SESSION 1

greenwich.edu.vn

UNIVERSITY of GREENWICH

Alliance with FPT Education

VERSION CONTROL SYSTEMS

- Version control (or revision control, or source control) is all about managing multiple versions of documents, programs, web sites, etc.
  - Almost all “real” projects use some kind of version control
  - Essential for team projects, but also very useful for individual projects
- Some well-known version control systems are CVS, Subversion, Mercurial, and Git
  - CVS and Subversion use a “central” repository; users “check out” files, work on them, and “check them in”
  - Mercurial and Git treat all repositories as equal
- Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

WHY VERSION CONTROL?

- For working by yourself:
  - Gives you a “time machine” for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes
- For getting an internship or job:
  - Any company with a clue uses some kind of version control
  - Companies without a clue are bad places to work

WHY GIT?

- Git has many advantages over earlier systems such as CVS and Subversion
  - More efficient, better workflow, etc.
  - See the literature for an extensive list of reasons
  - Of course, there are always those who disagree
- Best competitor: Mercurial
  - I like Mercurial better
  - Same concepts, slightly simpler to use
  - In my (very limited) experience, the Eclipse plugin is easier to install and use
  - Much less popular than Git

DOWNLOAD AND INSTALL GIT

- There are online materials that are better than any that I could provide
- Here’s the standard one: http://git-scm.com/downloads
- Here’s one from StackExchange: http://stackoverflow.com/questions/315911/git-for-beginners-the-definitive-practical-guide#323764

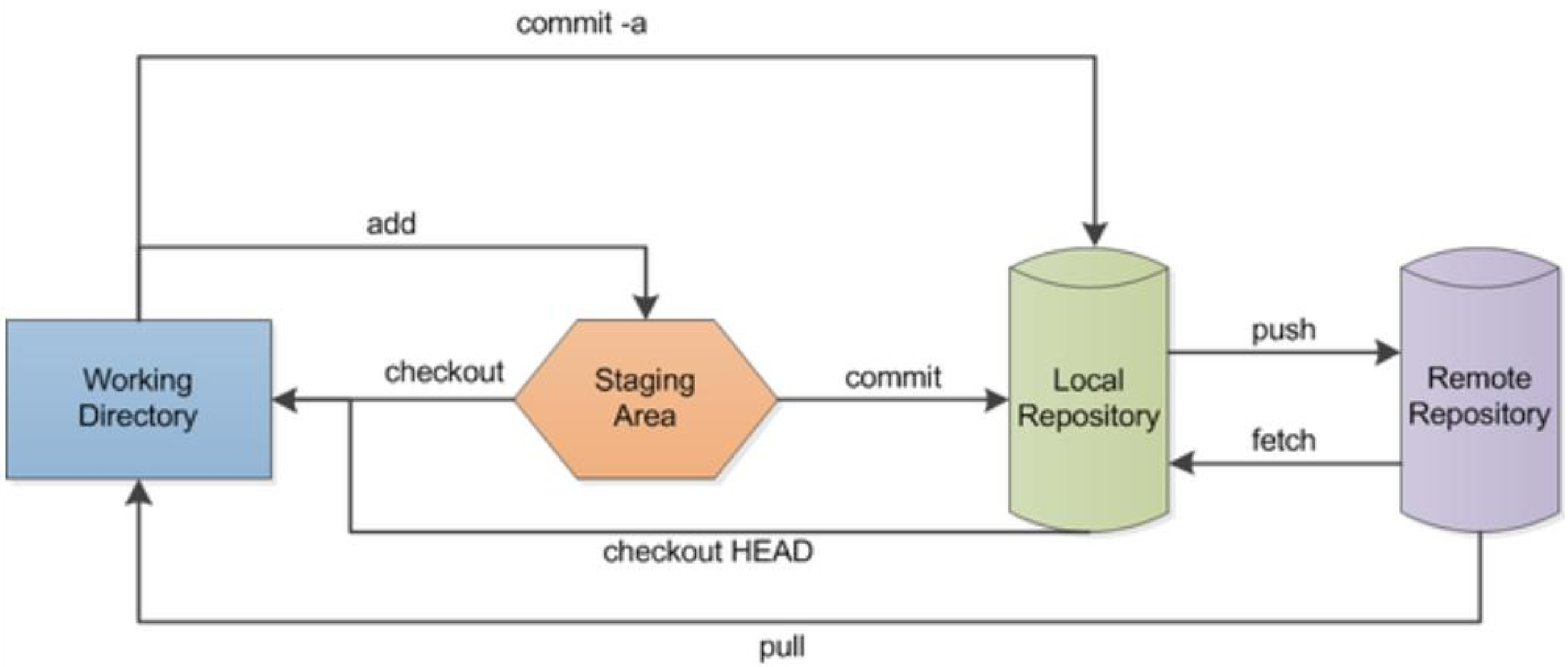
- Note: Git is primarily a command-line tool
- I prefer GUIs over command-line tools, but…
- The GIT GUIs are more trouble than they are worth (YMMV)

