Class Communication

Gregor von Laszewski

Editor

laszewski@gmail.com

https://cloudmesh-community.github.io/pub/vonlaszewski-communicate.epub

November 23, 2019 - 05:22 AM

Created by Cloudmesh & Cyberaide Bookmanager, https://github.com/cyberaide/bookmanager

Class Communication

Gregor von Laszewski

Class Communication

1 COMMUNICATION

1.0.1 General Remarks about Communication

Please do not use the e-mail of TA's and Instructors to communicate with them. They are **NOT** allowed to answer any questions send to them via e-mail. Instead use piazza and in some cases github issues.

The reason is simple. The class is managed by multiple people. If you send mail to an individual others (either instructors, TAs, or student can not see it).

Please read carefully and experiment practically how to communicate with piazza.

An assignment to post a **formal** bio is typically given to not only test your ability to use piazza, but also introduce yourself to the class.

1.0.2 Piazza

We use Piazza (https://piazza.com) because questions and answers on Piazza are community-edited and provides the opportunity not only for instructors, but also for students to contribute. Each question has a single answer edited by the students of the class and if needed an instructors' answer that is collaboratively edited by the instructors.

Due to this wiki-style question and answer, when a student has a question, one does not have to look through long e-mail threads but instead can look at the answer. For details that lead up to the answer you are highly encouraged to also look at some comments that lead up to the answer.

An advertisement video from Piazza summarizes the features:

-  Piazza Overview from Piazza

Piazza Support with a lot of information is available at:

- http://support.piazza.com

A good document about piazza is available at

- https://piazza.com/pdfs/piazza_product_introduction.pdf

1.0.2.1 Access to Piazza from Canvas

Piazza is one of the recommended IU supported technologies within CANVAS. It replaces the CANVAS discussion groups with superior technology targeted to support large student classes while also focusing on student engagement.

To access piazza you can have the following situations provided in the next four subsections. Please read **ALL** of them **CAREFULLY**, decide which applies tou you and follow the instructions. If you have improvements to this instructions, please let us know.

1.0.2.2 Piazza for 516

Due to a bug in canvas the piazza location for 516 could not be crosslisted to CANVAS. Thus we simply add the following link to it

- https://piazza.com/iu/fall2018/516

This link is used also for the undergraduates and the CS students

If you have any issues with enrolling in Piazza, please contact us in the office hours.

In Piazza we always use the `@iu.edu` account.

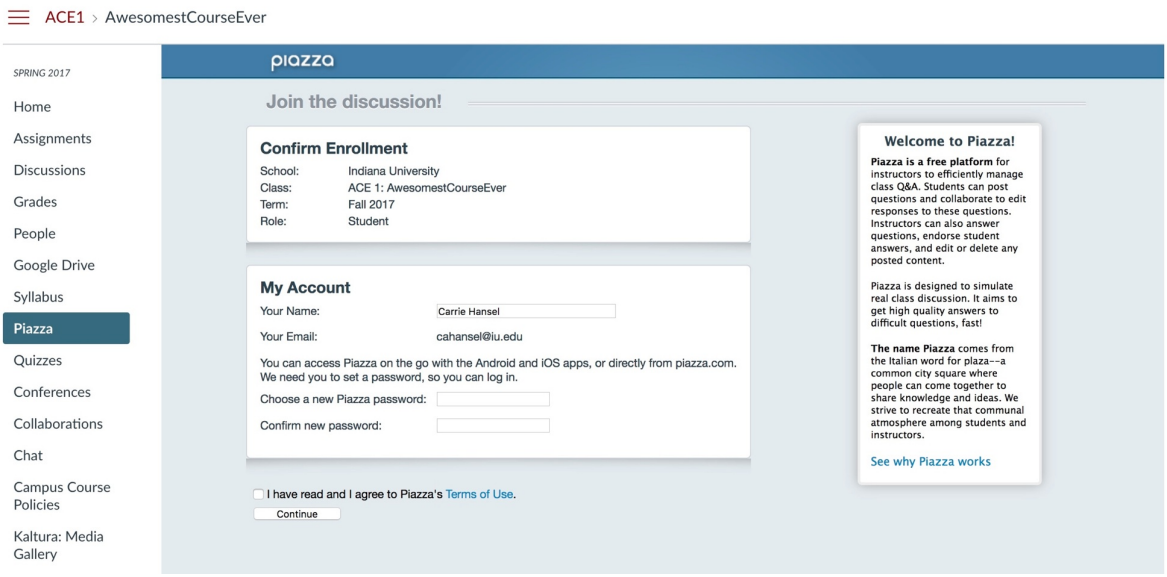
The accounts `yourid@indiana.edu` and `yourid@iu.edu`, even though they are they organizational IU accounts, are considered as two different accounts for Piazza. If you have used in piazza previously the `@indiana.edu` account, you will need to use the `@iu.edu` account instead as this is the preferred account for external services that IU offers. To do that please go to piazza and than go to the settings within piazza. Here you add your `@iu.edu` and merge it with your existing `@indiana.edu` account. This can also be done with any other account you may have used in

case it was not the `@iu.edu` account.

1.0.2.3 Piazza for other classes

1.0.2.3.1 Situation: You have never logged into piazza

First, Click the Piazza link on the left navigation of your Canvas course.

