Pontificia Universidad
JAVERIANA
Bogotá

Introduction to git and github

Notes of class

Iván Andrés Trujillo Abella

Facultad de ingeniería

1 Introduction

Git it is a open source technology or control system tool. Source Control Management that could be dowlan in www.git-smc.com.

Git is a distributed system:

```
Local machine   A -->  Remote repository <-- B machine Local
```

to check if already have installed git:

```
git --version
```

```
git [command] [--flags] [arguments]
```

to ask help

```
git help add
```

```
diff [-u]
c \\change
a \\ added}
```

We can uses vim as text editor and put all in the sequence to make running files thus we made with python, this allow us to run very esasy.

in a specifical folder:

```
‘‘‘bash
cd ~/
mkdir testdiff

cat > text1.txt
this text is composed
of three lines:
line1
```

```
line2
line3
‘‘‘
```

```
‘‘‘bash
cat > text2.txt
this text is composed
of two lines:
line1
line2
‘‘‘
```

```
‘‘‘bash
diff text1.txt text2.txt
‘‘‘
```

there are another commands with the equall purpose. **wdiff**, **vimdiff**.

Apply changes

we can find two version of the same document, now we can apply or not the changes.

Now we are interested in make *pipelines*.

```
diff -u old_file new_file.2 > diference.diff
```

this is very useful, but the mainly objective of my work it is create this kind of model.

a text with diff format, this is very practical due we can uses, the command patch to apply the changes.

patch

```
‘‘‘bash
patch text_to_apply_changes < text_with_changes.diff
‘‘‘
```

Lab seasson

```
‘‘‘bash
cat > document.txt
World Wide Web was designed to communicate computers
the universities contribute in a important way to this develop.
cp document.txt document_original.txt
‘‘‘
```

We send to a the collegue and receive its modifications.

```
‘‘‘bash
cat document_revised.txt
World Wide Web was designed to ARPA NET project to
communicate computers, the universities from EEUU
contribute in a important way to this network project.
‘‘‘
```

```
‘‘‘bash
diff document_revised.txt documnet_original.txt
‘‘‘
```

```
‘‘‘bash
diff -u document_revised.txt documen_original.txt  > changes.diff
‘‘‘
```

And if we think that the changes are good then we can accept it and modify our document.

```
‘‘‘bash
patch document.txt < changes.diff
‘‘‘
```

take in mind that a commit it is a session that modified files, and take a snapshot in these moment.

2 Shell Scripting

How to execute a shell script? By default script not can run a file, you must give the permission.

```
chmod + x script_name.sh
```

To execute the script we can put:

```
./script_name.sh
```

Now we can make a lab session, to deal with a this programs. In order to take automatic processes.

```
yum install git
sudo apt-get install git
```

```
‘‘‘bash
git config --global user.name ’Iván Trujillo’
git config  --global user.email ’ivantrujillo1229@gmail.com’
‘‘‘
```

” –global indicate to all repositories that we will create.
get the information

```
git config user.name
git config user.email
```

you run in a specify repository if there not local values settings then return global configuration.
To start to work the are some ways of create from scratch a repository or work with a a existing repository, then we can uses **git init** or **git clone**.
In detail **git init** initialized a empty repository in the current directory.

```
ls -l .git/
ls -a
```

One important concept in git it is staging area or index. This is a file, that contain all information upon the changes and files in the following commit.
for instance suppose that now we are in a repo. then, we can add modify a file if you type

```
‘‘‘bash
git diff
‘‘‘
```

then this show us the difference among files, that are commited to the repo. then with

```
‘‘‘
git add .
‘‘‘
```

Think in a flow ”Set of changes” $- >$ commit $- >$ If you modify you must said to git which changes will be loaded

```
git status
```

give us information about the working tree and pending changes.
A file could be in the working tree but not in staging are this could be named as **untracked files**.
we can add a single file to the staging area:

```
git add NAME_FILE
```

'appear in changes to be commited'

NOTE THAT IN WORKING THREE ONLY WILL BE TRACKED MODIFIED FILES.