SOME TERMINOLOGIES

- *Repository*: Where your project (including all files across all branches) is stored
- *Commit*: Saves the changes you have made to the files in the repository since the last time you committed. Remember that commits are local until you push them to the repository. When you commit, you are essentially “staging” the changes you make to be later pushed to the remote branches of the repository for other users to see publicly.
- *Branch*: By creating a new Branch, you make a new copy of the repository in its current state, allowing you to develop and modify features without changing other branches/versions. When a branch is completed or when it reaches a desired checkpoint, it can be merged with other branches

SOME TERMINOLOGIES(2)

- *Push*: Saves local commits to the remote repository
- *Pull*: Retrieve the most recent versions of the files in the repository and merge . Branches that you pull will be merged into the current local branch, so make sure that you only pull from the remote branch that corresponds to the branch you are currently working in
- *Fetch*: Retrieve the most recent version of the files in remote branches without merging them into your local branches. You can do a git fetch at any time to update your local copy of a remote branch. This operation never changes any of your own branches and is safe to do without changing your working copy.
- *Checkout Revision*: Copies a remote branch to your local repository (or switches to a different local branch).



commit -a: Directly commit modified and deleted files into the local repository (*no new files!*)
add: Add a file to the staging area.
checkout: Get a file from the staging area.
checkout HEAD: Get a file from the local repository
commit: Commit files from the staging area to the local repository
push: Send files to the remote repository
fetch: Get files from the remote repository
pull: Get files from the remote repository and put a copy in the working directory

INTRODUCE YOURSELF TO GIT

- Enter these lines (with appropriate changes):
  - git config --global user.name "John Smith"
  - git config --global user.email jsmith@seas.upenn.edu
- You only need to do this once

- If you want to use a different name/email address for a particular project, you can change it for just that project
  - cd to the project directory
  - Use the above commands, but leave out the --global

CREATE AND FILL A REPOSITORY

1. cd to the project directory you want to use
2. Type in git init
   - This creates the repository (a directory named .git)
   - You seldom (if ever) need to look inside this directory
3. Type in git add .
   - The period at the end is part of this command!
     - Period means "this directory"
   - This adds all your current files to the repository
4. Type in git commit –m "Initial commit"
   - You can use a different commit message, if you like

CLONE A REPOSITORY FROM ELSEWHERE

- git clone *URL*
- git clone *URL mypath*
  - These make an exact copy of the repository at the given URL
- git clone git://github.com/*rest_of_path/file.git*
  - Github is the most popular (free) public repository
- All repositories are equal
  - But you can treat some particular repository (such as one on Github) as the "master" directory
- Typically, each team member works in his/her own repository, and "merges" with other repositories as appropriate

THE REPOSITORY

- Your top-level **working directory** contains everything about your project
  - The working directory probably contains many subdirectories—source code, binaries, documentation, data files, etc.
  - One of these subdirectories, named .git, is your repository
- At any time, you can take a "snapshot" of everything (or selected things) in your project directory, and put it in your repository
  - This "snapshot" is called a commit object
  - The commit object contains (1) a set of files, (2) references to the "parents" of the commit object, and (3) a unique "SHA1" name
  - Commit objects do *not* require huge amounts of memory
- You can work as much as you like in your working directory, but the repository isn't updated until you commit something

INIT AND THE .GIT REPOSITORY

- When you said `git init` in your project directory, or when you cloned an existing project, you created a repository
  - The repository is a subdirectory named `.git` containing various files
  - The dot indicates a “hidden” directory
  - You do *not* work directly with the contents of that directory; various git commands do that for you
  - You *do* need a basic understanding of what is in the repository

MAKING COMMITS

- You do your work in your project directory, as usual
- If you create new files and/or folders, they are *not tracked* by Git unless you ask it to do so
  - git add *newFile1 newFolder1 newFolder2 newFile2*
- Committing makes a "snapshot" of everything being tracked into your repository
  - A message telling what you have done is required
  - git commit –m "Uncrevulated the conundrum bar"
  - git commit
    - This version opens an editor for you the enter the message
    - To finish, save and quit the editor
- Format of the commit message
  - One line containing the complete summary
  - If more than one line, the second line must be blank

COMMITS AND GRAPHS

- A commit is when you tell git that a change (or addition) you have made is ready to be included in the project
- When you commit your change to git, it creates a commit object
  - A commit object represents the complete state of the project, including all the files in the project
  - The *very first* commit object has no "parents"
  - Usually, you take some commit object, make some changes, and create a new commit object; the original commit object is the parent of the new commit object
    - Hence, most commit objects have a single parent
  - You can also merge two commit objects to form a new one
    - The new commit object has two parents
- Hence, commit objects form a **directed graph**
  - Git is all about using and manipulating this graph

WORKING WITH YOUR OWN REPOSITORY

- A head is a reference to a commit object
- The "current head" is called HEAD (all caps)
- Usually, you will take HEAD (the current commit object), make some changes to it, and commit the changes, creating a new current commit object
  - This results in a linear graph: A → B → C → …→ HEAD

- You can also take any previous commit object, make changes to it, and commit those changes
  - This creates a branch in the graph of commit objects
- You can merge any previous commit objects
  - This joins branches in the commit graph