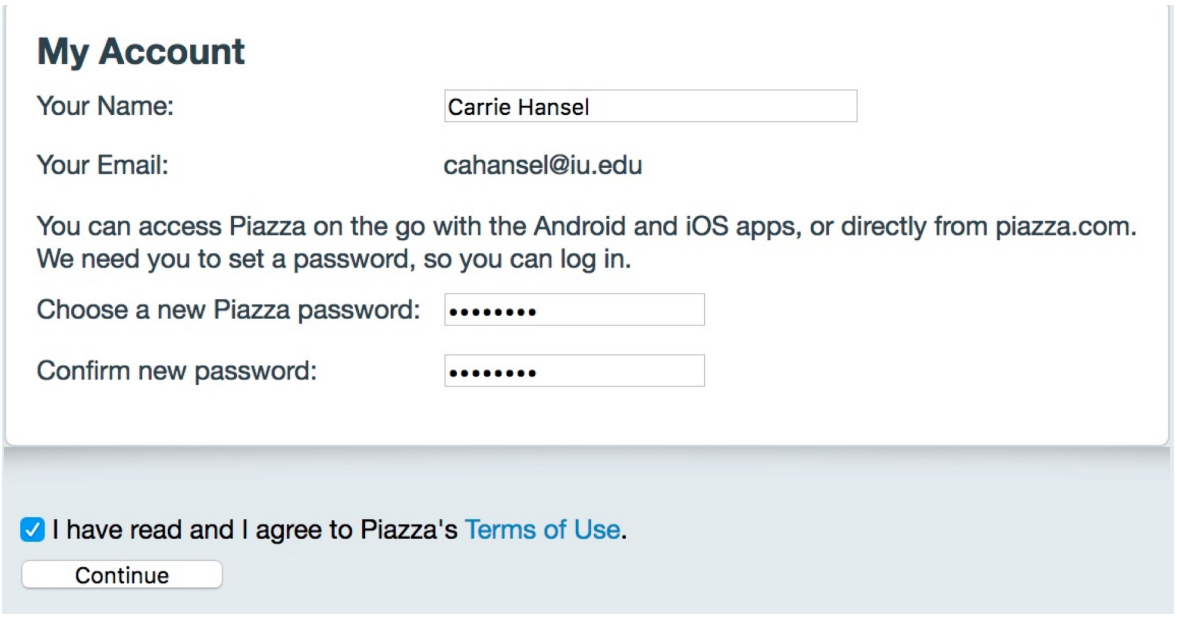


image

Second, create password and accept terms.

The email address shown on this screen is your default IU email address. It is the address Canvas sends to all integrated tools like Piazza. You can not edit it, so do not try.

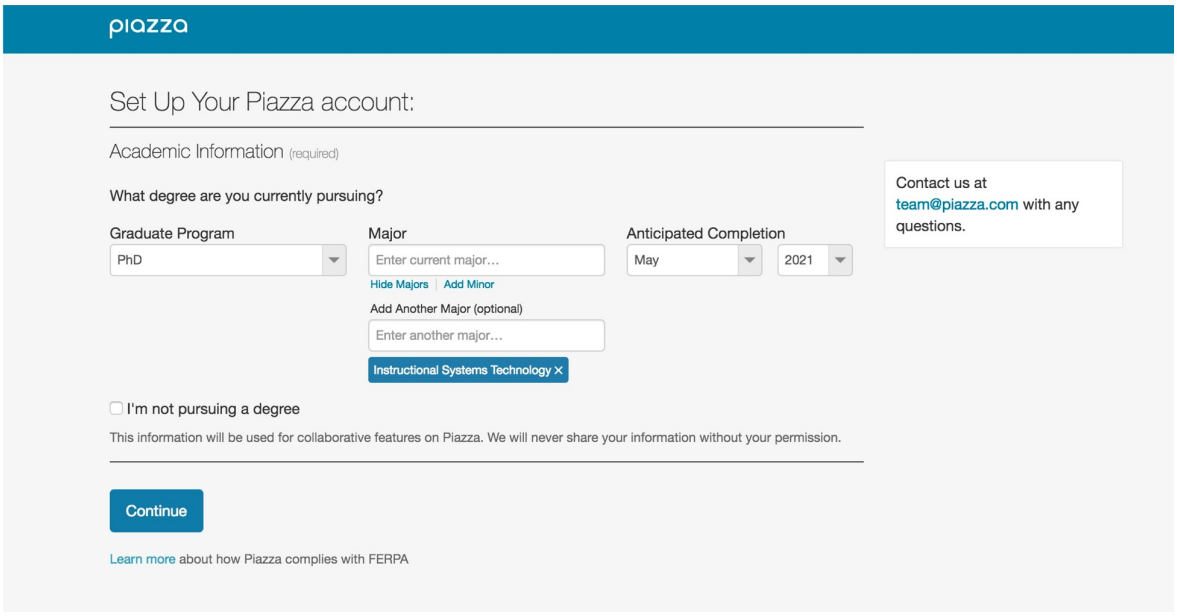
The password you create here is for accessing Piazza from a mobile device. You must use the default IU email address from this screen to access this account on another device, so make a note of it.



image

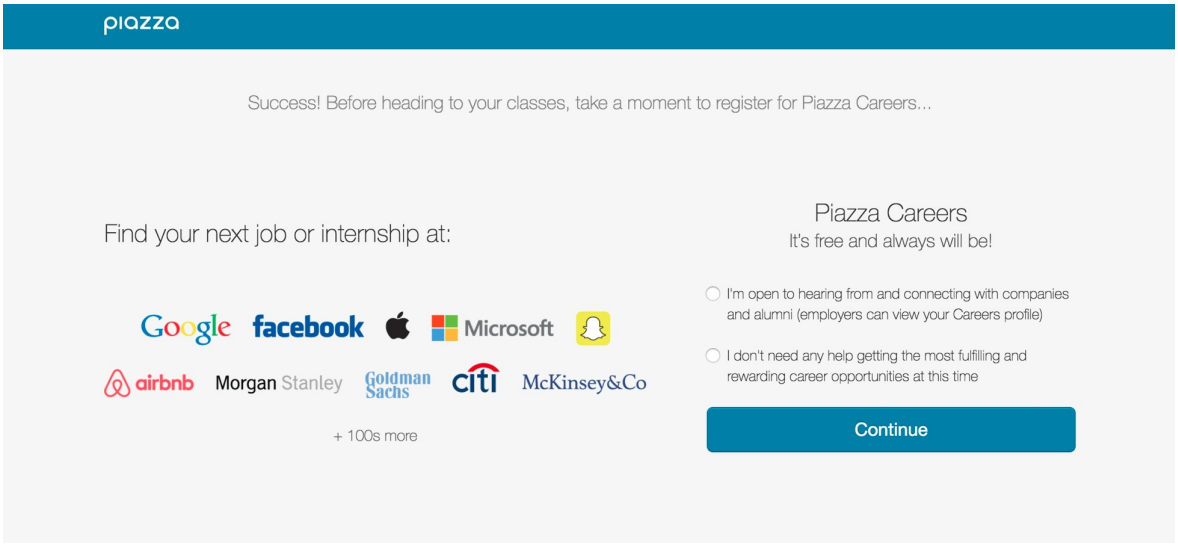
Choose current degree program (only important if you want to opt into their recruiting program on the next screen; choose whatever you want here)

