



Open Surce Project Communication



Introduction to GitHub

Gregor von Laszewski, laszewski@gmail.com

04 June, 2022

Contents

1	COMMUNICATION	3
1.1	General Remarks about Communication 	3
1.2	Organizing Collaboration in Open Source Teams 	4
1.2.1	Acknowlegments	4
1.3	Organizing Collaborative Research Teams	4
1.3.1	Communication	4
1.3.1.1	Video Conferencing	4
1.3.1.1.1	Google Meet	5
1.3.1.1.2	Zoom	5
1.3.1.2	Real-time and Offline Text Messaging	6
1.3.1.2.1	Slack	6
1.3.2	Creating Text	7
1.3.2.1	Dictation	7
1.3.2.2	Grammar Checkers	8
1.3.2.2.1	Grammarly	8
1.3.2.2.2	Word	8
1.3.2.3	Editors	9
1.3.3	Documentation with Markdown	10
1.3.4	ToDo Lists	10
1.3.5	Git and GitHub	10
1.3.5.1	Git from the command line	11
1.3.6	Git form IDEs	11
1.3.7	GitHub from a GUI	11
1.3.7.1	GitHub Commands	11
1.3.8	Appendix	12
1.4	Github 	12
1.4.1	Overview	13
1.4.2	Upload Key	14
1.4.3	Fork	14
1.4.4	Rebase	14
1.4.5	Remote	14
1.4.6	Pull Request	15
1.4.7	Branch	15
1.4.8	Checkout	15
1.4.9	Merge	16
1.4.10	GUI	16

1.4.11	Windows	16
1.4.12	Git from the Commandline	16
1.4.13	Configuration	17
1.4.14	Upload your public key	18
1.4.15	Working with a directory that will be provided for you	18
1.4.16	README.yml and notebook.md	19
1.4.17	Contributing to the Document	20
1.4.17.1	Stay up to date with the original repo	21
1.4.17.2	Resources	22
1.4.18	Exercises	23
1.4.19	Github Issues	23
1.4.19.1	Git Issue Features	24
1.4.19.2	Github Markdown	24
1.4.19.2.1	Task lists	24
1.4.19.2.2	Team integration	25
1.4.19.2.3	Referencing Issues and Pull requests	25
1.4.19.2.4	Emojis	25
1.4.19.3	Notifications	25
1.4.19.4	cc	25
1.4.19.5	Interacting with issues	26
1.4.20	Glossary	26
1.4.21	Example commands	27
1.4.21.1	Local commands to version contril your files	27
1.4.21.2	Interacting with the remote	29
1.5	Class Git 	30
2	PRESENTATIONS	30
2.1	Recording Audio with Autoplay 	30
3	REFERENCES	30

1 COMMUNICATION

1.1 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 sleck 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 slack.

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

1.2 Organizing Collaboration in Open Source Teams

Editor: Gregor von Laszewski

1.2.1 Acknowlegments

The following persons have contributed to this document Fidel Leal, Erin Seliger, Cooper Young, Agness Lungua, and Jacques Fleischer.

For more information, please contact: laszewski@gmail.com

See Also: Article on medium.com

1.3 Organizing Collaborative Research Teams

To organize a research team, it is of utmost importance to establish simple collaboration pathways. This includes ways to conduct video conferencing, text chat code sharing, editing, and task management.

In the following sections, we will list some useful tools that can be used by the research team while keeping the learning curve to a minimum. In general, it is good to ask the participants if they already use particular tools in a category and if all in the team use them to adopt them. However, this may limit the general availability in case the team grows into the open-source community. Hence, it is important to consider licensing issues and if possible adopt free tools for the research team.

1.3.1 Communication

We start by identifying tools for voice and text communications.

1.3.1.1 Video Conferencing Videoconferencing has undoubtedly become a major component of research teams. It allows face meetings without the need for traveling. Thus you can spend the time saved on travel. Also, it allows researchers to continue if unexpected events take place that does not allow in-person meetings such as the recent COVID epidemic.

Sometimes videoconferencing will produce acoustic feedback: a very loud loop of sound within a call. Members of a call can prevent this phenomenon by wearing headphones; if not possible, keep speaker volume down. Acoustic feedback also occurs when more than one device, in close proximity to each other, is on the same call. Avoid this feedback by wearing headphones or earbuds.

There are many conferencing tools available that can be used. You may even use multiple dependent on the particular meeting or preferences by the subgroup. To keep things simple it is however recommended to just use one tool.

1.3.1.1.1 Google Meet Google Meet is an online service that facilitates meetings as video and audio conference calls. It has evolved from Google Hangouts.

What are good features?

Some good features of Google Meet include the ability to use closed captions allowing the integration of participants having trouble hearing the speaker. It is compatible across devices and typically its sound quality is very good.

