Git

# General information

Used as:

* Persistent map
* Content Tracker
* Revision Control System
* Distribution System

Types of git objects:

* Blobs
* Trees
* Commits
* Annotated Tags

## Other information (not necessary useful):

* Git stores information key-value pair where for each value we have a key coded using SHA1 hash. If you want to see the hash that will be created for a string like “Apple Pie” use : echo “Apple Pie” | git hash-object –stdin
* In order to initialize an empty repository we use : git init and if you want to see the files added use: ls -a then open .git
* To see the type of the information stored under a certain hash: git cat-file 21345AD213(SHA1) –t
* To see the information pretty : git cat-file 21345AD213(SHA1) –p
* For the last two commands we don’t have to write the whole hash just enough letters in order to be able to uniquely identify it

# Working with git - commands

## Steps when you first create a repository:

* git init to create an empty repository
* git status to see the status of your files
* git add menu.txt add a specific file into the staging area or git add . to add all untracked files
* git commit –m “Message like first commit”
* git log to see all logs git log -2 to see the last two commits
* you can add tags(simple labels attached to an object) to git (they are like a branch that doesn’t move):
  + standard : git tag dinner
  + annotated : comes with additional information like a message: git tag –a mytagName –m “My tag message”

## Branches

* git branch -> see all the branches
* git branch develop -> create a branch named develop
* git checkout develop -> switch to the develop branch –all the staged files are still available

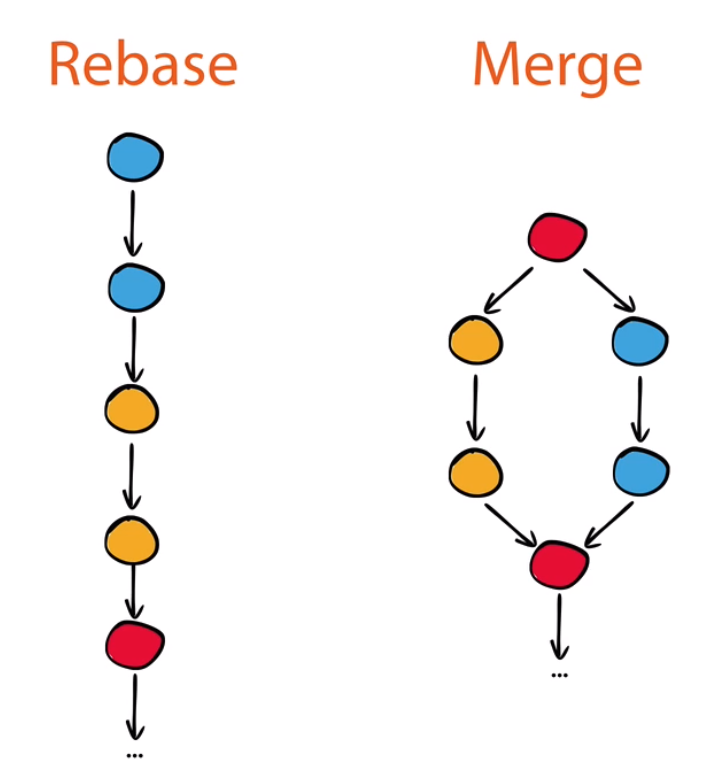
## Merging

* In order to merge two branches you must switch to the branch that you want to remain and use git merge otherBranchName - !!! you may have conflicts that you need to solve before merging
* use git status to see where is the conflict and solve it manually or using special tools after solving use git add and git commit
* you can make commits not attached to any branch by moving the head to point to a commit and not a branch git checkout 2134er32(SHA1 of a commit) but these commits will be deleted by garbage collector unless attributed to a branch
* in order to keep them we can point the head at them and create a new branch there using git checkout noGoodBranch

## Rebasing

* git rebase master -> (we are on the developer branch ) it copies all the commits on the develop branch after the last common commit and adds them over the master branch. The old commits

will be deleted by the garbage collector;