defintion of staging area (index) :

a file maintained by Git that contains all of the information about what files and changes will be commited.

```
git diff file
```

```
git commmit -a
```

with this option we save the uses of add but only in tracked files, for new files it is necessary uses add. This can be used with -m option also.

```
git commit -a -m 'new line'
```

there are ways of see the changes in files, this is very important due allow us take in mind, what are the additions to broke the code.

```
git diff
```

git diff only show by default unstaged changes, namely that are not currently in the staging area.

to see staged we need add:

```
git diff --staged
```

```
git add -p
```

You add a directory and this start out to track this folder(all files).

```
git status -s
```

git log show us the list of commits made in the current git we can uses to show differences see that the pointer HEAD $->$ MASTER

git log -p [file_name]

we can see how act this command a brief description. Notice that file$_nameitisoptional.$

we can make uses of **git show ID** where ID could be seen in **git log**.

another practical flag it is **git log –stat**

```
git diff ID_commit
```

to compare the actual with this version.

2.1 Locations

```
Working tree --> staging area --> local repository --> remote repository
             --> add --> commit --> push
```

- working tree the place where the files are added, view, removed or edit to the next commit.
- staging area files that will be commited
- local repository contain the history (commits) of a project
- remote repository contain the commits usually in the cloud.

At working tree there are *.git* file where the staging area and the local repository are saved.

3 rename or delete files in repository

git mv to rename files in repositories.

```
git mv old_name new_name
```

4 .gitignore

We can ignore files as produced by LaTeX when they are compiling that we need it is save this in a txt file and add its names for instance:

```
cat > .gitignore
file.log >> .gitignore
file.aux >>. gitignore
file.pdf >>. gitignore
```

Due this article variable files that depende the main .tex. or maybe we can using echo ¿ .gitignore

5 Undoing changes before comitting

we can uses **git checkout**

sometimes we we add all files with * or . we need not tracked one particular file then we can uses

```
git reset HEAD file
```

remember that *HEAD* it is the current "branch" of the project.

in other words **add** and **reset** commands are complementary.

6 Undoing things

When you put a commit and forget something then uses

```
git commit --ammend
```

to overwrite the previous commit.

git checkout to restore a modifed before of move to staging area.

when the changes are staged ready, we can uses **git reset** remenber that add and reset are complementary.

7 Rollback

Suppose that you need rollback a previous version of a file. Suppose that the new code not works well with another package for instance with pandas or matplotlib.

7.1 revert

this is a powerful command, that create a commit with the inverse last command commit. this very special due commits follow a sequence and dont lost commits.

```
git revert HEAD
```

but what happend if we want jump to another?

```
git revert commit_id
```

```
HEAD---
   |
   |
   |
  BRANCH
```

8 Notes

If you type **git checkour ID commit** create a branch. git branch to see this.

```
git log -p -1
```

the parameter number indicates what commit show in this case the first in log history or last commit made.

```
git diff id commit
```

show us differences among a commit and the current version file.

9 branch

```
 HEAD
  |
  |
  |
COMMIT 1
```

Branch are only pointers, list all branches in our repo:

```
‘‘‘bash
git branch
‘‘‘
```

to create a branch :

```
‘‘‘bash
git branch name_new_branch
‘‘‘
```

to crate and switch automatically:

```
‘‘‘bash
git checkout -b name_new_branch
‘‘‘
```

delete branch:

```
‘‘‘bash
git branch -d name_branch_to_delete
‘‘‘
```

we can **merge** this to the master branch.
Note: Remember that with **checkout** we can switch among branches.
git merge
master branch merge to another branch

```
‘‘‘bash
git merge branch_name
‘‘‘
```

10 conflicts in merge

Conflicts arise in merge, with the same line is altered. When merge is appear then the file marked where the conflict arise.

git merge –abort If there are merge conflicts (meaning files are incompatible), –abort can be used to abort the merge action.

```
git log --graph --oneline
```

push
after we commit and the working tree is clean we can make the changes in github using

```
‘‘‘bash
git push
‘‘‘
```

Note that we can change the file directly in web page, then what happens with the local repository? then if I'd like bring those changes then we need used

```
‘‘‘bash
git pull
‘‘‘
```

git pull to retrieve information about our remote repository.
how to create a way of avoid enter the email and the password every time, to push or pull?
this allow us save the password by 15 minutes

```
git config --global credential.helper cache
```

we can change this time

```
git config --global credential.helper 'cache --timeout=3600'
```