What are not so good features?

1. Google Meet does not provide an easy way to have others take control of a remote desktop. However, it is possible to use Google Remote Desktop for it.
 - <https://remotedesktop.google.com/?pli=1>
2. Google Meet cannot share each other's desktops at the same time. This feature was available in Hangouts but is no longer available as far as we can tell.
3. Google Meet restricts call times to one hour. After one hour has passed, a new call must be started.

Why you may consider choosing Google meet and not Zoom?

Google offers many services that are useful for collaboration. This includes Google Drive, Docs, Slides, Gmail, Calendar, and Groups. As they can be accessed through a single account, it is obvious that Google Meet provides a valuable set of services to any research team.

1.3.1.1.2 Zoom Zoom is a cloud-based communications platform that provides one-on-one's, group meetings, and webinars.

What are good features?

Some of Zoom's features include live chat, screen sharing, a whiteboard, and virtual reactions for meeting participants. Additionally, it can record meetings to the cloud or personal devices, create

breakout rooms, and allows participants to seamlessly move between them. A very important feature is that the meeting owner can remotely control another participant's screen. Zoom allows users to join a session through an established meeting URL. Participants do not need to be signed in or even have a Zoom account. Additionally, people joining from places with limited Internet access can call into the meeting's audio channel using dedicated telephone numbers.

What are not so good features?

Free account holders can host unlimited one-on-one meetings (meeting durations up to 24 hrs). In contrast to Google Meet (which has a one hour meeting limit), there is a duration limit of 40 mins for meetings with three or more participants. We observed that video quality can be unstable, and the overall platform performance can quickly deteriorate over limited bandwidth connections. In such cases, we recommend switching off the camera from the participant that has issues. Furthermore, if you do a lot of calculations at the same time on your machine, it may affect the quality of the call. This applies to older machines and should allow you to give a beautiful argument to get a new computer. In some cases, you may have a second computer and can use one for sharing your session, while the other one is used for sound, or you use a cell phone for the latter.

How can someone take control of a remote desktop?

To take control of a remote desktop, the host must activate screen sharing. Once the screen sharing is activated, we need to click the `View Options` drop-down menu (usually at the top of the screen) and click on `Request remote control`. The remote user will then get a prompt to approve the remote control request.

Institutional accounts may have the remote control functionality disabled by their account administrator. For further details, refer to the Zoom support pages.

1.3.1.2 Real-time and Offline Text Messaging In many research projects, participants may be in different timezones or have schedules that do not provide overlapping times for video-conferencing. For this reason, it is important to support a chat-like feature that allows the researchers to catch up with activities that took place they were not available for. Tools such as e-mail have filled this demand for quite some time. Recently additional tools such as Slack have appeared that enhance the e-mail activity while also allowing real-time text messaging.

1.3.1.2.1 Slack Slack is a communication software that is used for groups to send and receive text messages. Additional 3rd party services can be added to Slack that can even send messages to it automatically.

What are good features?

Slack offers a GUI that focuses on a real-time message stream. It can be used on computers, phones, and tablets. It is easy to send photos, which may be useful in case the device you need to discuss is not on the Internet but you need to share the content for example of its screen.

What are not so good features?

Slack is stream-based and does not provide a good mechanism for organizing messages once they have been sent. The thread feature is far inferior to that of even a simple e-mail client. If one is involved in many Slack workspaces, it becomes difficult to manage them. Most importantly, Slack comes with only a limited number of free messages in its free version. This means you will have to pay once you exceed the limit. Thus, even the integration of useful services such as GitHub notifications is not recommended as you will exceed the limits too quickly. A posting policy needs to be established. Those that are not using Slack frequently may be out of touch quickly. Although there is an unread threads feature, it may be filled with messages if you do not use Slack daily just to keep up.

To support the separation of topics, it is advisable to create several channels such as “general” or a channel for a particular topic. However, it is also important to limit the number of channels so it does not become too confusing. Keep in mind that Slack users are not automatically added to new channels.

How does one add all Slack workspace users to a new channel?

By default, users must manually browse the list of channels and join them by clicking `Add channels` on the left side of the GUI. However, any user can add everyone to the channel by doing the following:

1. Type `/who` in `#general` and then press `Enter` to send the message; you will see a message only you can see: a list of all the members in the channel.
2. Copy this message from Slackbot. Ensure you are copying the names of the users which all begin with `@`.
3. Paste the output in the new channels and press `Enter`. This will not actually mention everybody— it will only prompt a dialog box confirming to invite everyone. Click `Invite Them`.

1.3.2 Creating Text

As part of your project, you will need to develop documents such as manuals or reports; hence, you must have the means to easily add text to your project. This includes the creation of text via editors but also through dictation in case this is useful and works for you.