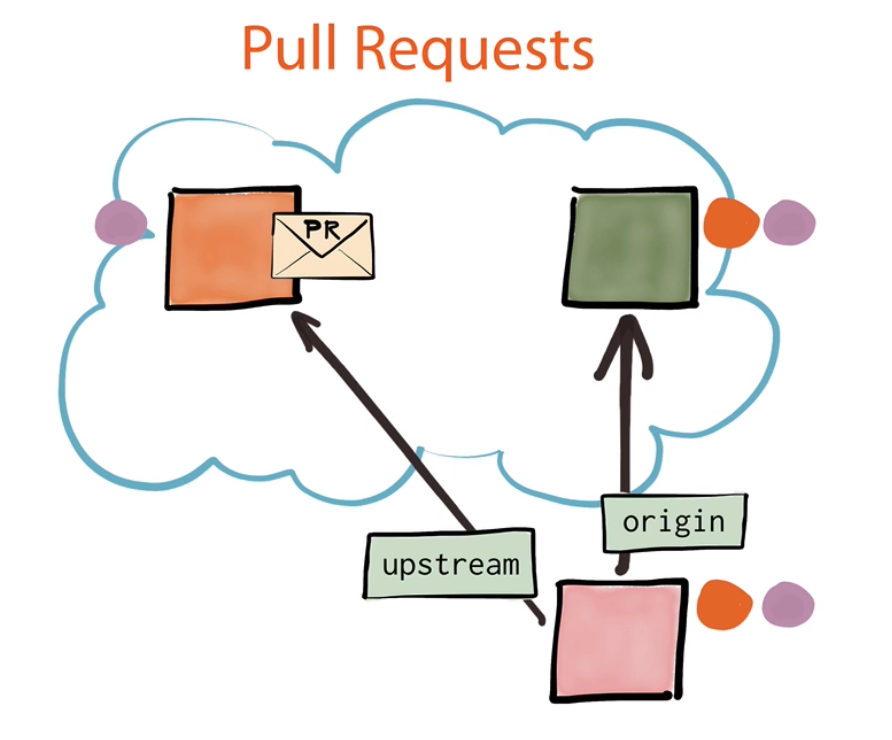


## Commands for version control

* git clone <https://github.com/nusco/cookbook.git> -> clone a repository to your computer (the code files and the git files with the objects and all witch allow us to see and access the project log history)
* git show-ref master to see the branch that is referenced
* git push -> the branches in the origin point to the version that got pushed
* In case of conflicts:
  + git push –f -> to force a push if conflicts exist (the origin changes and will be the same as the local so any additional information that were in the origin will be lost)
  + Solve the conflicts
    - git fetch -> changes the local master to the one that is remote
    - git merge master -> to have the master branch updated with all the information
    - git push -> add the changes to the remote version

# GitHub Features

* Fork -> remote clone a project and then clone the remote project on the local machine. This way there is no connection between our project and the original one.
* Remote upstream -> a remote that points to the original project
* Pull request -> ask the owners of the original project to change it according to your origin



# Mastering Git tutorial

## The four areas: Introduction

Git stores information in:

* Working area: where you keep your current files and folders
* Stash: temporary storage area
* Index or the staging area: where the files must be added before the commit
* Repository: contains the entire history of the project (where the files end up after the commit)

## The four areas: Basic Workflow

* git diff master developer -> see the differences between the two branches
* git rm FILE.txt --f ->forced remove of the FILE.txt both from the working area and the index
* git rm --cached FILE.txt -> removes the file just from the index
* git mv menu.md menu.txt -> changes the menu.md file into menu.txt both in the working area and the index

## The four areas: git reset

With reset we move an entire branch (with that also comes the moving of the head) and optionally copies data from the Repository to the other areas.

* --hard -> copies the files from the new commit to the working area and index. That means that the additional information from both of those will be lost
* --mixed -> copies date from the new current commit to the index (if you don’t specify any arguments to the reset command this is the default)
* --soft -> just moves the branch and doesn’t tough any data

## The four areas: Advanced Tools

## History: Exploring the past

## History: Fixing Mistakes

## Finding your workflow