KIBERNUM
IT Academy

Control de Versiones con Git

Contenidos

- Instalaciones necesarias.
- ¿Qué es Git?
- Git como Máquina del tiempo.
- Configuración de Git.
- Comandos más usados y uso de Ramas.
- Versionamiento semántico.
- ¿Qué es Github?
- Configuración Token y SSH.
- Git pull request, fork.

Objetivos

Objetivo General

El alumno logra utilizar los comandos basicos de Git y crear un repositorio en GitHub.

Objetivos Específicos

- Configurar ambiente de trabajo para Git y GitHub.
- Entender Modelo de Datos de Git.
- Crear cuenta en GitHub y Tunel SSH.
- Comprender comandos básicos para explorar un repositorio.
- Lograr hacer un commit y entender las tres fases de un commit.
- Lograr clonar un repositorio y hacer commits de forma colaborativa.

KIBERNUM

Instalaciones Necesarias

Contenido Parte 1

KIBERNUM

Instalaciones necesarias

https://git-scm.com/

https://code.visualstudio.com/

¿Qué es Git?

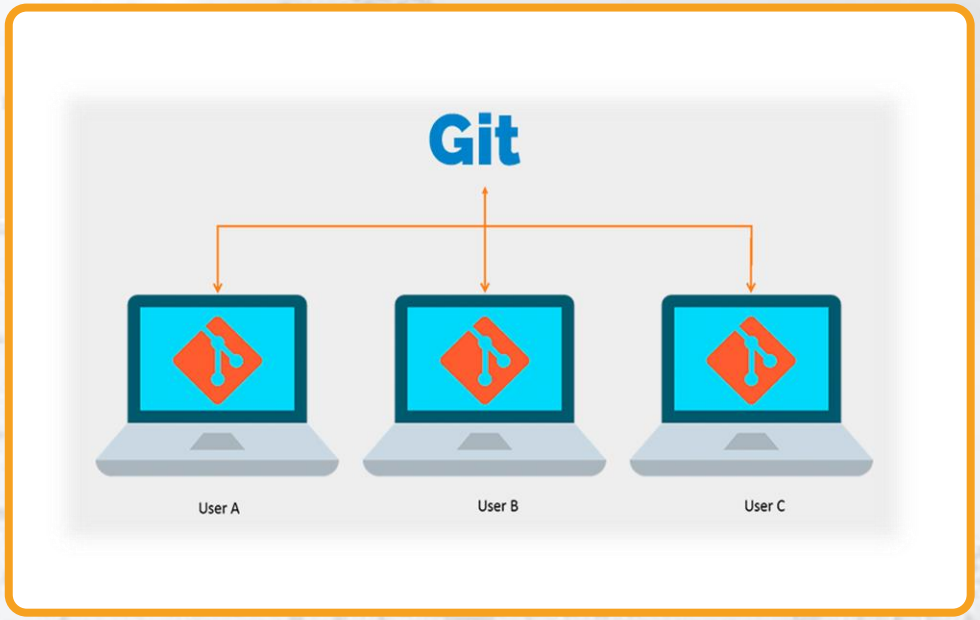
Contenido Parte 2

KIBERNUM

¿Qué es Git?

- Git es un sistema de control de versiones distribuido y de código abierto creado por Linus Torvalds, es una herramienta necesaria para los desarrolladores, que formen parte de equipos de trabajo.

- Con Git podremos tener una secuencia de estado de nuestro código y hacer viajes en el tiempo, que nos permitan corregir errores, recuperar el estado anterior del código y realizar una unión de nuestro código fuente con el código de otros desarrolladores que formen parte del equipo.



