**3- Pair programming y buenas practicas de la programacion**

·Pair programming(programación conjunta o de pares):

una persona escribe mientras las otras aportan ideas

Ventajas:

-te trabas menos

-tenes menos errores

-practicas explicar las ideas(complejas)

-aprendes de tus compañeros

-Ir rotando el que esta enfrente del teclado, cada vez que hacemos un commit, tratemos de cambiar quien esta en la computadora

-Es mejor para las partes difíciles

·Buenas practicas:

- entorno de trabajo: automatizar las tareas que hacemos siempre(makefile)

-encontrar el editor: recomendable tratar de aprender a usar al menos uno sin interfaz grafica como vim o nano (gnu nano) ademas de visual: nos permitiría trabajar colaborativamente como si estuvieramos trabajando en un ???(emance, sublime: para poca memoria) porque cuando nos conectemos a un servidor externo no vamos a tener interfaz grafica. Configurar editor, tamaño de texto, identacion automatica, tamaño de linea en menos de 80 lineas, shortcuts: atajos

-desarrollo incremental:

1- primer commit con el código inicial LIMPIO(bajar el esqueleto y hacer un commit agregando los archivos del esqueleto tal y cual como estan

2- Implementen una función y fijense si compila

funcion→ compilen→ ejecuten

3- commits pequeños: uno por función( para asi si pasa algo poder volver atrás)

4- no commitear sin probar que compile

No todo es correctitud: tmb tiene que ver con que el codigo sea legible claro e intuitivo

Principio KISS- Keep it simple:

Estilo de código:

1. Identacion

2. Tamaño de lineas

3. Bloques

Legibilidad:

4. Nombres significativos(aprovechen autocompletar)

5. Funciones pequeñas

6. Eviten mas de 4 o 5 niveles de anidación

7. Estructuras de código simple

**1-Introduccion a git:**

Colaboración en linea y entornos de trabajo:

Hace control de versiones, se encarga de mezclar el codigo de distintas personas en un mismo repositorio

Guarda el historial de cambios para poder volver atras(punto donde sepamos que funcionaba bien el codigo

Es distribuido: cada uno va a tener una copia que va a ir sincronizando

Bitbucket-servidor en la nube:

(No es lo mismo que git: protocolo de comunicación entre los distintos desarrolladores, distintas copias del desarrollo)

Evita conectarse computadora a computadora sino que nosotros nos conectamos al servidor se encarga de hacer de punto de conexión y nos provee una copia segura.

Otra alternativa sería GitHub(ahora microsoft) o gitlab, pero es mas comod p instituciones educativas

-Servicio de almacenamiento(y otras cosas) provisto por una empresa

-Sirve para mantener una copia en la nube que todos pueden acceder

-Por comodidad, todos los cambios se hacen a través de este servidor

**Comandos:**

**0. Git Clone:** con esto descargamos el repositorio

1. Instalar git

2. Ingresar al repo de su grupo desde el navegador. Copiar el comando donde dice CLONE, usar el protocolo HTTPS(no SSH)

usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]

[--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]

[-p | --paginate | --no-pager] [--no-replace-objects] [--bare]

[--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]

<command> [<args>]

Estos son comandos comunes de Git usados en varias situaciones:

comienza un área de trabajo (ver también: git help tutorial)

clone Clona un repositorio dentro de un nuevo directorio

init Crea un repositorio de Git vacío o reinicia el que ya existe

trabaja en los cambios actuales (ver también: git help everyday)

add Agrega contenido de carpetas al índice

mv Mueve o cambia el nombre a archivos, directorios o enlaces simbólicos

reset Reinicia el HEAD actual a un estado especifico

rm Borra archivos del árbol de trabajo y del índice

examina el historial y el estado (ver también: git help revisions)

bisect Use la búsqueda binaria para encontrar el commit que introdujo el bug

grep Imprime las líneas que concuerdan con el patron

log Muestra los logs de los commits

show Muestra varios tipos de objetos

status Muestra el estado del árbol de trabajo

crece, marca y ajusta tu historial común

branch Lista, crea, o borra ramas

checkout Cambia ramas o restaura los archivos de tu árbol de trabajo

commit Graba los cambios en tu repositorio

diff Muestra los cambios entre commits, commit y árbol de trabajo, etc

merge Junta dos o mas historiales de desarrollo juntos

rebase Vuelve a aplicar commits en la punta de otra rama

tag Crea, lista, borra o verifica un tag de objeto firmado con GPG

colabora (mira también: git help workflows)

fetch Descarga objetos y referencias de otro repositorio

pull Realiza un fetch e integra con otro repositorio o rama local

push Actualiza referencias remotas junto con sus objetos asociados

'git help -a' y 'git help -g' listan los subcomandos disponibles y algunas

guías de concepto. Consulte 'git help <command>' o 'git help <concepto>'

para leer sobre un subcomando o concepto específico.