Third, associate your IU account



image

Forth, if all goes well you see the Success screen



image

1.0.2.3.2 Situation: You have logged into piazza and used your default IU e-mail

1. Click the Piazza link on the left navigation of your Canvas course.
2. You will be automatically enrolled in the course Piazza site and logged in.
3. Start using Piazza.

1.0.2.3.3 Situation: You have logged into piazza and you used another non IU e-mail

1. Click the Piazza link on the left navigation of your Canvas course.
2. Proceed as in #1. This will create your new Piazza account that is linked to your courses in Canvas. This is the account you should always use in your IU courses.
3. If you wish to merge other accounts that you own, please see Add an email address or merge two accounts.

1.0.2.3.4 Situation: You have multiple accounts in piazza

1. If one of your multiple accounts corresponds with your default IU email address, you will be automatically enrolled in the course Piazza site and logged in.
2. If none of your accounts corresponds to your default IU email address, follow the instructions in #3.
3. If you wish to merge other accounts that you own, please see Add an email address or merge two accounts.

I post the official response form the CANVAS team here: “When a student clicks the Piazza link in your course navigation, they will be authenticated through to Piazza. If the student already has a Piazza account that matches their default Canvas email, they will simply be enrolled in the Piazza course. If the student does not have an account, Canvas sends the pertinent information (default email address primarily) to Piazza, Piazza creates the student’s account and enrolls the student in the Piazza course. There is nothing you need to do to.”

If you have any questions regarding accessing piazza, please send them to

“Ricci, Margaret P” <>

1.0.2.4 Verify you are on Piazza via a post

Post in the **bio** folder a short introduction about yourself. One that you could include in a paper.

An example is provided at https://laszewski.github.io/bio.html with an image at https://laszewski.github.io/_images/gregor.jpg

Use the subject line *Biography: Firstname Lastname* and post it into the bio

folder.

1.0.2.5 Making Piazza Work

In order for Piazza to work students and instructors need to participate

Students participate: Students must collaboratively work on an answer to a question. Students must not post irrelevant followups to a question. If you notice your comment was irrelevant, please delete it. Students must **search** prior to asking a new question if the question has already been asked. Duplicated questions can be merged.

Instructors guide: The instructor guides the students in order to obtain an answer to a question. In some cases the instructor may be the only one knowing the answer in which case he tries to provide it.

Not using e-mail: Instructors will and must not use e-mail to communicate with a student. All communication will be done via piazza. There, are only very view situations where e-mail is allowed, ask on piazza first if you should engage in e-mail conversations.

Not using CANVAS discussions: We will not engage in any CANVAS message exchange. Any communication is to be done on Piazza. It is in your responsibility to enroll in Piazza to make it work for you. Instructions are posted in this document. (Grade discussion will be done in CANVAS)

1.0.2.6 Towards good questions

Naturally when you ask a question you need to do it in a reasonable form and provide sufficient information so that the question can be answered. It is in the responsibility of the student to update the question to provide enough information.

Thus information may include: Firstname Lastname, HID, and URL to document in question

To give you an example of a **bad** question consider:

```
*send from Xi Lee*

Hi Professor:

I read a nice article about apples
and potato's and updated my
paper. Please give me feedback

Thank you

Kevin
```

Here the reasons why this can be improved:

1. As professors and instructors may review your document it is unnecessary to start with “Hi Professor:”, just leave it away. If you want a particular instructor use the name explicitly, such as “Gregor:”, e.g. multiple professors may be teaching your course.
2. You have not specified which article you read, you need to include the **URL** to the article so we can follow your argument.
3. You have not included the **URL** to your document so we do not know what you are talking about. Remember there are many others students in the class.
4. You are using a different name from the one that you are registered with. This can lead to confusion when we look up your name. We prefer that you use only one name that is associated with your e-mail.

The previous bad question will simply be commented on (if at all):

“Missing information” or “?” indicating that information is missing.

It is in your responsibility to figure out which information is missing. You need to modify the original post and.

1.0.2.7 Guide on how to ask good questions

This guide is adapted from

- http://www.techsupportalert.com/content/how-ask-question-when-you-want-technical-help.htm

Steps to getting your question answered on piazza

1. Before you even go to ask a question, think through what your problem is.

Write down how you are going to describe it. Think about it from the other side - what would you need to know if a student came to you and asked the question? Gather all the system information that seems to bear on the problem (see how at this link). Sometimes it even happens that by thinking through the problem, you come up with the answer yourself.

2. Verify that your question has not yet been answered with a search on the Web, Class Web page, or class piazza, this may require multiple searches.

3. In case it is a technical question, write down any error codes that appear on your screen. Do **not use screenshots** if the text is characters. This is because a reply my need to paste and copy from the original. Also screenshots are not searchable. We will not answer any questions that post screenshots if they are not necessary. It is far easier to copy and paste and use terminal type in the formatting. Also if the text is posted it is searchable. (Any unnecessary screenshot will receive a point deduction. Based on experience we have to do this as previous students in other classes ignored this policy).