COMMIT MESSAGES

- In git, “Commits are cheap.” Do them often.
- When you commit, you must provide a one-line message stating what you have done
  - Terrible message: “Fixed a bunch of things”
  - Better message: “Corrected the calculation of median scores”
- Commit messages can be very helpful, to yourself as well as to your team members
- You can’t say much in one line, so commit often

CHOOSE AN EDITOR

- When you "commit," git will require you to type in a commit message
- For longer commit messages, you will use an editor
- The default editor is probably `vim`
- To change the default editor:
  - `git config --global core.editor /path/to/editor`
- You may also want to turn on colors:
  - `git config --global color.ui auto`

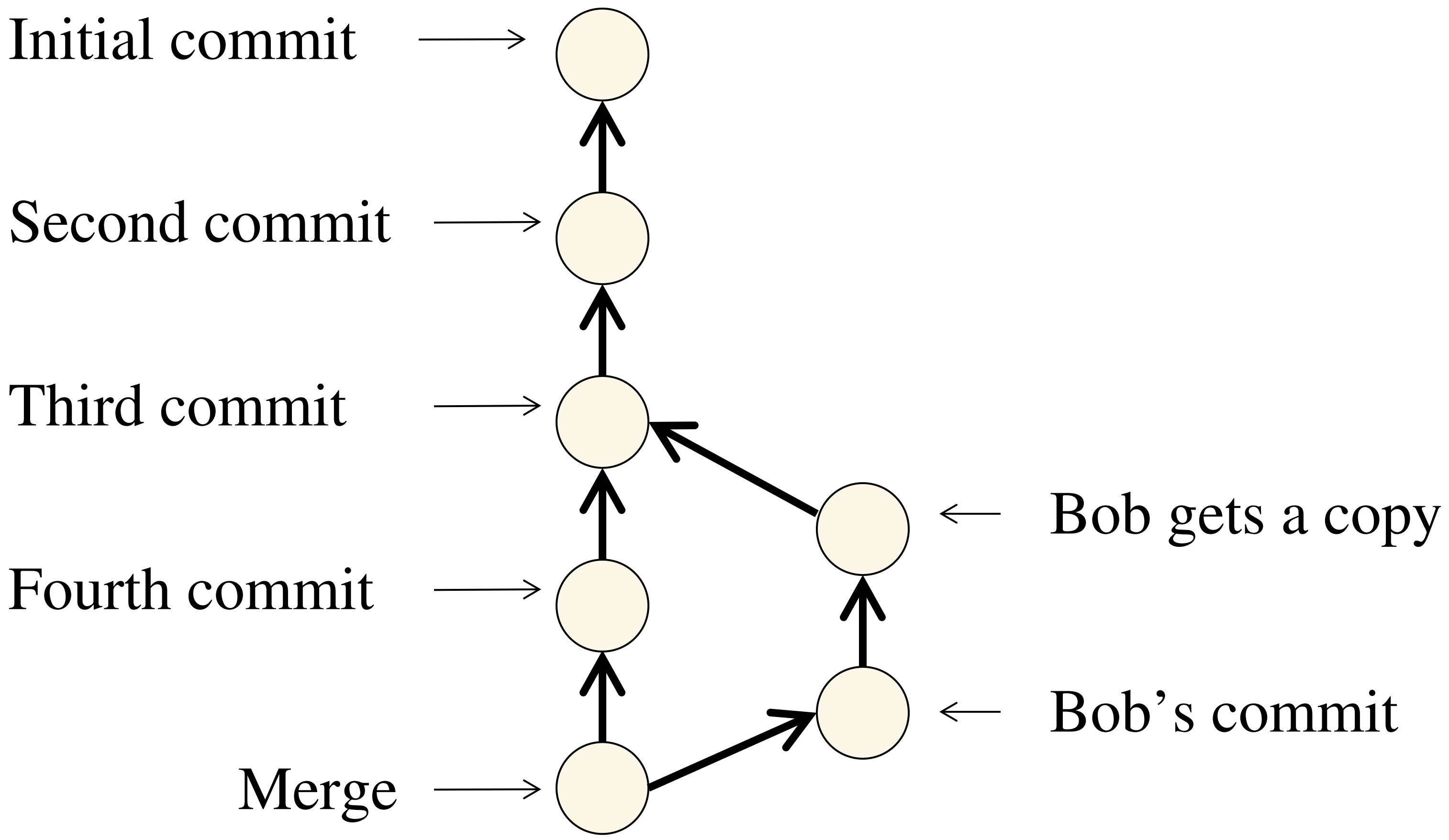
WORKING WITH OTHERS

- All repositories are equal, but it is convenient to have one central repository in the cloud
- Here's what you normally do:
  - Download the current HEAD from the central repository
  - Make your changes
  - Commit your changes to your local repository
  - Check to make sure someone else on your team hasn't updated the central repository since you got it
  - Upload your changes to the central repository
- If the central repository *has* changed since you got it:
  - It is *your* responsibility to **merge your two versions**
    - This is a strong incentive to commit and upload often!
  - Git can often do this for you, if there aren't incompatible changes

TYPICAL WORKFLOW

- git pull *remote_repository*
  - Get changes from a remote repository and merge them into your own repository
- git status
  - See what Git thinks is going on
  - Use this frequently!
- Work on your files (remember to add any new ones)
- git commit –m "*What I did*"
- git push