11 Pratical with git

when i'm interested in see or compare currently with older

```
‘‘‘bash
git diff

‘‘‘
```

when this is in the index (staging area).

```
‘‘‘bash
git diff commit_id
‘‘‘
```

show us the difference among the snapshot in commit$_idandHEAD$.
Note: HEAD is a nick name to the currently commit. therefore git diff HEAD do not show up nothing.
when you uses git checkout id_commit to back a snapshot, then automatically put in another branch, then to back to the previous state used

```
‘‘‘bash
git checkout master
‘‘‘
```

to revert the last one, commit you can type

```
‘‘‘bash
git revert HEAD
‘‘‘
```

12 Remote

```
‘‘‘bash
git remote -v
‘‘‘
```

to see URL of our repositories.
hotfix= revisión, a small pice of code to fix a bug.

```
‘‘‘bash
git remote show origin
‘‘‘
```

13 git fetch

it is related with updates in remote server. allow us compare before make the pull, this is very important to do not damage our code, in case of a mistake.
pull is composed of two commands fetch and merge.

13.1 to check

Only modified a file directly in github web page.

```
‘‘‘bash
git branch -r
‘‘‘
```

to check the remote repositores we can

```
‘‘‘
git remote -v
‘‘‘
```

git fetch retrieve all modifie files that are not in out local repo. and save in a hidden directory.
we can move on to branch FETCH_HEAD
note if you type

```
‘‘‘
git checkout FETCH_HEAD
git branch
‘‘‘
```

whats means

```
‘‘‘git branch -a
‘‘‘
```

to merge to master we used

```
‘‘‘git merge FETCH_HEAD
‘‘‘
```

in other words git pull is a shorthand to git fetch and git merge FETCH$_{H}EAD$.
The word origin is associated with the URL of our remote. if we'd like uses our remote we can associate a name with a URL for instance: the open source communite refer forking to the fact of clone a repository and work in it.

14 Kind of merge

14.1 fast-forward

a fast forward merge is when there are a commond ancestor therefore the label only forward to the new commit.

```
            master      branch*
              |            |
              |            |
Commit k - commit k+1 - commmit
```

then, fast forward update the label of master to branch*.

14.2 three way -merge

In this case the ancestor no is preceding inmediatly

```
                        master
                          |
                          |
Commit k - commit k+1 - commit
                 |
                 |
              Commit j - commit j+1 - commit*
```

this situation could create problems, that will must be fixed manually. the problems arise if in master try merge with commit*, then the process redirect to the common acenstor commmit k+1, but the error will be arise if the same file has differents changes in the sames lines in branches. then you must, the the error must be fixed manually.

when a document have conflicts appear the marks <<<<<<<=======>>>>>> .

15 Stashing and cleaning

16 git rebase and git merge

Both are similar, but you must avoid uses rebase in public branches, instead uses merge, and locally it is prefereable uses git rebase due save linearity in commits.

tips: Always is prefereable create a branch and after merge.

17 tips in remote

```
‘‘‘bash
git remote
git remote -v
‘‘‘
```

the default name of remote repo is *origin*.

to more information:

```
‘‘‘bash
git remote show origin
‘‘‘
```

to see branches in remote repo

```
‘‘‘bash
git branch -r
‘‘‘
```

```
‘‘‘bash
git log originr/master
‘‘‘
```

after of fetch data we also can uses:

```
‘‘‘bash
git merge origin/master
‘‘‘
```

```
git log -p -1
```

take in mind that -1 or any number represent the last change, -2 the last two changes and so on.

```
‘‘‘bash
git remote update
‘‘‘
```

this command will update, all of your branches and git fetch only the branch where you are on.

```
git log --graph --oneline --all

git log -p origin/master
```

18 Branches in remote repositories

to push a remote branch in a repository we can type:

```
‘‘‘bash
git push -u origin branch THE_BRANCH_NAME
‘‘‘
```

to remove the remote branch,

```
‘‘‘bash
git push --delete origin branch_name
‘‘‘
```

rebase is used to integrate changes among branches.

19 git bitsec

19.1 note

HEAD the last number.