4. Place your question or problem in a forum that is relevant to its subject. That may seem obvious but anyone who has experience with forums knows that a lot of questions show up in the wrong place. You will need to identify one or more a fitting piazza “folders” (folders sort the posts by topics).

5. Select a title that briefly and accurately describes your problem. A title like “Help!” or “Computer will not work” will often get ignored. Almost any problem can be titled with a few key words that will raise interest in somebody who is familiar with the subject. A corollary to this is to avoid using all caps or a lot of exclamation points. Something like “HELP!!!” turns many people off.

6. In the post, briefly describe the problem in a paragraph. Leave out unnecessary details. Save everybody time by listing any solutions that you have tried but did not work. Avoid using screenshots if they are not needed. (I mention this again).

7. In case of a technical issue describe relevant system details. For example, it is essential to designate your operating system and type of computer and

any components that might be involved in your problem. List any error code that has been displayed. Be prepared to provide more details if asked.

8. Tell what you were doing when you encountered the problem. If it is a reproducible problem, list the steps or computer operations that cause the problem.

9. If applicable, List any recent software you have installed or hardware changes you have made. If you have updated any drivers recently, also list that.

10. Formulate your questions and answers in a courteous manner. Respect the answers from others. Somebody is giving you their time and expertise for free. You may want to come back to the forum and it pays to be friendly.

11. If a suggested solution works, be sure to return to piazza and report your success. It is the least you can do to return something for the help you have been given. It will make you welcome in the forum the next time you go there for help.

1.0.2.8 Piazza class Links

> ⚠ *Using the direct links listed in the next two sections, can lead to you not getting proper access via Canvas. If you click on these links* ***before they create*** *the account via the link in your current Canvas course, you will create an account that is not matched up with Canvas. To avoid issues make sure you integrate to piazza via Canvas first.*

If you have questions about this contact Margaret Ricci (mricci@iu.edu).

1.0.2.8.1 Current Classes

> ⚠ *Before clicking on the links you need to have an account created.*

Class	Semester	Year	Link
E516	Fall	2018	piazza home link

1.0.2.8.2 Previous Classes

Class	Semester	Year	Link
E516	Spring	2018	piazza home link
E616	Spring	2018	piazza home link
I524	Spring	2018	piazza home link
I523	Fall	2017	piazza home link
I524	Spring	2017	piazza home link
I523	Fall	2016	piazza home link

1.0.2.9 Piazza Curation

We are using Piazza in a curated fashion. This means that we try to file posts into folders and delete messages that are not relevant or do not provide any additional information. It also means that if you see an error in the post that students and instructors have provided, you should improve it. This may include spelling errors.

As part of the curation we are introducing a number of folders. The folders may vary from class and we can add new folders if needed. Please let us know.

We provide next a list of example folders that we have used in previous classes and list their purpose. Please help us to add folders form your class.

Folder	Description
logistics	Any question and discussion related to the logistics of the course.
lectures	Any question and discussion related to the lectures.
project	Any question and discussion related to the project.
chapter	Any question and discussion related to the chapter assignment
section A	ny question and discussion related to the section assignment.

python	Any question and discussion related to python.
pi	Any question and discussion related to the Raspberry Pi 3. We are not using older Raspberry Pi's and therefore can not comment to them.
bio	A homework folder in which you only publish your bio. The bio needs to be published as a *note*. This assignment also serves us to see if you are in piazza. Please do this assignment ASAP. You need to post a formal bio. See the many great examples in the folder.
help	If you need help and none of the other folders fits, please use this folder. If information from here will result into new Web page content it will be added and marked into the folder *resolved*. See the *resolved* folder for more detail.
resolved	Sometimes we move some general help messages to the resolved folder in case the help message results into information that is posted on our class Web page. We than will add a link to where in the class Web page this question was answered. The TAs will aggressively try to put information into the Web page.
discussion	Any content that deserves its separate discussion and is not covered in the listed folders.

In addition to these general folders we also have some folders which **MUST NOT BE USED BY ANY STUDENT TO POST CONTENT**. These folders serve to communicate your assignments and are used internally between Gregor and the TA's. Please do not mark any questions about assignments with the `assignments` folder, but instead use `help`.

assignments:

> This folder only lists the assignments. At any time in the class you can click on the assignment folder and list the assignments given to the class. Thus there is no confusion which assignments have been given. In case students have questions about assignments they should not use the *assignments* folder, but the *help* folder. TAs are instructed to correct wrongly filed messages in folders.

In case you decide to post privately and the information is useful for others also,

the message will be published to the class.

1.0.2.10 Read the Originals, not just the e-mail

Piazza provides a convenient mechanism to update you through e-mail when an answer is changed or when someone posts.

> ⚠ *In many cases additional information is available for the post and it is **not sufficient to just read the mail**.*

The mail is just a reminder that something happened. Use the `<click here>` feature in the mail to get not only to the update, but to the actual post. Than you can get reminded about the information that is part of the post and potentially answers your question in full. It is not sufficient to participate in this class while only reading email, you should participate while visiting piazza and actively contribute to it.

1.0.2.11 Exercises

Piazza.1

> Enroll in piazza

Piazza.2

> Post a short formal bio in the bio folder and optionally include a professional portrait of yourself. Make sure you understand what a formal bio and portrait is. Research this in the internet. Look at IEEE papers for examples.

Piazza.3

> How do you find out within Piazza which assignments have been posted?

Piazza.4

Please watch the Video about Piazza

Piazza.5

Why is it important to not just read the piazza e-mail, but to visit the post by using the `<click here>` feature in the mail?

1.1 GITHUB

Learning Objectives

- Be able to use the github cloud sevices to collaborately develop contents and programs.
- Be able to use github as part of an open source project.

In some classes the material may be openly shared in code repositories. This includes class material, papers and project. Hence, we need some mechanism to share content with a large number of students.

First, we like to introduce you to git and github.com (Section 1.1). Next, we provide you with the basic commands to interact with git from the commandline (Section 1.12). Than we will introduce you how you can contribute to this set of documentations with pull requests.

