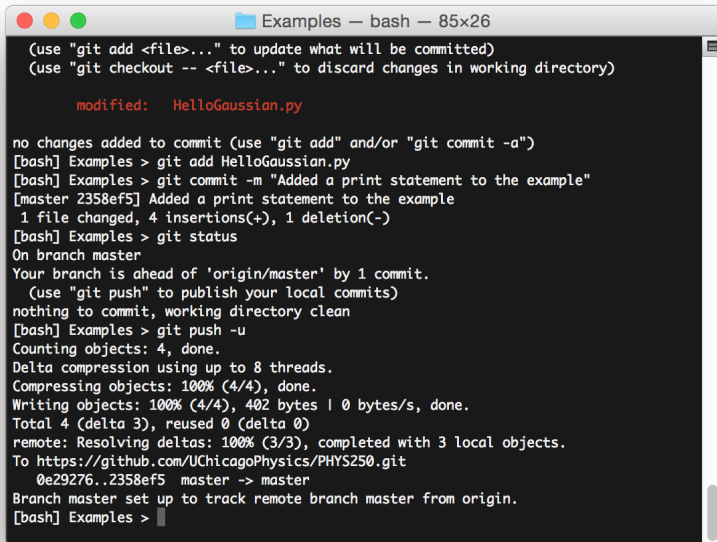
A terminal window titled "Examples — bash — 85x26" with a macOS-style title bar (red, yellow, green buttons). The terminal shows a sequence of git commands and their output. The output includes status messages, file modification details, and commit information. The file "HelloGaussian.py" is highlighted in red in the output.

```
[bash] Examples > git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   HelloGaussian.py

no changes added to commit (use "git add" and/or "git commit -a")
[bash] Examples > git add HelloGaussian.py
[bash] Examples > git commit -m "Added a print statement to the example"
[master 2358ef5] Added a print statement to the example
 1 file changed, 4 insertions(+), 1 deletion(-)
[bash] Examples > git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
[bash] Examples > git push -u
```

git workflow example: edit HelloGaussian.py

A terminal window titled "Examples — bash — 85x26" with a dark background and light text. It shows a sequence of git commands and their outputs. The user adds a file, commits it with a message, checks the status, and pushes it to a remote repository. The output shows the commit being successful and the branch being updated.

```
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)

    modified:   HelloGaussian.py

no changes added to commit (use "git add" and/or "git commit -a")
[bash] Examples > git add HelloGaussian.py
[bash] Examples > git commit -m "Added a print statement to the example"
[master 2358ef5] Added a print statement to the example
 1 file changed, 4 insertions(+), 1 deletion(-)
[bash] Examples > git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
[bash] Examples > git push -u
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 402 bytes | 0 bytes/s, done.
Total 4 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/UChicagoPhysics/PHYS250.git
   0e29276..2358ef5  master -> master
Branch master set up to track remote branch master from origin.
[bash] Examples >
```


PHYS 250 *GitHub*

<https://github.com/UChicagoPhysics/PHYS250>

Course materials are hosted in the **GitHub** UChicagoPhysics repository

UChicagoPhysics / **PHYS250** Watch 0 Star 0 Fork 0

<> Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

University of Chicago PHYS 250 Computational Physics software repository Edit

Manage topics

15 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

fizisist Update example		Latest commit 0f09be5 25 minutes ago
Examples	Update example	25 minutes ago
LearningGoals	Update Learning Goals	22 hours ago
Slides	Update Lecture 1 Slides	an hour ago
Syllabus	Updates to syllabus	22 hours ago
.gitignore	Update slides for day 1	3 days ago
README.md	Update README.md	3 days ago

- Slides (e.g. *these!*), syllabi, learning goals, and code examples
- Stable versions will be cross-posted to **Canvas** as well.
- Homework submission will be done via **GitHub** (*instructions to come*)

Outline

1 *Quick **git**/**GitHub** tutorial*

- Basics of **git**
- **git** workflow
- Our usage of **git** and **GitHub** resources

2 *Plan for homework*

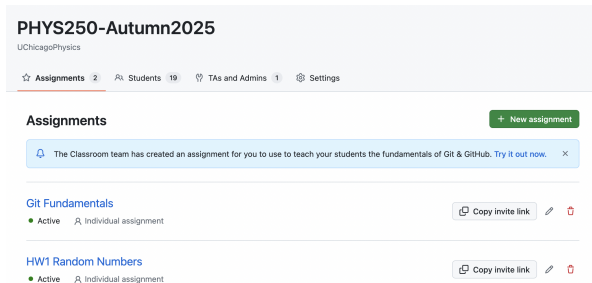
- Using **GitHub** Classroom

3 *Physics*

- Random numbers: Introduction and motivation
- Types of random numbers
- Hypothesis testing and random numbers

GitHub Classroom: assignments (I)

As mentioned on Tuesday, we will be using **GitHub** for distribution, assessment, and collection of assignments.