19.2 squash

when apply rebase, change pick keyword by squash.

```
       commit_b3
          |
       commit_b2
          |
      commit_b1                   unified_commit*
       /                           /
      /                           /
Commit 2     >>>>>>>         commit 2
   |                            |
commit 1                     commit 1
```

```
git rebase -i HEAD~3
```

change the word in the new document.

20 An improve workflow with rebase

20.1 excersice with rebase

```
      /  Branch_commit
     /
    |
 commit x
```

after a time you update the local repository with the changes in remote repo, then:

```
  commit x*
     |
     |   Branch_commit
      /
     |
     |
  commit x
```

After the apply rebase git rebase master

```
        Branch_commit
      /
     /
     |
     |
 commit x*
     |
     |
   commit x
```

where also could be see conflicts the rebase is realized from the branch.

```
git checkout master
git rebase branch
```

```
Brach_commit
  |
  |
commit x*
  |
  |
commit x
```

20.2 Notes to rewrite history

In this section interactive rebase play a important role.

```
git commit --amend --no-edit
```

Remember that this will add the staged documents to the snapshot.
–no-edit indicate that let the before commit equal.

20.2.1 rewrite a commit

```
git rebase -i HEAD~1
```

this is for the last commit and change pick for reword after appear the vim again, and could reword the commit. if you drop a commit you drop the changes made in it. drop commit has the same logic.

21 rebase vs merge

remember that:

```
‘‘‘bash
git merge feature master
‘‘‘
```

```
‘‘‘bash
git rebase refactor
‘‘‘
```

move the current branch on top of the refactor branch.
a practical example of rebase, for instance we are working in a branch, but your partner pull a update, then rebase a linear a after merge to your branch.

21.1 Good practices

To collaborate with another persons. why is better a fetch-rebase- push workflow than a pull ?
when a update occur in the remote repo, then if we uses pull then a thre-way merge appear, instead, we need a liner history with rebase.

```
git fetch
git rebase origin/master
```

always update the repository.

21.2 a simple git pull request

Note: forking is to create a copy of a repository that belong to another person.

Note: Pull request is a commit or series of commits, that are send to the owner repo.

21.3 team work

After that we create the repo, we pull our collaborators in settings, add collaborators, they have all power to modify then we are need cautiously. the collaborators only need clone the repo in their local machines (forking not).

this is the better work flow: Always create branches to work, when you finish your new features then you can merge to the master branch. you can merge to the master but is better take a person in charge of them this changes. is better assing a person to handling merge or reviwier.

Remember work in branch. You can see diff with branches

~~~
```bash
git diff branch
```
~~~

when you put, then you consider a put request, and i can check without alter the branch.

22 Review concepts

Forking works through pull request, therefore we do not affect directly the original repo. if you are not collaborator (add) when you clon a repo, you cant not suggest changes.

fork contain main data; files and unlike, issues; branches; pull request not.

upstream= is the original repository where i fork the project or files.

it is important to know hot sync a fork:

```
git remote add upstream https://github.com/octocat/Spoon-Knife.git

git remote -v
```

We need up keep update, our fork, if upstream is very active.

conde in development.

~~~
```bash
git fetch
git merge upstream/master
```
~~~

22.0.1 Updating pull request

Once time that you submit a pull request in a branch, automatically appear on the commits in the same place. to create a different pull request you need create a different branch.

we can speak with the user who send the pull request and ask more information or examples.

```
#pick   >>>>>>  pick
#pick           squash


‘‘‘git push -f
‘‘‘
```

23 Tools to integrate

23.1 CI/CD

continuous integration and continous delivery.

pipeline remember that is the input of a program is the output of another.

```
standard input
standar output
standar error.


‘‘‘bash
history
‘‘‘
```

we can store the output of any program in a text file. with > redirect

```
‘‘‘bash
python3 sort.py > output.txt
‘‘‘
```

>> redirect without lost the content. it means that put the content at final of the document as a **append**

23.2 Grep

Global regular expressions print it is used to search patterns in text. We can uses grep to search >>>> when bugs are presented.

grep ′ >>>>′ file

by default grep, search a regular expression, this means, that include substrings where the expressions match.