1.1.1 Overview

Github is a code repository that allows the development of code and documents with many contributors in a distributed fashion. There are many good tutorials about github. Some of them can be found on the github Web page. An interactive tutorial is for example available at

- https://try.github.io/

However, although these tutorials are helpful in many cases they do not address some cases. For example, you have already a repository set up by your

organization and you do not have to completely initialize it. Thus do not just replicate the commands in the tutorial, or the once we present here before not evaluating their consequences. In general make sure you verify if the command does what you expect **before** you execute it.

A more extensive list of tutorials can be found at

- https://help.github.com/articles/what-are-other-good-resources-for-learning-git-and-github

The github foundation has a number of excellent videos about git. If you are unfamiliar with git and you like to watch videos in addition to reading the documentation we recommend these videos

- https://www.youtube.com/user/GitHubGuides/videos

Next, we introduce some important concepts used in github.

1.1.2 Upload Key

Before you can work with a repository in an easy fashion you need to upload a public key in order to access your repository. Naturally, you need to generate a key first which is explained in the section about ssh key generation (🅾 TODO: lessons-ssh-generate-key include link) before you upload one. Copy the contents of your `.ssh/id_rsa.pub` file and add them to your github keys.

More information on this topic can be found on the github Web page.

1.1.3 Fork

Forking is the first step to contributing to projects on GitHub. Forking allows you to copy a repository and work on it under your own account. Next, creating a branch, making some changes, and offering a pull request to the original repository, rounds out your contribution to the open source project.

 Git 1:41 Fork

1.1.4 Rebase

When you start editing your project, you diverge from the original version. During your developing, the original version may be updated, or other developers may have some of their branches implementing good features that you would like to include in your current work. That is when *Rebase* becomes useful. When you *Rebase* to certain points, could be a newer Master or other custom branch, consider you graft all your on-going work right to that point.

Rebase may fail, because some times it is impossible to achieve what we just described as conflicts may exist. For example, you and the to-be-rebased copy both edited some common text section. Once this happens, human intervention needs to take place to resolve the conflict.

 Git 4:20 Rebase

1.1.5 Remote

Collaborating with others involves managing the remote repositories and pushing and pulling data to and from them when you need to share work. Managing remote repositories includes knowing how to add remote repositories, remove remotes that are no longer valid, manage various remote branches and define them as being tracked or not, and more.

Though out this semester, you will typically work on two *remote* repos. One is the office class repo, and another is the repo you forked from the class repo. The class repo is used as the centralized, authority and final version of all student submissions. The repo under your own Github account is for your personal storage. To show progress on a weekly basis you need to commit your changes on a weekly basis. However make sure that things in the master branch are working. If not, just use another branch to conduct your changes and merge at a later time. We like you to call your development branch dev.

- https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes

1.1.6 Pull Request

Pull requests are a means of starting a conversation about a proposed change back into a project. We will be taking a look at the strength of conversation,

integration options for fuller information about a change, and cleanup strategy for when a pull request is finished.

 Git 4:26 Pull Request

1.1.7 Branch

Branches are an excellent way to not only work safely on features or experiments, but they are also the key element in creating Pull Requests on GitHub. Lets take a look at why we want branches, how to create and delete branches, and how to switch branches in this episode.

 Git 2:25 Branch

1.1.8 Checkout

Change where and what you are working on with the checkout command. Whether we are switching branches, wanting to look at the working tree at a specific commit in history, or discarding edits we want to throw away, all of these can be done with the checkout command.

 Git 3:11 Checkout

1.1.9 Merge

Once you know branches, merging that work into master is the natural next step. Find out how to merge branches, identify and clean up merge conflicts or avoid conflicts until a later date. Lastly, we will look at combining the merged feature branch into a single commit and cleaning up your feature branch after merges.

 Git 3:11 Merge

1.1.10 GUI

Using Graphical User Interfaces can supplement your use of the command line to get the best of both worlds. GitHub for Windows and GitHub for Mac allow

for switching to command line, ease of grabbing repositories from GitHub, and participating in a particular pull request. We will also see the auto-updating functionality helps us stay up to date with stable versions of Git on the command line.

 Git 3:47 GUI

There are many other git GUI tools available that directly integrate into your operating system finders, windows, …, or PyCharm. It is up to you to identify such tools and see if they are useful for you. Most of the people we work with us git from the command line, even if they use PyCharm, eclipse, or other tools that have build in git support. You can identify a tool that works best for you.

1.1.11 Windows

This is a quick tour of GitHub for Windows. It offers GitHub newcomers a brief overview of what this feature-loaded version control tool and an equally powerful web application can do for developers, designers, and managers using Windows in both the open source and commercial software worlds. More: http://windows.github.com

 Git 1:25 Windows

1.1.12 Git from the Commandline

Although github.com provides a powerful GUI and other GUI tools are available to interface with github.com, the use of git from the commandline can often be faster and in many cases may be simpler.

Git commandline tools can be easily installed on a variety of operating systems including Linux, macOS, and Windows. Many great tutorials exist that will allow you to complete this task easily. We found the following two tutorials sufficient to get the task accomplished:

- https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
- https://www.atlassian.com/git/tutorials/install-git

Although the later is provided by an alternate repository to github. The installation instructions are very nice and are not impacted by it. Once you have installed git you need to configure it.

1.1.13 Configuration

Once you installed Git, you can need to configure it properly. This includes setting up your username, email address, line endings, and color, along with the settings' associated configuration scopes.