1.3.2.1 Dictation Sometimes it is convenient to directly dictate the text for a manual or tutorial into an editor. On macOS and Windows, you will find useful tools for this. A voice-to-text recorder may also

help you in case you have a recoded video of yourself to generate transcripts. A disadvantage for a lot of non-native English speakers is that the accuracy may be limited and that not using them leads to unacceptable results. Some of them can be trained.

Hence before you integrate it into your toolset, we recommend trying it out. Different participants may be more successful than others. However, errors will happen and you will have to clean up the document after dictation.

Examples include:

- Apple Dictation (free for Apple devices): You can directly dictate into various applications that help you improve your text such as Grammarly and MS Word if you have installed them.
- Windows 10 Voice Recognition (free for Windows users). You can directly record into MS Word so you get a free grammar checker. **Note: We received reports from some of our participants that they could not get this to work. More investigation is needed.**
- Google Docs offers build-in voice typing. Its recognition quality is very good as well as fast. It does have difficulties recognizing some names and acronyms. **Note: that we were unable to get it to work in Chromium on Ubuntu 21.04. We have not tried other versions.**

1.3.2.2 Grammar Checkers When developing content for tutorials and documentation, it is important to check their correctness with a grammar checker. We have made the best experience with Grammarly followed by MS Word. The best way to use them is to copy and paste small sections into them from your document and then check them. After you are satisfied, copy the contents back to the original one while overwriting the old text.

1.3.2.2.1 Grammarly What are good features?

Grammarly works well, is available for free, and the free version is good enough for most.

What are not so good features?

As with any grammar checker, not everything is corrected properly. In some text, it shows false errors, but it's still very good.

1.3.2.2.2 Word What are good features?

The word grammar checker is built into Word and has a high accuracy.

What are not so good features?

In practice, we observed that Grammarly performs better. Copying text back and forth can introduce errors when it comes to using quotes and other symbols. Thus you must check all symbols after you

copy them into a markdown document. In case you have the choice we currently recommend you use Grammarly.

1.3.2.3 Editors You will likely need multiple editors as part of your research activities. This is motivated by the fact that we do lots of development on your local machine, but also do remote development via terminal access to a remote computer that does not have a GUI. In case you only want to learn one editor to do all of this, just use emacs. We have listed below some editors and you may want to choose

- `emacs` vs `vi/vim` for terminal editing
- `pycharm` vs MS code for fancyful python code development

There is a third option that we will use and that is Jupyter which allows us to interactively develop Python code. Jupyter is important as it is used by many data scientists due to its ad hoc interactive mode while programming. However, PyCharm and VS Code also provide options to view and run Jupyter notebooks. However, not everything that works in Jupyter is working on these platforms.

Here is a list of popular editors for python:

- `emacs`
 - Pro: terminal, established, very good markdown support, block format with ESC-q, keyboard shortcuts also used in bash, and their shells have a python and markdown mode
 - Cons: some users have a hard time remembering the keyboard shortcuts, the editor may get stuck in some unknown mode that you activated by accident. ALL this is easy to fix by remembering CTRL-X-s (save) and CTRL-G (get out of a strange mode)
- `vi`
 - use vim instead, vi is available on all Linux machines, but has rather archaic editing controls.
- `vim`
 - Pros: like vi, but with cursor control
 - Cons: often awkward to use
- `Pycharm`
 - Pro: best Python editor
 - Cons: needs lots of resources, the steep learning curve
- `VS Code from Microsoft`
 - Pro: often used on Raspberry
 - Cons: Pycharm seems to have more features from the start

1.3.3 Documentation with Markdown

We have developed a special booklet that helps you to feel the many features of markdown that can be used to create scientific documentation please follow the link to this document to find out more. For open-source projects, you often need to develop documents that are portable across platforms and do not depend on editors provided by a specific vendor. They exist a great number of formats that could be used but most recently markdown has become one of the easier-to-use document formats For developing portable documentation.

- Scientific Writing with Markdown, Gregor von Laszewski

There are many editors for editing markdown locally on your computer or and services that are offered remotely. For groups one of the services that we have successfully used is Hackermd.io. On macOS one of the very useful editors at macdown. However, emacs already provides a built-in markdown editor for you if. If you are using PyCharm it also comes with a MacDown editor but you cannot enable it if it does not come with your version.

For complex documents, the management of the document with Github is superior to those of online editors. This is especially the case if the documents have to be split up into multiple documents due to their size.

1.3.4 ToDo Lists

It is important to communicate quickly some tasks in documents that we write as a team. To do this we use the keyword TODO, followed usually by an explanation what needs to be done. As a TODO can be hopefully resolved quickly it should be able to complete them in 1-2 hours. Any TODO that may take longer we also add to our GitHub project for it to be recorded and if we identify delays in its execution we can assign additional team members to help on these larger tasks.