23.2.1 options

> -i eliminate the casesensitivness.. -R search in all files are subdirectories. -c to count the number of times that appear. -n to show the number of line that appear. -w exact mathc this means whole word. -r to include all subdirectories -v print all lines that not contain the word. -x include all sentence whole match. grep support several files

```
grep ’>>>>>’ file1 file2 file3
```

thus search in each file.

search multiple words.

```
‘‘‘bash
grep word1 file | grep word2 file
‘‘‘
‘‘‘bash
ls | grep Doc
‘‘‘
```

remember use ’ ’ to search a whole phrase. Is better uses find to search files.

wc allow us count how many lines have a file. the output of wc is lines, words and characters.

24 Concepts in computers UNIX-like Systems

A pipe is a special file that connect the process in the flow stdout > stint. | pipe ampersand they are control operators. Pipes are unidirectional left to to right.

filter is a program to process streams and produce streams. for instance, sed, grep and awk are filters.

24.1 issues

we can close issues when commit using the keywords closes N where N is the number of issue.

24.2 Add another commands

```
git remote --verbose
```

to get information about remote repository.

a local repository could be nested to a remote repository this is useful when you already have a local repository and commits and you want add to a remote.

you only need create the repository in github,

```
git init
```

in the folder

```
%git add .
%git commit -m 'first'
git remote add origin  URL
git remote -v
git push -u origin master
```

```
git remote add
```

hint: all commits belong to a branch

push add commits from a local to a remote repository.

```
git push [-u] [repository] [branch]
```

```
git push -u origin master
```

for what is is the flag -u?

git push or git push origin master?

```
git push # by deafult is poniter to origin this means that push all branches
git push master origin # only the master branch is update  in origin.
```

Remember that is acyclic.

a branch occur if a commit have more than one child. a merge occur when a commit has more than one parent.

```
branch A ---
              \      Merge
              / ( have two parents)
branch B ----
```

```
git log --online --graph
```

24.3 git objects

- commit object
- annotated tag
- tree directories
- blob

24.4 git IDs

is the name of the git object (hash algorithm) SHA-1

24.5 YAML format

what is YAML format? thereby=de este modo. foresight=precaución, previsión

24.6 research

git remote add upstream http of the upstream it means where you fork.

to check git remote -v.

"Closes: 1

we put in origin and we can make a pull request after by github to the original or upstream.

git push origin improve-username-behavior

25 GET

get have some uses,

get= receive, obtain , buy.

0) i got some awards at university.

0.1) we've gotten 40 requested to change code.

0.12) what thins are you getting?

get // have got = have = ownership, relationship, obligation and neccesity.

-1.1) have got two computers.

-1.2) you have got to get up early in the morning.

-1.3) she has got to read the book.

for past only need had.

get= offers and request

*1) could you get me the paper please? *2) can i get you something to see while you wait?

get = reach a place.

1) call me as soon as you get to the office.

get= become / change 2) the kids got really scared when they saw the spider.

get= understand 3) i dont get it

26 Research

How revert changes to a single documment not all.

27 git log

show us, the most recent commit as the first.

28 Git with remote repository

In this case we can introduce to gitlab or github, we uses github.

after we created an account in github, and and a repo, we can clone or dowland the repo in our local machine using **git clone URL**

for instance we can modify the README.md the extension md stand for *Mark down.*

if we edited then we can used to probe how works github.

https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/syncing-a-fork

https://docs.github.com/en/enterprise/2.13/user/articles/syncing-a-fork

28.1 References

master is the default branch of the repository.

Head is a reference to the current commit.

```
git show HEAD
```

```
  master
    |
    |
Commit 1 - commit 2
               |
               |
             testing
              HEAD
```

HEAD usually point to the last commit in the branch we are on, however we can detached to any branch or commit.

hint: remember that HEAD is unique.

```
git log --oneline
```

28.2 Detached HEAD

28.3 tildes and carets

used to refer relative previous commits.

28.4 Tags

uses to special commits, for instance versions v.2.0 or v.1.0

```
git tag v1
```

if not specify git IDs then the HEAD or current commit will be tag.

only *git tag* shows the tagas in alphabetical order.

git push do not load automatically transfer tags to the remote repository. to transfer all

```
git push
```

These are very practical plugins, airline proportion a good status bar to see in what branch we are on. and fugitive as handy features as

```
Gbrowse
```

28.5 Git show

28.6 Checksums

a small data derived from another digital data, checksums is useful to data integrity (not authenticity) this piece of data for instance 123455 is derived from a checksum algorithm.

checksum hash (SHA-1) rembemer that SHA-1 are very volatile when modify a document, is a fingerprint in the integrity of data.