MULTIPLE VERSIONS



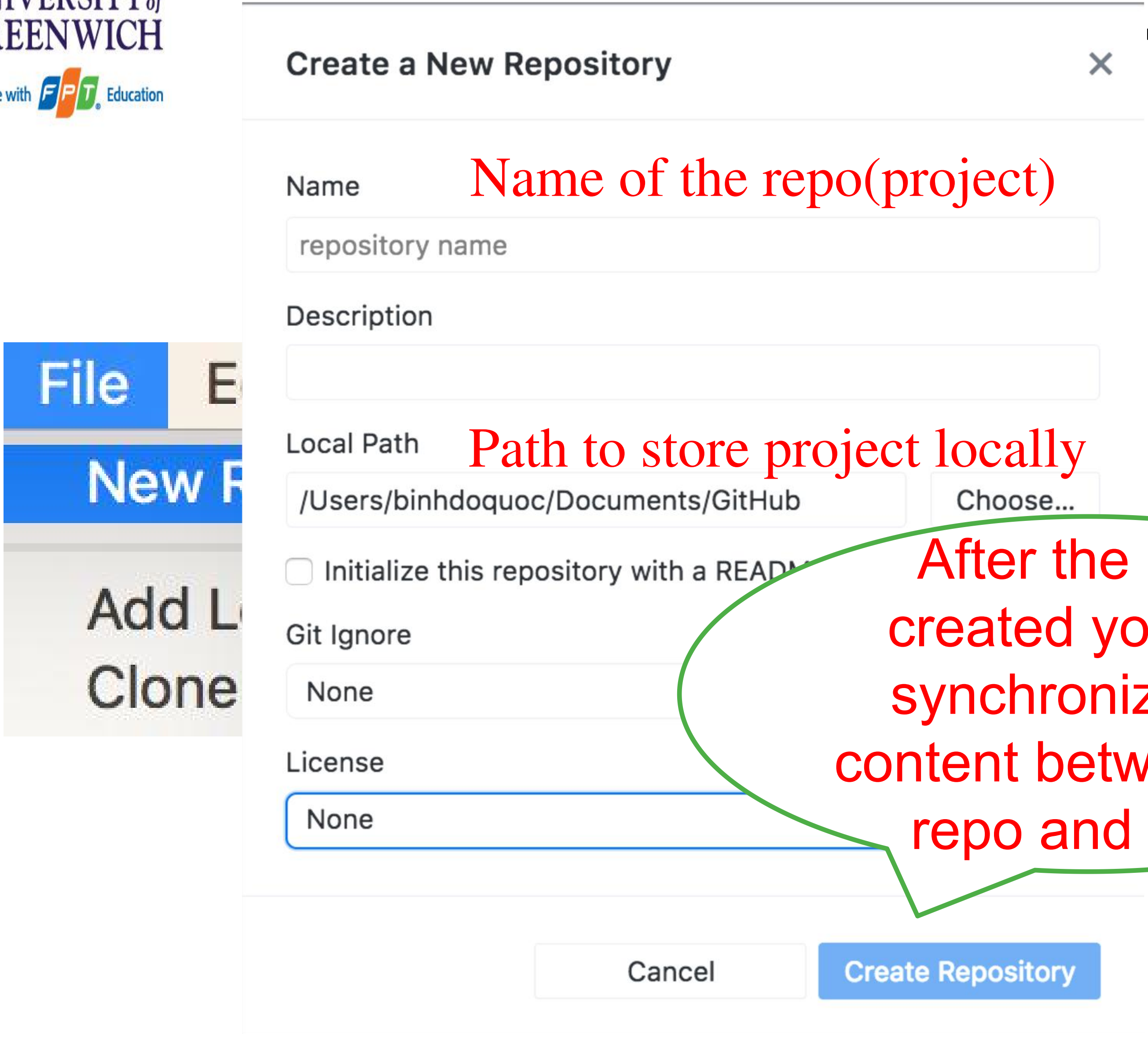
KEEPING IT SIMPLE

- If you:
  - Make sure you are current with the central repository
  - Make some improvements to your code
  - Update the central repository before anyone else does
- Then you don't have to worry about resolving conflicts or working with multiple branches
  - All the complexity in git comes from dealing with these
- Therefore:
  - Make sure you are up-to-date before starting to work
  - Commit and update the central repository frequently
- If you need help: https://help.github.com/

WORKING WITH DESKTOP GITHUB

- Working with command line is not very easy for many people
- There is a tool which helps people don’t like to work with command line: Desktop Github
- This is a free tool working on both Windows and Mac
- Go to the website to download https://desktop.github.com

TORY



Create a New Repository

Name — Name of the repo(project)

repository name

Description

Local Path — Path to store project locally

/Users/binhdoquoc/Documents/GitHub

Choose...