HW1: <https://classroom.github.com/a/xhy9sCAY>
Git Practice: <https://classroom.github.com/a/Lb3oG6gf>

GitHub Classroom: assignments (II)

- You will receive a link to the assignment
 - <https://classroom.github.com/a/xhy9sCAY>
- This provides your own unique repository based on a starter repository with examples and info that you might find useful for the assignment.
- The deadline will be **Thursday at 2pm**, at which time **GitHub Classroom** will save the latest commit from each repository as a submission.
- Submission commits are viewable **only to me and the TAs** on the assignment page.
- We can then make comments and grades directly on your submission.

Outline

1 *Quick **git**/**GitHub** tutorial*

- Basics of **git**
- **git** workflow
- Our usage of **git** and **GitHub** resources

2 *Plan for homework*

- Using **GitHub** Classroom

3 *Physics*

- Random numbers: Introduction and motivation
- Types of random numbers
- Hypothesis testing and random numbers

Random numbers

Random numbers may seem innocuous, but they underlie nearly everything you do electronically, rely on for security, or employ in the context of simulations and evaluation of models.

- **cryptography**
- **computer simulation**
 - most well-known of which is named after gambling town, **Monte Carlo**
- **sampling**

As such, their use, their control, and the evaluation of just how random they are, are of **paramount importance for computational physics.**

And you thought lava lamps were a thing of the past



Cryptography relies on the ability to generate random numbers that are both unpredictable and secret. But **random** is a very malleable term.

True vs. Pseudo: it's more than semantics

The computers of today are, by design and by implementation, *deterministic*. That means that an **algorithm cannot**, on its own and without input from an external source, provide a stream of 100% uncorrelated random sequences. Hence the division:

- **TRNGs: True** random number generators
 - generally use some physical process that is unpredictable, but often slow and requires specialized hardware
 - possibly combined with some compensation mechanism that might remove any bias in the process
 - examples include: lava lamps, quantum processes, radioactive decays, thermal noise
- **PRNGs: Pseudo**-random number generators
 - based on algorithms and, therefore, not truly *random*
 - do not require special hardware and therefore are very portable and fast
 - can be reproduced given initial conditions

We will, perhaps obviously, focus on PRNGs.

(but if you want to build a wall of lava lamps in KPTC, I will cheer you on)

Pseudo-random number generators (PRNGs)

The idea behind an algorithmic PRNG is to generate a sequence of numbers, $x_1, x_2, x_3 \dots$ using a recurrence of the form

$$x_i = f(x_{i-1}, x_{i-2}, \dots, x_{i-n}), \quad (1)$$

where n initial numbers (“seeds”) are needed to begin the recurrence. The magic lies in the function, f , used, and the resulting *uniformity* and *correlation length* across some sequence of numbers.

Linear congruential generators (LCGs) are one of the oldest PRNGs and have the form

$$x_{i+1} = (ax_i + c) \mod m \quad (2)$$

IBM mainframes in the 60s had $a = 65539$, $c = 0$, and $m = 2^{31}$. This leads to

$$x_{i+2} = 6x_{i+1} - 9x_i \quad (3)$$

which, maybe you can tell, isn't great.

Python random number generator: Mersenne Twister

Python has a built-in (`stdlib`) RNG that can be accessed with:

```
import random as rng  
rng.random()
```

`numpy` has an even more developed set of tools with `numpy.random`.

```
from numpy import random as rng  
rng.random()
```

Both of these use the **Mersenne Twister algorithm**, developed in 1997 by Matsumoto and Nishimura; is a version of a generalized feedback shift register PRNG. The name due to the fact that the period is given by a Mersenne prime (most commonly: $n = 2^{19937}$):

$$M_n = 2^n - 1, n \in \mathbb{N} \quad (4)$$

(In preparing this, I found out that the largest known prime number $2^{77,232,917} - 1$ is a Mersenne prime, found on December 26, 2017.)

Testing the true “randomness” of a RNG

Tests of RNGs must look for patterns in sequences of given lengths and frequencies and test those possible patterns against the probability that they occurred “accidentally” or whether they are happening more often than they **should**.

→ This brings us to our first example of **hypothesis testing**

Big business for RNGs:

Acert Research Lab													
Random Number Generator Results													
Show only: Any Generator with a at least: 0 B/s and: any quality: No problems Potentially deterministic Not random Test failed, but not relevant Legend:													
Generator	Vendor	Speed	Price	Upload Size	Entropy (->8)	Birthday Spacing	Matrix Ranks	6x8 Matrix Ranks	Minimum Distance Test	Random Spheres Test	The Squeeze Test	Overlapping Sums Test	Submission YYYY-MM
Meigo	Ribeiro Alvo	200000000 B/s	0 USD	365 MiB	8.000000	0.841970	0.802	0.060	0.160384	0.766411	0.749666	0.022813	2014-02
libfmrng, test #12	post-factum	40000 B/s	0 USD	372 MiB	8.000000	0.020897	0.096	0.970	0.235635	0.879316	0.877657	0.056526	2014-02
libfmrng, test #13	post-factum	40000 B/s	0 USD	423 MiB	8.000000	0.796904	0.518	0.034	0.443598	0.937561	0.262805	0.035331	2014-02
MyHash_16_v3		30000 B/s	USD	1024 MiB	8.000000	0.386206	0.095	0.828	0.038319	0.329140	0.028324	0.000000	2014-03
MyHash_16_v4		30000 B/s	USD	1024 MiB	8.000000	0.002113	0.737	0.000	0.083575	0.059226	0.000119	0.005475	2014-03