28.7 SHA algorithm

Tree requirements, fast, avalanche requirements. SHA-1 was published by NASA 1995.

29 Branch

Create a branch is a create a label.

30 Remote branch

insights about remote, therefore it is important to know how this work,

31 Deleting a branch label

When you delete a branch the commits associate will be dangling commits this means that will be garbage collected.

git branch -d BRANCHNAME

Usually a BRANCH is delete with the feature to improve was merge to the master branch.

then when there are not merge commits, of the branch that we want delete uses

git branch -D BRANCHNAME

you can recover dangling commits:

```
git reflog
```

anote the hash and

git checkout -b HASH-1(of dangling commit)

31.1 tracking branch

Remember that all branches are pointers to certain commits.

```
git pull
```

to bring data, that is is the remote to your local machine.

Example one workflow

```
git branch NEWBRANCH
git checkout NEWBRANCH
```

the following command is a shorthand of make the two previous:

```
git checkout -b NEWBRANCH
```

detached HEAD: if you are in a detached head state. you can back to another point and modify some points in the project or version.

you back to the SHA-1 of a commit. if you want initialized to work in this point you must create a branch label or branch and checkout to the branch thus you are again in a no detached state. hint: to resolve a detached head create a branch.

analyze this command

git push –set-upstream origin add-tex

remember that in git branch NAMEBRANCH. NAMEBRANCH is the argument of the command.

hint: remember that **checkout** updates HEAD and update working tree with its commits.

32 Merging

32.1 Kind of merges

- fast-forward merge
- merge commit
- squash merge
- rebase

32.1.1 FAST-FORWARD

move the base branch label to the tip of the topic branch. This is possible only if there are not other commits since we create the branch.

```
go to the master ( checkout)
merge the branch
delete the branch
```

you note that the fast-forward merge produce a linear history.

32.2 Merge-Commit

This allow us additional commits in master branch before of branching.

this is produced when there are commits in the master after the branching, this combine the commits a make a merge commit.

here could be arise conflicts. because merge-commit is merging the tips of each branch. this means that a merge commit is linked to multiple parents

```
checkout master
merge BRANCH
delete BRANCH
```

At this moment is good take different notebooks with different specifications as Machine learning implemntations, Statistical models, visualizations.

each team has history policies: linear history merge commits

to force to no uses fast forward:

```
git checkout master
git merge --no-ff BRANCH
git branch -d BRANCH
end{verbatim}
recursive strategy

tips:
we have a master and crate the branch of development until the branch is merge with master.

\begin{verbatim}
git log  --graph --all
```

tips: is good include in the commit where occur a merge

32.2.1 Merge conflicts

araise when the commits modify the same word in different ways

Note that if you want merge the commits of B to A you need are on B and write

```
git checkout A
git merge B
```

you can delete de branch label

33 Team work flow

for instance locally we create a branch of master or main, i finish my function and then i push the branch for the request, the request are accepted then when pull the main to update, we can update the branch of development if i go to the branch and put

```
git checkout BRANCHDEVELOPMENT
git merge master
```

Create pull request to merge, to the master branch.

33.1 clonning or forking

the difference to consider is the communication among the repositories.

when you clone a repo, you modify the repo that you clone. 1with fork this is independent.

the changes that you make in the original repo, not are transmited to the fork.

remeber the shortcut of

```
git commit -a -m 'add commit message'
```

to see the history of mergins uses

```
git log --graph --oneline
```

33.2 conlifcts

arise when there are modifications in the same line of code.

when appear the message about merge conflict you must tyoe

Automatic merge failed, fix confflics

```
git status
```

modify the lines of error and work all that you want

33.3 guideline to contributing

is better make changes and after made pull request with forking.

endeavor; esfuerzo.

```
git show remote origin
```

pull - merge workflow.

```
git add -p
```

rebase:

it is important try of not uses rebase with pushes commits.