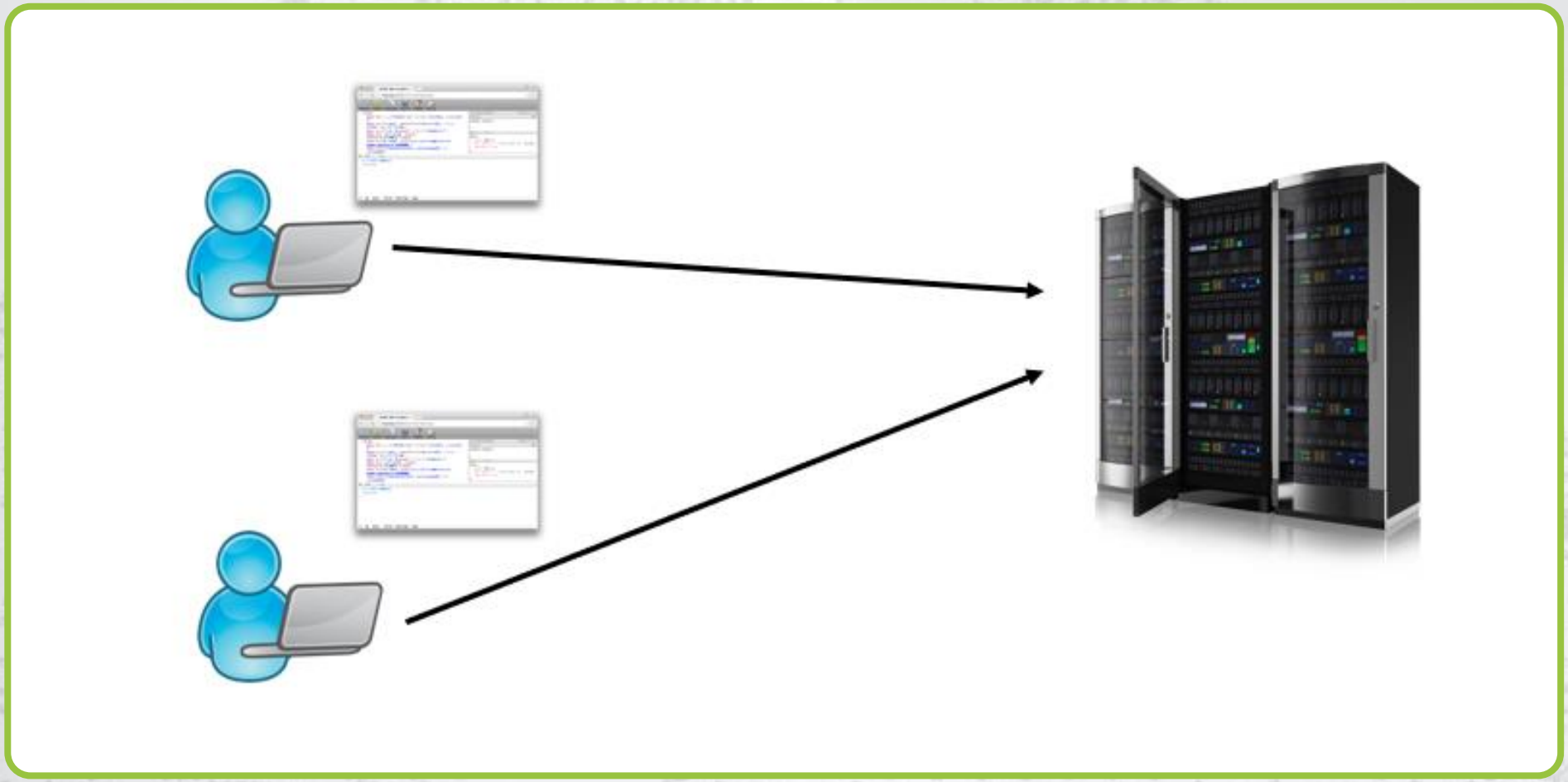
KIBERNUM

¿Para qué usar Git?