Initialize this repository with a READM

Git Ignore: None

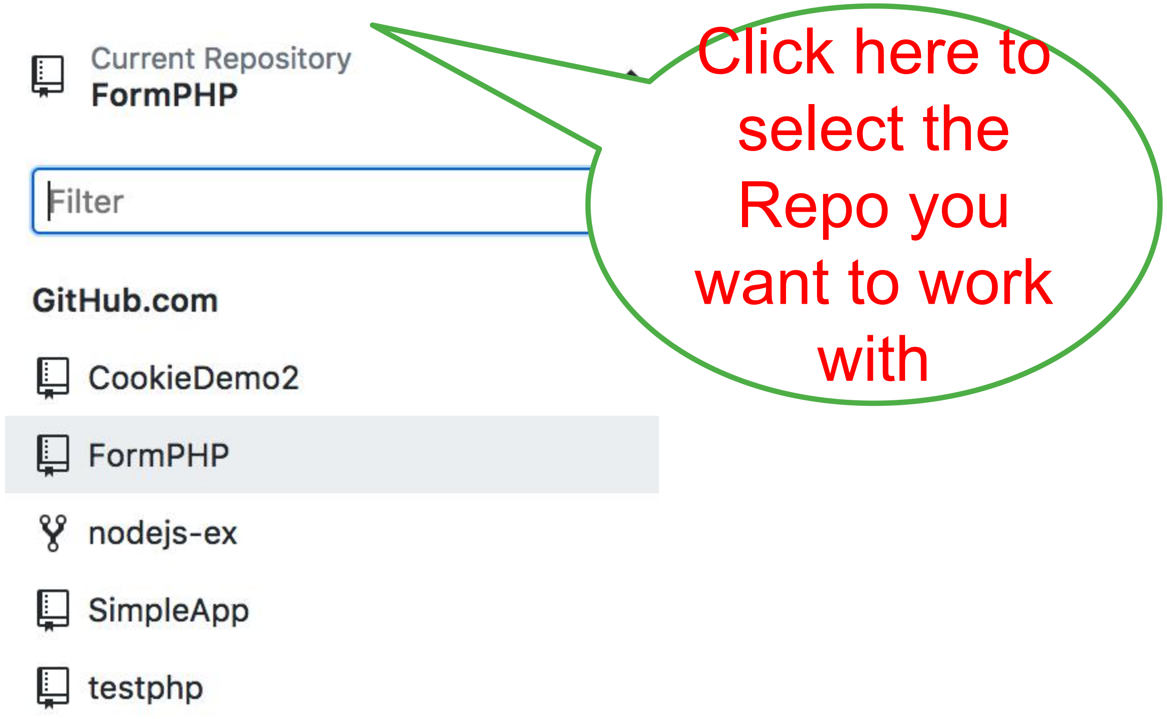
License: None

Cancel

Create Repository

After the repo created you can synchronize the content between the repo and local