 Git 2:47 Configuration

It is important that make sure that use the `git config` command to initialize git for the first time on each new computer system or virtual machine you use. This will ensure that you use on all resources the same name and e-mail so that git history and log will show consistently your checkins across all devices and computers you use. If you do not do this, your checkins in git do not show up in a consistent fashion as a single user. Thus on each computer execute the following commands:

```
$ git config --global user.name "Albert Zweistein"
$ git config --global user.email albert.zweistein@gmail.com
```

where you replace the information with the information related to you. You can set the editor to emacs with:

```
$ git config --global core.editor emacs
```

Naturally if you happen to want to use other editors you can configure them by specifying the command that starts them up. You will also need to decide if you want to push branches individually or all branches at the same time. It will be up to you to make what will work for you best. We found that the following seems to work best:

```
git config --global push.default matching
```

More information about a first time setup is documented at:

```
* http://git-scm.com/book/en/Getting-Started-First-Time-Git-Setup
```

To check your setup you can say:

```
$ git config --list
```

One problem we observed is that students often simply copy and paste instructions, but do not read carefully the error that is reported back and do not fix it. Overlooking the proper set of the push.default is often overlooked. Thus we remind you: **Please read the information on the screen when you set up**.

1.1.14 Upload your public key

Please upload your public key to the repository as documented in github, while going to your account and find it in settings. There you will find a panel SSH key that you can click on which brings you to the window allowing you to add a new key. If you have difficulties with this find a video from the github foundation that explains this.

1.1.15 Working with a directory that will be provided for you

In case your course provided you with a github directory, starting and working in it is going to be real simple. Please wait till an announcement to the class is send before you ask us questions about it.

If you are the only student working on this you still need to make sure that papers or programs you manage in the repository work and do not interfere with scripts that instructors may use to check your assignments. Thus it is god to still create a branch, work in the branch and than merge the branch into the master once you verified things work. After you merged you can push the content to the github repository.

Tip: Please use only **lowercase** characters in the directory names and no special characters such as `@ ; / _` and spaces. In general we recommend that you avoid using directory names with capital letters spaces and _ in them. This will simplify your documentation efforts and make the URLs from git more readable. Also while on some OS's the directories *MyDirectory* is different from *mydirectory* on macOS it is considered the same and thus renaming from capital to lower case can not be done without first renaming it to another directory.

Your homework for submission should be organized according to folders in your clone repository. To submit a particular assignment, you must first add it using:

```
git add <name of the file you are adding>
```

Afterwards, commit it using:

```
git commit -m "message describing your submission"
```

Then push it to your remote repository using:

```
git push
```

If you want to modify your submission, you only need to:

```
git commit -m "message relating to updated file"
```

afterwards:

```
git push
```

If you lose any documents locally, you can retrieve them from your remote repository using:

```
git pull
```

1.1.16 README.yml and notebook.md

In case you take classes e516 and e616 with us you will have to create a README.yaml and notebook.md file in the top most directory of your repository. It serves the purpose of identifying your submission for homework and information about yourself.

It is important to follow the format precisely. As it is yaml it is an easy homework to write a 4 line python script that validates if the README.yaml file is valid. In addition you can use programs such as `yamllint` which is documented at

- https://yamllint.readthedocs.io/en/latest/

This file is used to integrate your assignments into a proceedings. An example is provided at

- https://github.com/cloudmesh-community/hid-sample/blob/master/README.yml

Any derivation from this format will not allow us to see your homework as our

automated scripts will use the README.yml to detect them. Make sure the file does not contain ay TABs. Please also mind that all filenames of all homework and the main directory must be **lowercase** and do not include spaces. This will simplify your task of managing the files across different operating systems.

In case you work in a team, on a submission, the document will only be submitted in the author and hid that is listed first. All other readme files, will have for that particular artifact a `duplicate: yes` entry to indicate that this submission is managed elsewhere. The team will be responsible to manage their own pull requests, but if the team desires we can grant access for all members to a repository by a user. Please be aware that you must make sure you coordinate with your team.

We will not accept submission of homework as pdf documents or tar files. All assignments must be submitted as code and the reports in native latex and in github. We have a script that will automatically create the PDF and include it in a proceedings. There is no exception from this rule and all reports not compilable will be returned without review and if not submitted within the deadline receive a penalty.

Please check with your instructor on the format of the README.yaml file as it could be different for your class.

To see an example for the notebook.md file, you can visit our sample hid, and browse to the notebook.md file. Alternatively you can visit the following link

- https://github.com/cloudmesh-community/hid-sample/blob/master/notebook.md

The purpose of the notebook md file is to record what you did in the class to us. We will use this file at the end of the class to make sure you have recorded on a weekly basis what you did for the class. Inactivity is a valid response. Not updating the notebook, is not.

The sample directory contains other useful directories and samples, that you may want to investigate in more detail. One of the most important samples is the github issues (see Section 1.19). There is even a video in that section about this and showcases you how to organize your tasks within this class, while copying

the assignments from piazza into one or more github issues. As we are about cloud computing, using the services offered by a prominent cloud computing service such as github is part of the learning experience of this course.

1.1.17 Contributing to the Document

It is relatively easy to contribute to the document if you understand how to use github. The first thing you will need to do is to create a fork of the repository. The easiest way to do this is to visit the URL

- https://github.com/cloudmesh-community/book

Towards the upper right corner you will find a link called **Fork**. Click on it and chose into which account you like to fork the original repository. Next you will create a colne from your corked directory. You will see in your fork a green clone button. You will see a URL that you can copy into your terminal. If the links does not include your username, it is the wrong link.

In your terminal you now say

```
git colne  https://github.com/<yourusername>/book
```

Now cd into this directory and make your changes.

```
$ cd book
```

Use the usual git commands such as `git add`, `git commit`, `git push`

Note you will push into your local directory.

1.1.17.1 Stay up to date with the original repo

Form time to time you will see that others are contributing to the original repo. To stay up to date you want to not only sync from your local copy, but also from the original repo. To link your repo with what is called the upstream you need to do the following once, so you can issue `git pull` tha also pulls from the upstream

Make sure you have upstream repo defined:

```
$ git remote add upstream \
     https://github.com/cloudmesh-community/book
```

Now Get latest from upstream:

```
$ git rebase upstream/master
```

In this step, the conflicting file shows up (in my case it was refs.bib):

```
$ git status
```

should show the name of the conflicting file:

```
$ git diff <file name>
```

should show the actual differences. May be in some cases, It is easy to simply take latest version from upstream and reapply your changes.

