## Pasos básicos para iniciar Git:

1. Instalar Git y ejecutar git bash
2. Definir el nombre de usuario que nos identificará dentro de git:
   1. Git config –global user.name “user\_name”
   2. Git config –global user.email “email@email.com”
3. We can list our changes:
   1. Git config --list
4. Initialize the git repository
   1. Git init
5. Link our local repository with a remote repository
   1. Git remote add *remote\_branch* <https://github.com/ferguzon/project_refresh>
   2. Configuramos la conexión en caso de estar detrás de un proxy: git config --global http.proxy http://172.16.3.1:8080



In the image:

1. Workspace: where you are actually working with files in your computer
2. Index: tells git where are the files that it should be controlling
3. Local repository: these are the files that are stored or version controlled in your local repository
4. Remote repository: remote server in which you store your files

## Pasos básicos para manejar cambios

**Staging index**

1. Add files to the staging index so git knows it has to monitor those files
   1. Git add . // adds all new files
   2. Git add –u // updates tracking for files that changed names or were deleted
   3. Git add – A // does both of the previous
   4. Git add filename // add the file to control by its name
2. Delete files from the staging index and our computer
   1. Git rm filename
3. Check the status of our working folders
   1. Git status
4. Quitar del staging index un archivo que no queremos enviar al siguiente commit
   1. Git reset HEAD *nombre\_archivo*

**Commits**

1. Commit changes to push them into your local repository
   1. Git commit // opens the text editor to insert a description of the commit
   2. Git commit -m “message”
   3. Título en imperativo con menos de cincuenta caracteres, el resto del mensaje con líneas de menos de 72 caracteres cada una
2. Recuperar archivos de un commit anterior
   1. Git checkout *commit\_number* *file\_name*
3. Examinar los distintos commits que hemos hecho
   1. Git log
   2. Git log –oneline –decorate (show everything in one line and with colors)
   3. Git log –p -2 (shows the changes in that commit –the patch-, only in the last two commits)
   4. Git log --pretty=*short | full | fuller* (show commits with more or less info)
   5. Git log --relative-date (shows a relative date, not the date the commit was made)
   6. Git log --since=*beginning\_date* --until=*end\_date* (limits de range date of the commits. You can use relative dates “3 minutes”)
   7. Git log --after=*beginning\_date* --before=*end\_date* (limits de range date of the commits. You can use relative dates “3 minutes”)
   8. Git log --grep=”*text\_to\_search”* (allows you to search for specific text in the commit message)
   9. Git log --author=*author\_name* (allows you to search for commits made by a specific author)
4. Crear tags (alias a commits concretos)
   1. Git tag *new\_name* (if we are in the commit right now)
   2. Git tag *new\_name­* *number\_of\_commit* (if we are not in the commit right now)
5. Ver la lista de tags
   1. Git tag
6. Regresar a un commit previo
   1. Git checkout *number\_of\_commit*
   2. Git commit –amend (allows us to “amend” the last commit. You can modify a file befor you execute this command and it is going to take it as part of the last commit you made, that is, the commit you are amending)
7. Regresar a la versión anterior de un archivo (la versión que teníamos en el último commit)
   1. Git checkout *commit\_number* *nombre\_archivo*
8. Regresar el repositorio complete a un estado anterior:
   1. Git checkout *commit\_number*
9. Cambiar de commit guardando el estado actual para trabajarlo posteriormente (sin necesidad de hacer el commit):
   1. Git stash

**Push**

1. Push changes to the remote repository
   1. Git push *direction branch\_name* (branch names referes to one of our branches, the one we want to push)
   2. We can use --rebase to make sure our changes are sent

**Files**

1. Mover archivos
   1. Change the location manually in your OS (Git status will detect you deleted the file but added it in other direction)
   2. Execute the command “rm” to the moved file and the command “add” to the file on its new location
      1. Git rm funciones.h
      2. Git add include/funciones.h
   3. Git status will detect the movement

## Branches

1. Creamos nuevas ramas
   1. Git branch *name\_of\_branch*
   2. Git checkout –b *name\_of\_branch* //allows us to create a branch and move to it inmediatelly
2. Check all branches in our Project
   1. Git branch
   2. Git branch --all
3. Delete a branch
   1. Git branch –d *name\_of\_branch*
4. Fusionar ramas
   1. Tenemos que estar ubicados en la rama de destino (p.ej. master)
   2. Git merge ­*rama\_de\_origen* (rama de origen se refiere a la rama desde la cual vienen los cambios)

## Commits

## Pasos básicos para administrar repositorios

1. Clonar un repositorio
   1. Git clone https://github.com/ferguzon/datasciencecoursera

## Otros trucos

1. Cambiar el editor de texto de Git
   1. Entrar al archivo config en la carpeta de .git y agregar:

[core]

editor = 'E:/Descargas/Programas portables/Notepad++Portable/Notepad++Portable.exe'

## Avanzado

1. Examinar los distintos commits que hemos hecho. Podemos usar el parámetro –pretty para indicar a git el formato en el que queremos que muestre nuestros commits
   1. Git log --pretty=format:” "%h - %an : %ae" (muestra el número hash corto del commit, nombre del autor y su dirección de correo. Existen muchos más parámetros para este comando
   2. git log --pretty=format:"%h %s" –graph (añade una gráfica que muestra la evolución de nuestras ramas)
   3. Git log –stat (show statistics for files modified in each commit)