SELECT AN AVAILABLE REPOSITORY TO WORK WITH


Current Repository
FormPHP

Filter

GitHub.com

CookieDemo2

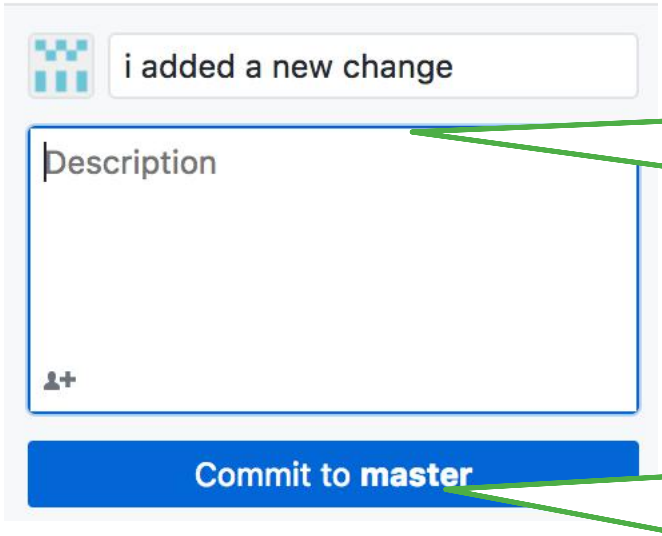
FormPHP

nodejs-ex

SimpleApp

testphp

COMMIT CHANGES TO A BRAND

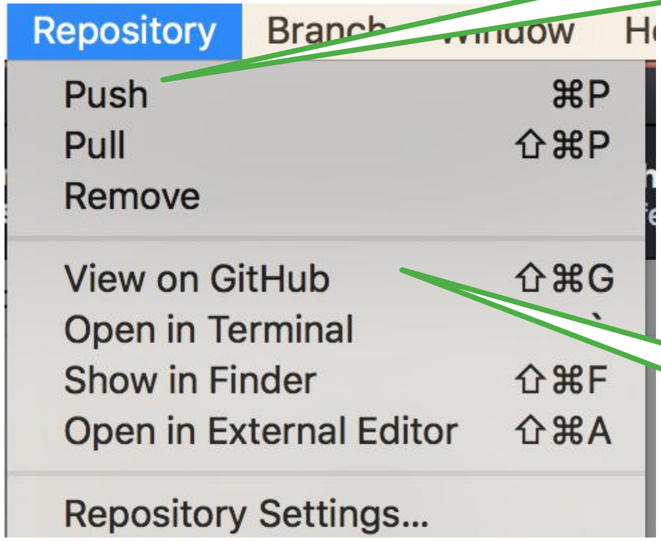


Enter a comment

More description if needed

Click here to submit changes from local

PUSH OR PULL FROM REPOSITORY

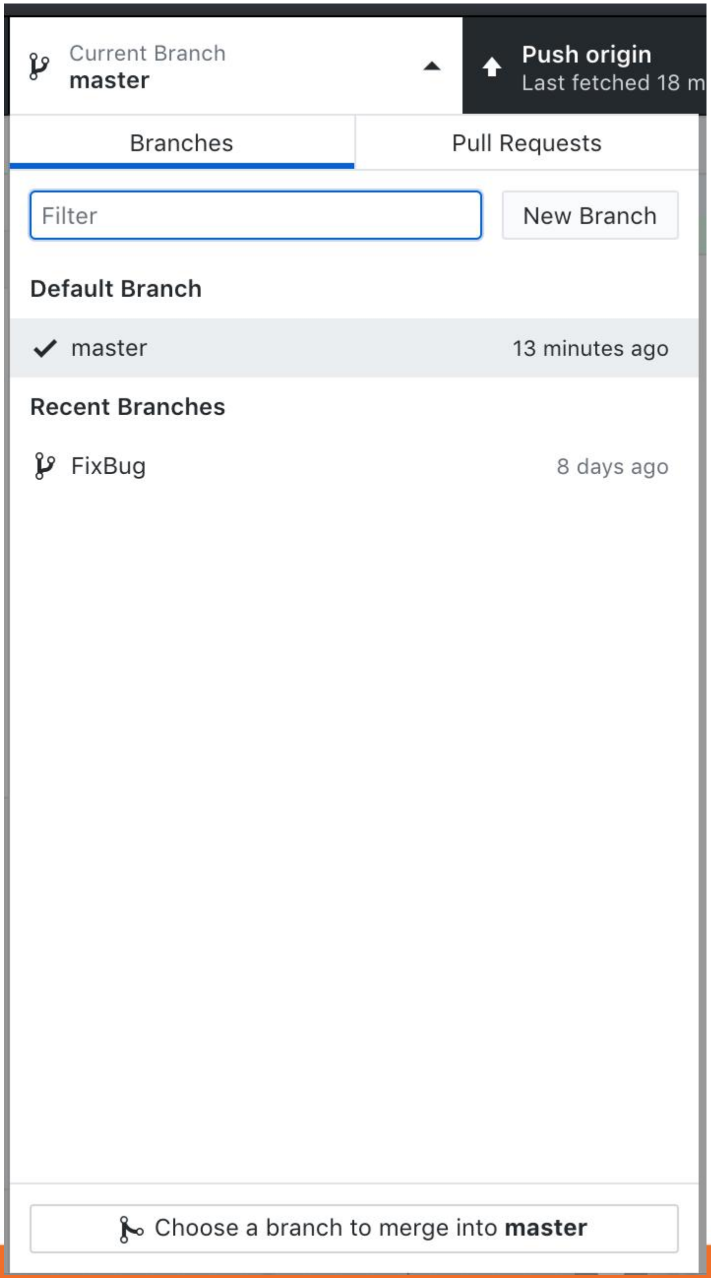


Update Repo with local changes

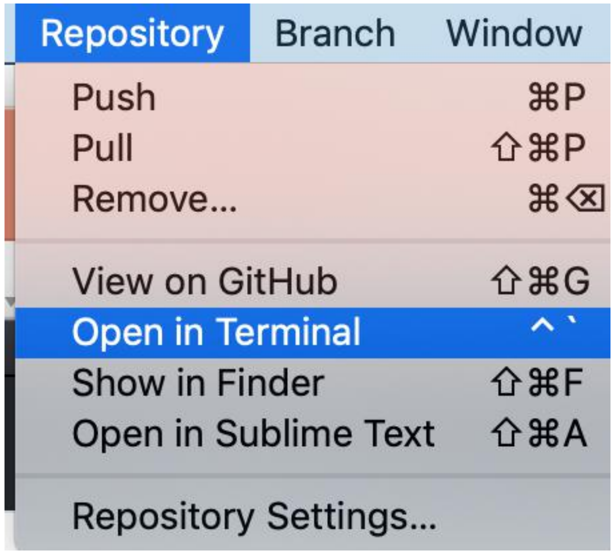
changes from Repo

View Repo on Github

You can create a new brand than merge it in to in another branch

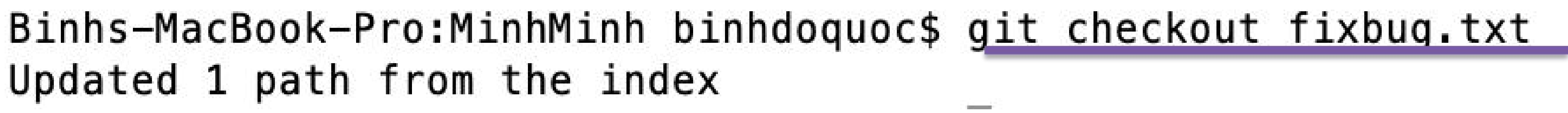

Current Branch
master
Push origin
Last fetched 18 m
Branches
Pull Requests
Filter
New Branch
Default Branch
master 13 minutes ago
Recent Branches
FixBug 8 days ago
Choose a branch to merge into **master**