So you can decide to checkout one version earlier of the specific file. At this stage, the re-base should be complete. So, you need to commit and push the changes to your fork:

```
$ git commit
$ git rebase origin/master
$ git push
```

Then reapply your changes to refs.bib - simply use the backed up version and use the editor to redo the changes.

At this stage, only refs.bib is changed:

```
$ git status
```

should show the changes only in refs.bib. Commit this change using:

```
$ git commit -a -m "new:usr: <message>"
```

And finally push the last committed change:

```
$ git push
```

The changes in the file to resolve merge conflict automatically goes to the original pull request and the pull request can be merged automatically.

You still have to issue the pull request from the Github Web page so it is registered with the upstream repository.

1.1.17.2 Resources

- Pro Git book
- Official tutorial
- Official documentation
- TutorialsPoint on git
- Try git online
- GitHub resources for learning git Note: this is for github and not for gitlab. However as it is for gt the only thing you have to do is replace github, for gitlab.
- Atlassian tutorials for git

In addition the tutorials from atlassian are a good source. However remember that you may not use bitbucket as the repository, so ignore those tutorials. We found the following useful

- What is git: https://www.atlassian.com/git/tutorials/what-is-git
- Installing git: https://www.atlassian.com/git/tutorials/install-git
- git config: https://www.atlassian.com/git/tutorials/setting-up-a-repository#git-config
- git clone: https://www.atlassian.com/git/tutorials/setting-up-a-repository#git-clone
- saving changes: https://www.atlassian.com/git/tutorials/saving-changes
- collaborating with git: https://www.atlassian.com/git/tutorials/syncing

1.1.18 Exercises

E.Github.1:

> *How do you set your favorite editor as a default with github config*

E.Github.2:

> *What is the differencebetween merge and rebase?*

E.Github.3:

> *Assume you have made a change in your local fork, however other users have since committed to the master branch, how can you make sure your commit works off from the latest information in the master*

> *branch?*

E.Github.4:

> *Find a spelling error in the Web page or a contribution and create a pull request for it.*

E.Gitlab.5:

> *Create a README.yml in your github account directory provided for you for class.*

1.1.19 Github Issues

 Github 8:29 Issues

When we work in teams or even if we work by ourselves, it is prudent to identify a system to coordinate your work. While conduction projects that use a variety of cloud services, it is important to have a system that enables us to have a cloud service that enables us to facilitate this coordination. Github provides such a feature through its *issue* service that is embedded in each repository.

Issues allow for the coordination of tasks, enhancements, bugs, as well as self defined labeled activities. Issues are shared within your team that has access to your repository. Furthermore, in an open source project the issues are visible to the community, allowing to easily communicate the status, as well as a roadmap to new features.

This enables the community to participate also in reporting of bugs. Using such a system transforms the development of software from the traditional closed shop development toa truly open source development encouraging contributions from others. Furthermore it is also used as bug tracker in which not only you, but the community can communicate bugs to the project.

A good resource for learning more about issues is provided at

- https://guides.github.com/features/issues/

1.1.19.1 Git Issue Features

A git issue has the following features:

title

– a short description of what the issue is about

description

a more detailed description. Descriptions allow also to conveniently add check-boxed todo’s.

label

a color enhanced label that can be used to easily categorize the issue. You can define your own labels.

milestone

a milestone so you can identify categorical groups issues as well as their due date. You can for example group all tasks for a week in a milestone, or you could for example put all tasks for a topic such as developing a paper in a milestone and provide a deadline for it.

assignee

an assignee is the person that is responsible for making sure the task is executed or on track if a team works on it. Often projects allow only one assignee, but in certain cases it is useful to assign a group, and the group identifies if the task can be split up and assigns them through check-boxed todo’s.

comments

allow anyone with access to provide feedback via comments.

1.1.19.2 Github Markdown

Github uses markdown which we introduce you in Section [S:markdown].

As github has its own flavor of markdown we however also point you to

as a reference. We like to mention the special enhancements fo github's markdown that integrate well to support project management.

1.1.19.2.1 Task lists

Taks lists can be added to any description or comment in github issues To create a task list you can add to any item `[ ]`. This includes a task to be done. To make it as complete simple change it to `[x]`. Whoever the great feature of tasks is that you do not even have to open the editor but you can simply check the task on and of via a mouse click. An example of a task list could be

```
Post Bios

* [x] Post bio on piazza
* [ ] Post bio on google docs
* [ ] Post bio on github
* [ ] \(optional) integrate image in google docs bio
```

In case you need to use a `(`have at the beginning ot the task text, you need to escape it with a `\`

1.1.19.2.2 Team integration

A person or team on GitHub can be mentioned by typing the username proceeded by the `@` sign. When posting the text in the issue, it will trigger a notification to them and allow them to react to it. It is even possible to notify entire teams, which are described in more detail at

- https://help.github.com/articles/about-teams/

1.1.19.2.3 Referencing Issues and Pull requests

Each issue has a number. If you use the `#` followed by the issue number you can refer to it in the text which will also automatically include a hyperlink to the task. The same is valid for pull requests.

1.1.19.2.4 Emojis

Although github supports emojis such as `:+1:` we do not use them typically in our class.

1.1.19.3 Notifications

Github allows you to set preferences on how you lik to receive notifications. You can receive them either via e-mail or the Web. This is controlled by configuring it in *your settings*, where you can set the preferences for participating projects as well as projects you decide to watch. To access the notifications you can simply look at them in the *notification* screen. In this screen when you press the `?` you will see a number of commands that allow you to control the notification when pressing on one of them.

1.1.19.4 cc

To carbon copy users in your issue text, simply use `/cc` followed by the `@` sign and their github user name.