33.4 interactive rebase

Not make this over push code. only in your local repository.

```
git rebase -i HEAD~4
```

the word pick let the commit intact, and this will appear in the editor in the following way: pick HASH-1 pick HASH-1,2 pick HASH-1,3 pick HASH-1,4 if we want to squast or melt the last three we need rewrite to: pick HASH-1 squash HASH-1,2 squash HASH-1,3 squash HASH-1,4 and only appear a commit

work with github in a collaborative TEAM: how contribute to another repositories?

```
fork a external repo
clone the repo
push changes
pull request
```

take in mind that the original repo could make another changes, that were submmited after of forking, then to update que can uses upstream.

the update version is

```
git remote -v
```

this show us where fetch and push

```
git remote add upstream URL-ORIGINAL-REPO
git remote -v
git fetch upstream
git rebase upstream/master
git push origin master --force
make the changes
and pull request
git push origin master
```

probar: fork with remote branchs, work witth de deployment branch.

In tableOne contribute to make improvements in calculating automatically the distribution of each variable and as reponses use a functionality additional try collaborate to tableone

hint: remember that by defualt remote repositores have the origin repository

to get more information about remote you can uses

```
git remote show origin
```

```
git branch -r
```

to create development branch only need

```
git checkout -b deployment
make changes
git push origin deployment
```

33.5 fetch

git fetch dowland the data from remote repos witout merge with your pure locally branches, therefore not force to accept the changes. therefore this command need git merge.

```
git log origin/master
```

```
git merge origin/master
```

create a local copy of a remote branch

```
git checkout NEW_REMOTE_BRANCH
```

```
git log -p -1
```

to all branches without automatically merge is:

```
git remote update
```

```
git log --graph --oneline --all
```

```
git log -p origin/master
```

33.6 Pushing remote branches

```
git push -u origin upstream
```

34 rebase

rebase put the branch of the tip of master altering the hasesher

```
Merge  \
  |     \
  |     commit b1
  |       |
commitp  commit b0
  |     /
  |    /
commit
  |
  |
commit
```

rebase put commit b0 and b1 over commtip keeping the history linear uses git log to see how the graph is.
hint: rember the three way merge is the algorithm of commit merge.
to delete a remote branch

```
git push --delete origin BRANCH
```

```
git rebase and the branch that will be the new base
```

think in that you do not a three way, isnteand aplly a fast forward merge if you want. take in mind that you are on in the branch alternative not master. and put your changes over HEAD of the master then.

```
git rebase master
```

now you can merge the alternative branch to master or rebase again.

```
git remote add origin https://github.com/it-ces/labHealth
```

35 Introduction on linux

why is important due, with this we don not worry about preinstallation configuration and else about the software architecture of development. if the package there is not in the server then only type

36 SSH protocol on linux

Type in terminal

```
ssh-keygen
```

this command create two files; **id_rsa** and **id_rsa.pub**

```
cat ~/.ssh/id_rsa.pub
```

copy this key and put in the official site of github. finally type:

```
eval $(ssh-agent -s)
ssh-add ~/.ssh/id_rsa
```

37 Collaborate cloud Data Science

Colab as work enviorment, drive as cloud storage and git-github as control of versions.

storage - colab - gitub.

in your notebook code put:

```
from google.colab import drive
drive.mount('/content/drive')
```

Create your **Access token** in github, **Settings/Developer Settings/ Personal access tokens** Generate a new access token activate all repo options in checkbox.

not share the token. You could create a directory in drive named **repos**.

37.1 Create or clone a repo

37.1.1 Create a repo from scratch

```
!git init test_repo
%cd test_repo
!ls -a
```

37.1.2 Create a repo from scratch

go to github and create a repo without documents or license inside,

```
https://github.com/<username>/<repository>
```

replace **username** and **repository** for the really names in official site.

```
git remote add <remote-name> https://{git_token}@github.com/{username}/{repository}.git
```

```
git push -u <remote-name> <branch-name>
```

```
username = 'it-ces'
token = 123
repository = 'test'
!git remote add origin https://{token}@github.com/{username}/{repository}.git
!git remote -v
!git push -u origin master
```

37.1.3 Clone a repo

from an existing repo copy the **http** url.

```
git clone https://{token}@github.com/{username}/{repository}
!cd {repository}
```

therefore we can store the data in drive and update the notebook from colab.