GO TO CONSOLE SO YOU CAN TYPE COMMAND



RECOVER A FILE WHICH HAS NOT BEEN COMMITTED

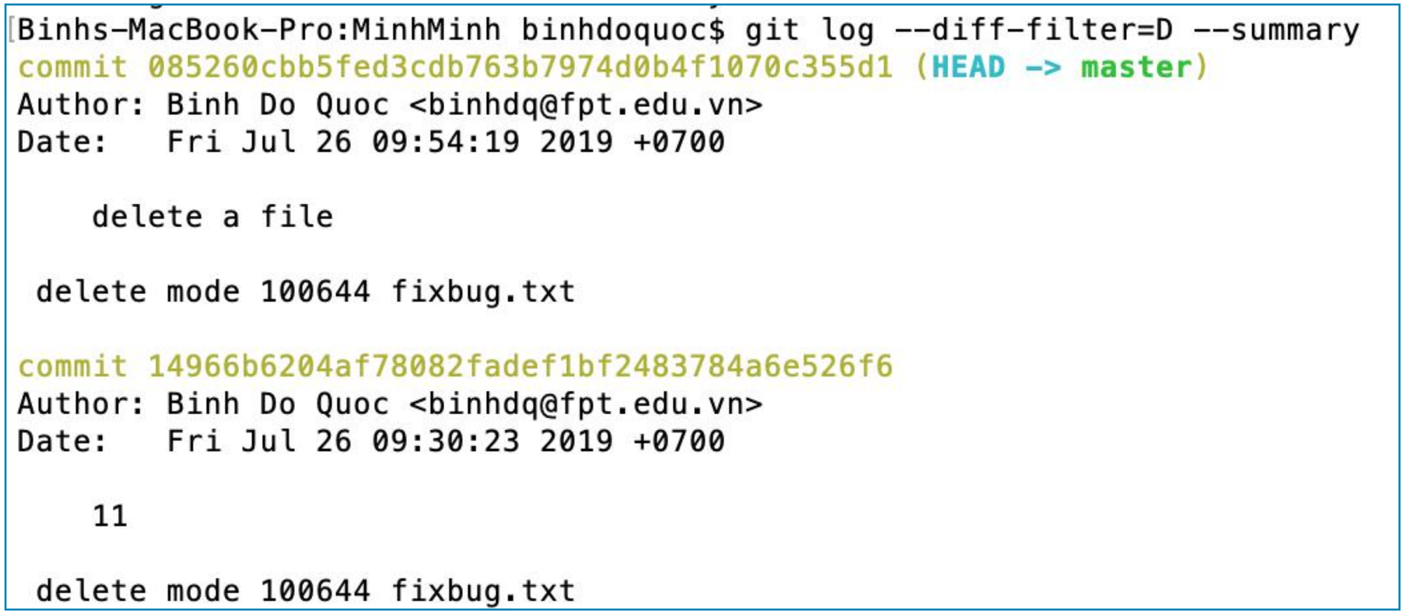
- $ git checkout -- <file>



```
Binhs-MacBook-Pro:MinhMinh binhdoquoc$ git checkout fixbug.txt
Updated 1 path from the index
```

RECOVER A FILE WHICH HAS BEEN COMMITTED(1)

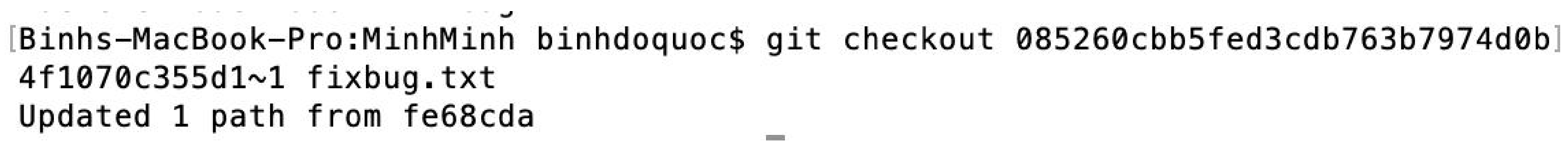
1. Search for the path of deleted file
   - git log --diff-filter=D --summary



```
Binhs-MacBook-Pro:MinhMinh binhdoquoc$ git log --diff-filter=D --summary
commit 085260cbb5fed3cdb763b7974d0b4f1070c355d1 (HEAD -> master)
Author: Binh Do Quoc <binhdq@fpt.edu.vn>
Date:   Fri Jul 26 09:54:19 2019 +0700

    delete a file

 delete mode 100644 fixbug.txt

commit 14966b6204af78082fadef1bf2483784a6e526f6
Author: Binh Do Quoc <binhdq@fpt.edu.vn>
Date:   Fri Jul 26 09:30:23 2019 +0700

    11

 delete mode 100644 fixbug.txt
```

RECOVER A FILE WHICH HAS BEEN COMMITTED(2)