En bitbucket:

clone→ da el comando entero para generar la conexión entre el repositorio local y el repo en git, selecccionar todo el comando y ejecutarlo en la consola

$git clone htps://user@bitbucket.org/sistop-famaf/so20lab1gxx.git

$cd so20lab1gxx.g

$ls -a ///p ver lo que tenemos

-En linux todos los archivos que empiezan con un punto son archivos ocultos

3. Usar el comando para copiar o clonar el repositorio desde bitbucket a su maquina y trabajar localmente

**1. GIT STATUS:**

1. Modificamos un archivo que ya existe

2. creamos un archivo de prueba, que estara en nuestro espacio de trabajo

3. Con git status -s vemos que esta modificado y que esta fuera de nuestro repositorio

$ echo “\*.o” >> .gitignore ///agregamos una linea nueva, gitignore tiene patrones de archivos que

////no se deberian agregar nunca a un repositorio en este caso los .o(archivos compilados de c)

//con el comando le digo a gitignore que ignore todos los archivos que terminen en .o

$touch NOMBRE.txt //crea un archivo nvo

$git status -s //con -s dice el archivo del repositorio

M .gitignore → modificado

?? NOMBRE.txt→ archivo nuevo, no tiene estados anteriores

**2. GIT ADD:**

1. Para poder sincronizar los cambios que hicimos, tenemos que agregarlos al repo en dos pasos:

2. Con git add [archivos] agregamos los cambios al area de staging

3. Con git status -s vemos tambien los archivos staging

$ git add .gitignore Nombre.txt //no agregar ejecutables, .o, fijarnos que archivos agregamos

$git status -s //consultamos el status

M .gitignore

A Nombre.txt //adicion, se agrego un archivos

**3. GIT COMMIT**

1. Tenemos que preparar un paquete de cambios con todas las modificaciones, llamado commit(paquete de cambios ya cerrado- comitear cuando agrupamos todos los archivos, cuando hacemos el comit es que el paquete se cierra)

2. Con git commit -m “Descripcion del commit” formamos el commit con todos los archivos del area de staging y lo agregamos al repositorio

3. Comprobamos que staging esta vacio

$git commit -m “Primer commit de prueba” //Todo commit debe tener una descripcion, si no //ponemos -m va a abrir un editor de texto

$git status -s // cuando pregutamos nos va a mostrar que no tenemos nada en staging despues de commitear

La primera vez que lo usemos puede saltar error porque falte configuración:

$git config –global user.name “Estudiante A”

$git config –global user.email [estudiante@mi.unc.edu.ar](mailto:estudiante@mi.unc.edu.ar)

4. Git Push

1.Para enviar los cambios a bitbucket, primero debemos volver a sincronizar el repositorio local con git pull (digitalizamos los archivos en nuestra copia local pero nunca los mandamos a bitbucket→ el repo que esta en la nube es distinto del repo que tenemos nosotros)

2. con git push suben todos los cambios en su repositorio local a Bitbucket. Solo la primera vez que hacen un push en un repositorio, deben agregar -u origin master

$git pull

$git push -u origin master //la primera vez que lo hacemos las siguientes con git push ya esta

1ro- traer la copia del repositorio en la nube a nuestro repositorio local porque puede haber cambios nuevos de nuestro colaboradores y nosotros tenemos que agregar los cambios encima de los cambios que hayan hecho

Entonces la primera vez que trajimos los cambios del repositorio de bitbucket a nuestro repositorio local lo hicimos con git clone, pero eso es solo la primera vez es como que estamos copiando el repositorio entero; las veces siguientes lo tenemos que hacer con el comando git pull eso es lo que trae los cambios desde el repositorio externo a nuestro repositorio local

---podemos hacer muchos adds antes de hacer un commit y tmb podemos hacer muchos commits antes de hacer un push

**5. Conflictos**

1. Si dos personas modifican la misma linea en el mismo archivo, git les dira que hay un conflicto al hacer pull. Para solucionarlo:

a. En cada archivo con conflicto, desde un editor de texto, buscar los marcadores de conflicto y modificar el archivo hasta que sea correcto

b. agregar los archivos con **git add**

c. Hacer un nuevo commit

comando basicos:

clone

add

commit

push