See: <https://www.random.org>, <http://www.cacert.org/>, etc

Hypothesis testing

The most common **test statistic** is the chi-squared, χ^2

$$\chi^2 = \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i} = N \sum_{i=1}^n \frac{(O_i/N - p_i)^2}{p_i} \quad (5)$$

where

- χ^2 = Pearson's cumulative test statistic, which asymptotically approaches a χ^2 .
- O_i = the number of observations of type i .
- N = total number of observations
- p_i = the fraction of type i w.r.t. the total (N)
- $E_i = Np_i$ = the expected (theoretical) count of type i
- n = the number of cells in the table.

This resembles a normalized sum of squared deviations between observed and theoretical frequencies of occurrence.

Hypothesis testing in our case

Instead of testing the **randomness** we can test the **uniformity** of our RNG using the χ^2 and a simple histogram:

$$\chi^2 = \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i} = N \sum_{i=1}^n \frac{(O_i/N - p_i)^2}{p_i} \quad (6)$$

In the following, we can think of these quantities as:

- O_i = the number of times we get a number in some range (i.e. bin i of the histogram)
- N = total number of random numbers that we analyze
- p_i = the fraction of the total range of the random numbers that each bin i represents
- $E_i = Np_i$ = the expected number of times a random number lands in each bin i
- n = the number of bins in the histogram.

Testing the uniformity of this PRNG

A histogram is a graphical representation of a discrete probability distribution. To make a simple histogram, all you need to do is:

```
# Import the numpy random number generator  
from numpy import random as rng  
  
# Import the plotting libraries  
import matplotlib.pyplot as plt  
  
# Generate 100 random nums  
# distributed between [0,1)  
# (returns a numpy array)  
data = rng.random(100)  
  
# Fill a histogram  
plt.hist(data)  
plt.show()
```

Testing the uniformity of this PRNG

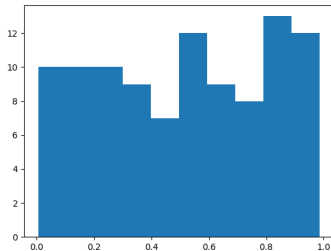
A histogram is a graphical representation of a discrete probability distribution. To make a simple histogram, all you need to do is:

```
# Import the numpy random number generator
from numpy import random as rng
```

```
# Import the plotting libraries
import matplotlib.pyplot as plt
```

```
# Generate 100 random nums
# distributed between [0,1)
# (returns a numpy array)
data = rng.random(100)
```

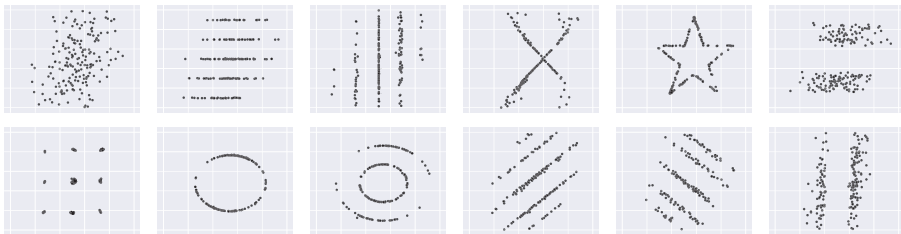
```
# Fill a histogram
plt.hist(data)
plt.show()
```



Words of caution regarding hypothesis testing and data sets

We want to compute a χ^2 to see if it's uniform (part of your homework assignment due next Thursday).

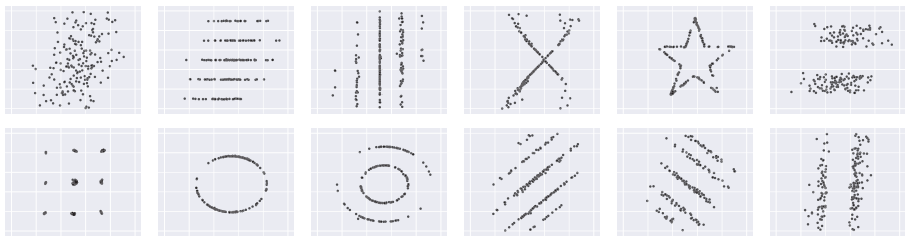
However, before going further: be **very wary of reliance on any one method** for analyzing a dataset. For example, look at these graphs:



Words of caution regarding hypothesis testing and data sets

We want to compute a χ^2 to see if it's uniform (part of your homework assignment due next Thursday).

However, before going further: be **very wary of reliance on any one method** for analyzing a dataset. For example, look at these graphs:



As discussed in [this paper](#), while different in appearance, **each has the same summary statistics to 2 decimal places:**

- means: $\bar{x} = 54.02, \bar{y} = 48.09$
- std. deviation: $\sigma_x = 14.52, \sigma_y = 24.79$
- Pearson's correlation coefficient: $r = +0.32$