2. Using check out

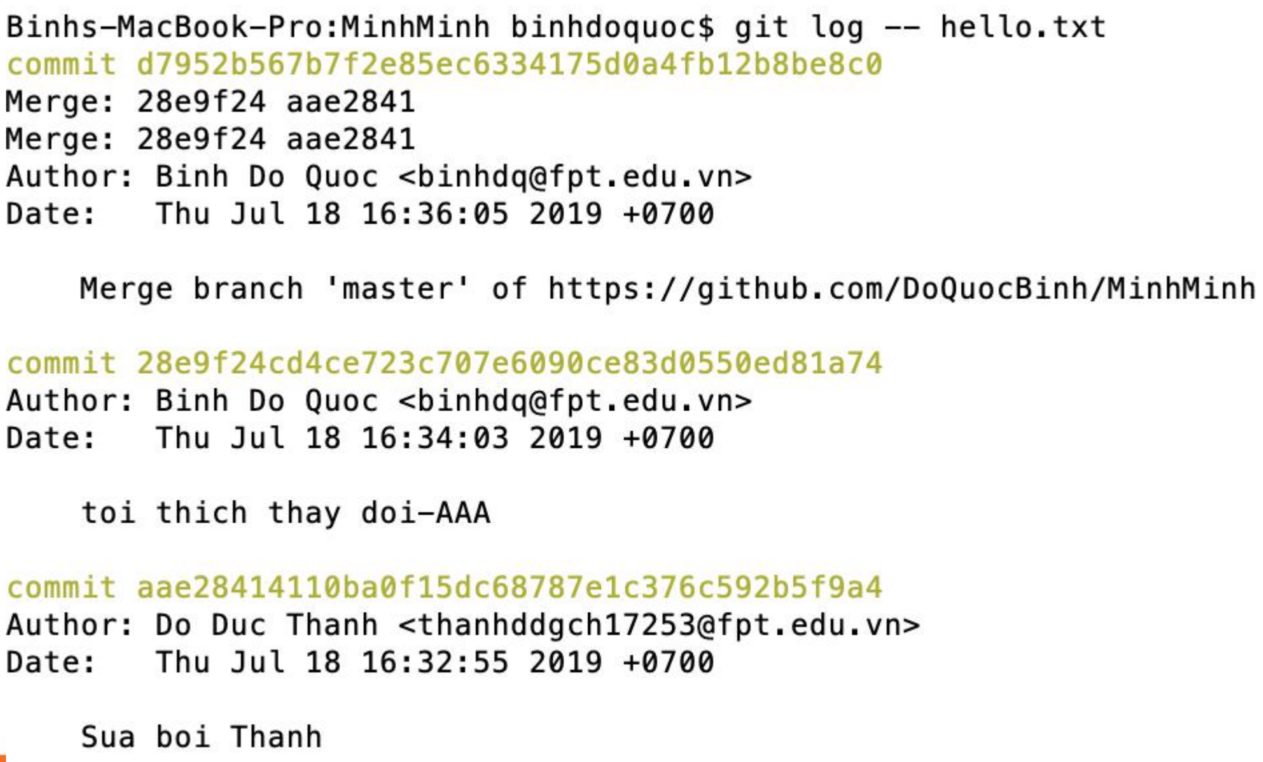


```
[Binhs-MacBook-Pro:MinhMinh binhdoquoc$ git checkout 085260cbb5fed3cdb763b7974d0b]
4f1070c355d1~1 fixbug.txt
Updated 1 path from fe68cda
```

3. $ git commit –m ”I recover my file”
4. $git push

TO VIEW A FILE HISTORY

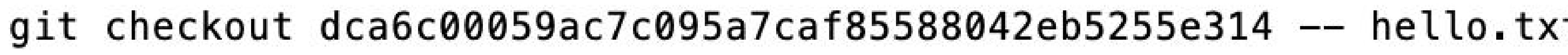
- $ git log -- file



```
Binhs-MacBook-Pro:MinhMinh binhdoquoc$ git log -- hello.txt
commit d7952b567b7f2e85ec6334175d0a4fb12b8be8c0
Merge: 28e9f24 aae2841
Merge: 28e9f24 aae2841
Author: Binh Do Quoc <binhdq@fpt.edu.vn>
Date:   Thu Jul 18 16:36:05 2019 +0700

    Merge branch 'master' of https://github.com/DoQuocBinh/MinhMinh

commit 28e9f24cd4ce723c707e6090ce83d0550ed81a74
Author: Binh Do Quoc <binhdq@fpt.edu.vn>
Date:   Thu Jul 18 16:34:03 2019 +0700

    toi thich thay doi-AAA

commit aae28414110ba0f15dc68787e1c376c592b5f9a4
Author: Do Duc Thanh <thanhddgch17253@fpt.edu.vn>
Date:   Thu Jul 18 16:32:55 2019 +0700

    Sua boi Thanh
```

TO RESTORE A FILE TO A SPECIFIC POINT

- git checkout <commit_hash> -- <file>



```
git checkout dca6c00059ac7c095a7caf85588042eb5255e314 -- hello.tx
```

EXERCISE 1

- Create a repository by Desktop Github(or by command line)
- Create a text file and push it to the Repo
- Share the Repo with your friend
- Ask your friend change the file
- Update your local from the repo
- Check that if you can see the changes made by your friend

EXERCISE 2

- What is the difference between commit and push?
- What is brand concept in Git?
- Create a new brand and update a change in that brand