1.1.19.5 Interacting with issues

Github has the ability to search issues with a search query and a search language that you can find out more about it at

https://guides.github.com/features/issues/#search

A dashboard gives convenient overviews of the issues including a *pulse* that lists todo's status if you use them in the issue description.

1.1.20 Glossary

The Glossary is copied from

- https://cdcvs.fnal.gov/redmine/projects/cet-is-public/wiki/GitTipsAndTricks#A-suggested-work-flow-for-distributed-projects-NoSY

Add

put a file (or particular changes thereto) into the index ready for a commit operation. Optional for modifications to tracked files; mandatory for hitherto un-tracked files.

Branch
: a divergent change tree (eg a patch branch) which can me merged either wholesale or piecemeal with the master tree.

Commit
: save the current state of the index and/or other specified files to the local repository.

Commit object
: an object which contains the information about a particular revision, such as parents, committer, author, date and the tree object which corresponds to the top directory of the stored revision.

Fast-forward
: an update operation consisting only of the application of a linear part of the change tree in sequence.

Fetch
: update your local repository database (not your working area) with the latest changes from a remote.

HEAD
: the latest state of the current branch.

Index
: a collection of files with stat information, whose contents are stored as objects. The index is a stored version of your working tree. Files may be staged to an index prior to committing.

Master
: the main branch: known as the trunk in other SCM systems.

Merge
: join two trees. A commit is made if this is not a fast-forward operations (or one is requested explicitly.

Object
: the unit of storage in git. It is uniquely identified by the SHA1 hash of its contents. Consequently, an object can not be changed.

Origin
: the default remote, usually the source for the clone operation that created the local repository.

Pull

shorthand for a fetch followed by a merge (or rebase if –rebase option is used).

Push
: transfer the state of the current branch to a remote tracking branch. This must be a fast-forward operation (see merge).

Rebase
: a merge-like operation in which the change tree is rewritten (see Rebasing below). Used to turn non-trivial merges into fast-forward operations.

Remote
: another repository known to this one. If the local repository was created with “clone” then there is at least one remote, usually called, “origin.”

Stage
: to add a file or selected changes therefrom to the index in preparation for a commit.

Stash
: a stack onto which the current set of uncommitted changes can be put (eg in order to switch to or synchronize with another branch) as a patch for retrieval later. Also the act of putting changes onto this stack.

Tag
: human-readable label for a particular state of the tree. Tags may be simple (in which case they are actually branches) or annotated (analogous to a CVS tag), with an associated SHA1 hash and message. Annotated tags are preferable in general.

Tracking branch
: a branch on a remote which is the default source / sink for pull / push operations respectively for the current branch. For instance, origin/master is the tracking branch for the local master in a local repository.

Un-tracked
: not known currently to git.

1.1.21 Example commands

To work in your local directory you can use the following commands. Please note that these commands do not upload your work to github, but only introduce version control within your local files.

The command list is copied from

- https://cdcvs.fnal.gov/redmine/projects/cet-is-public/wiki/GitTipsAndTricks#A-suggested-work-flow-for-distributed-projects-NoSY

1.1.21.1 Local commands to version contril your files

Obtain differences with

```
$ git status
```

Move files from one part of your directory tree to another:

```
$ git mv <old-path> <new-path>
```

Delete unwanted tracked files:

```
$ git rm <path>
```

Add un-tracked files:

```
$ git add <un-tracked-file>
```

Stage a modified file for commit:

```
$ git add <file>
```

Commit currently-staged files:

```
$ git commit -m <log-message>
```

Commit only specific files (regardless of what is staged):

```
$ git commit -m <log-message>
```

Commit all modified files:

```
$ git commit -a -m <log-message>
```

Un-stage a previously staged (but not yet committed) file:

```
$ git reset HEAD <file>
```

Get differences with respect to the committed (or staged) version of a file:

```
$ git diff <file>
```

Get differences between local file and committed version:

```
$ git diff --cached <file>
```

Create (but do not switch to) a new local branch based on the current branch:

```
$ git branch <new-branch>
```

Change to an existing local branch:

```
$ git checkout <branch>
```

Merge another branch into the current one:

```
$ git merge <branch>
```

1.1.21.2 Interacting with the remote

Get the current list of remotes (including URIs) with

```
$ git remote -v
```

Get the current list of defined branches with

```
$ git branch -a
```

Change to (creating if necessary) a local branch tracking an existing remote branch of the same name:

```
$ git checkout <branch>
```

Update your local repository ref database without altering the current working area:

```
$ git fetch <remote>
```

Update your current local branch with respect to your repository’s current idea of a remote branch’s status:

```
$ git merge <branch>
```

Pull remote ref information from all remotes and merge local branches with their remote tracking branches (if applicable):

```
$ git pull
```

Examine changes to the current local branch with respect to its tracking branch:

```
$ git cherry -v
```

Push changes to the remote tracking branch:

```
$ git push
```

Push all changes to all tracking branches:

```
$ git push --all
```

1.1.22 Class Git

This class will use git to manage all assignment submissions. We use the publicly available github.com. The class git is hosted at

- https://github.com/cloudmesh-community

It is in the responsibility of the student to create a github account and make sure that you will be added to the class github within one week of joining the class. Make sure to fill out the survey to communicate the github.com username.

Previous github locations include:

- https://github.com/cloudmesh-community
- https://gitlab.com/cloudmesh_fall2016
- https://github.com/bigdata-i523

Previous book githubs include

- https://github.com/cloudmesh/book

2 PRESENTATIONS

2.1 RECORDING AUDIO WITH AUTOPLAY

In some classes you may be asked to prepare a presentation that can be played at any time with recorder audio. Powerpoint provides such a mechanism, while allowing to combine the audio for each page to a consecutive recording.

To help you achieve this, we have provided the following simple demonstration.

Powerpoint with Autoplay and Sound (1:42)

3 REFERENCES