Once a team member has identified a TODO item, the team member can simply put his name behind it, as well as the date and time of teh anticipated completion so others know you work on it. Also, it is recmmended to communicate on slack about the task you do if you run into issues or have questions.

In case a TODO has not yet been assigned, a team member can simply tak it and complete it. In practive we have seen in some groups that the project lead needs to assign tasks to speed up the development and avid length times of inactivity by team members.

1.3.5 Git and GitHub

Git is a distributed version control system to support working on project in teams while allowing different team members to contribute and to curate the contribution through reviews.

GitHub is a service offered for free with the limitation that the repositories should not be larger than 1GB and the individual files must be smaller than 200MB. Github is very popular for OpenSource projects and through its free offering allows community building around OpenSource Projects.

1.3.5.1 Git from the command line

Git can easily be installed on all platforms including

- macOS: You will need to install Xcode which includes not only git but user Linux programs such as Makefile
- Ubuntu: You can install it via `apt install git`
- Windows: You can install it via *Git Bash* which is distributed from <https://git-scm.com/downloads>. When installing read carefully the available options. We recommend you install a desktop shortcut.

1.3.6 Git form IDEs

Pycharm is one of the best editors for Python. It does provide build-in support to interact with GitHub. For beginners, we do recommend to get started with PyCharm's GitHub features.

1.3.7 GitHub from a GUI

Some may fancy using a Graphical user interface to interact with GitHub. However, in many cases, the terminal access is simpler. However, if you like to browse the repositories and see the commit tree, these GUI interfaces are useful. Several such interfaces are available at:

- <https://git-scm.com/downloads/guis>

1.3.7.1 GitHub Commands Many tutorials introduced get up in its full details however many of the contributing participants in open source project may not need them. That is it important to identify which commands are the most useful ones to get team members started

What are the most important commands?

- `git pull`
 - retrieves the latest content from the shared remote repository
- `git add FILENAME`
 - adds a filename to the local repository. Also, do never use the git command `git add .` as that adds all files and you could have files that you do not want to commit. instead **always** use `git add FILENAME` , where FILENAME is the file you like to add.

- `git commit -m "commit comment" FILENAME`
 - commits the current content of the FILE to the local repository
- `git push`
 - pushes the content from the local repository to the shared remote repository

More advanced feature of git is the use of branches. With branches we can coordinate individual contributions that are merged into the main remote repository. This is done with a branch name as well as a git pull request.

- `git checkout BRANCHNAME`
 - creates a new branch in your local repository
- `git push -u origin BRANCHNAME`
 - pushes the branch with the name BRANCHNAME to the remote repository for all to see

After your branch may be reviewed by the team it may be decided to merge it into the main branch. This has to be done carefully and multiple people should review such a merge before it is executed and pushed. You can try out a branch by checking it out with the command

```
$ git checkout BRANCHNAME
```

After the check out you have the contents of the branch in your local repository and you can verify if it works. Usually one person or team members are dedicated to merging branches. A branch can be merged with the commands

```
$ git checkout main  
$ git merge main BRANCHNAME
```

To push the branch into the remote repository simply use the push command.

```
$ git push
```

1.3.8 Appendix

1.4 Github



Learning Objectives

- Be able to use the github cloud services to collaboratively 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). Then we will introduce you how you can contribute to this set of documentations with pull requests.

1.4.1 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.

1.4.2 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.

1.4.3 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

1.4.4 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

1.4.5 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>

1.4.6 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

1.4.7 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

1.4.8 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

1.4.9 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

1.4.10 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.

1.4.11 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

1.4.12 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.

1.4.13 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.**

1.4.14 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.

1.4.15 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
```

1.4.16 README.yml and notebook.md

In case you take classes e516 and e616 with us you will have to create a README.yml 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.yml 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 any 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.

1.4.17 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 clone 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.

1.4.17.1 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.

1.4.17.2 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>

1.4.18 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.

1.4.19 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 to a 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/>

1.4.19.1 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.

1.4.19.2 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 for github's markdown that integrate well to support project management.

1.4.19.2.1 Task lists Task 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 simply 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 off 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 of the task text, you need to escape it with a `\`

1.4.19.2.2 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/>

1.4.19.2.3 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.

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

1.4.19.3 Notifications Github allows you to set preferences on how you like 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.

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

1.4.19.5 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.

1.4.20 Glossary

The Glossary is copied from

- <https://cdcv.s.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 be 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.

1.4.21 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://cdcv.s.fnal.gov/redmine/projects/cet-is-public/wiki/GitTipsAndTricks#A-suggested-work-flow-for-distributed-projects-NoSY>

1.4.21.1 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>
```

1.4.21.2 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
```

1.5 Class Git

This class uses 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 GitHub 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