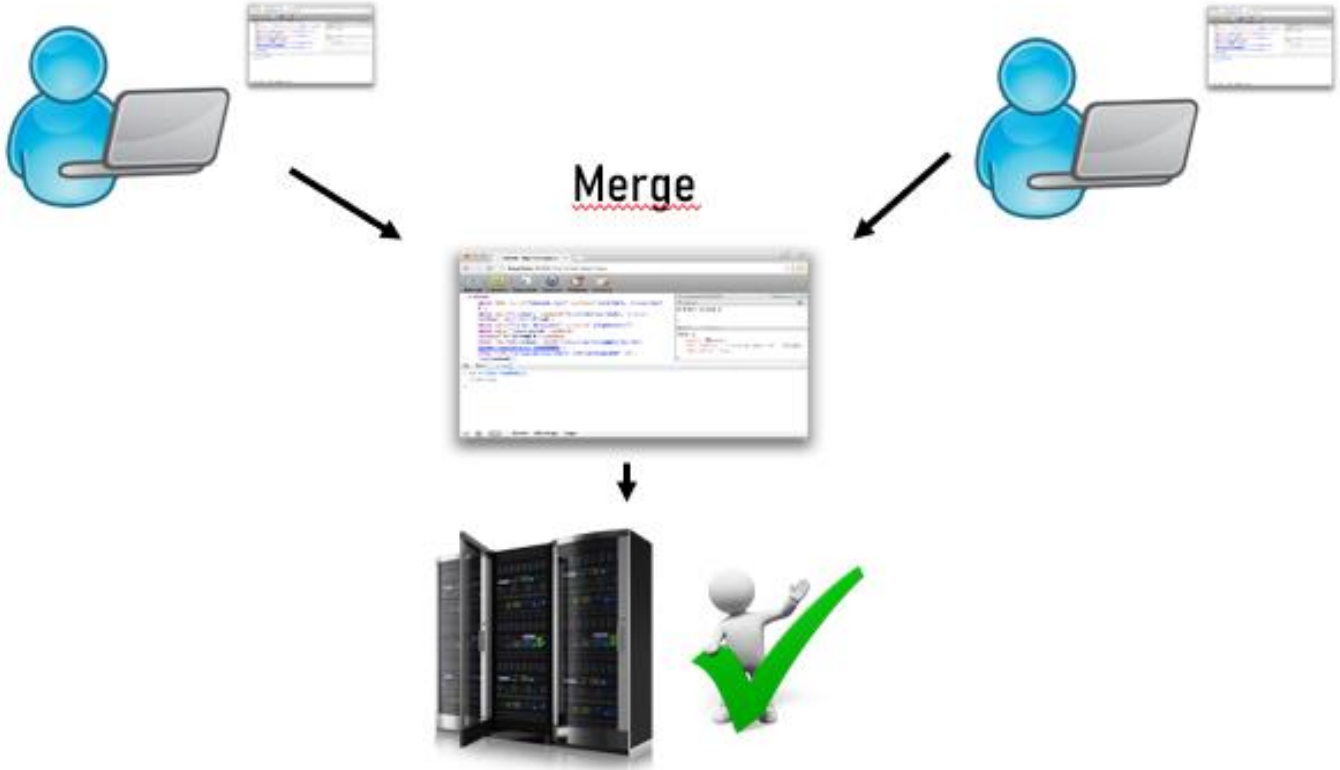
- Permite regresar a versiones anteriores de forma sencilla y muy rápida.
- Facilita el trabajo colaborativo.
- Permite respaldar tus proyectos en la nube (ej: con GitHub).
- Reduce considerablemente los tiempos de deploy.
- Las "branches" o ramas, permiten trabajar con una base de código paralela al proyecto en sí.

KIBERNUM

Un sistema de control de versiones



Un sistema de control de versiones



Git como Máquina del Tiempo

Contenido Parte 3

KIBERNUM

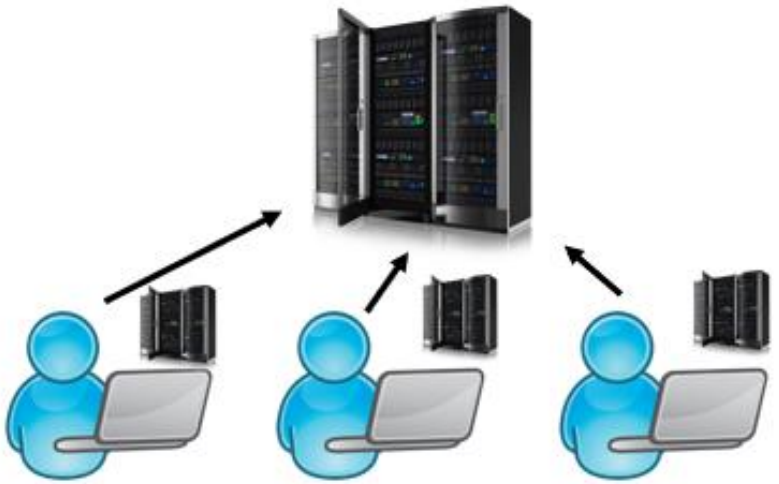
Nuestro proyecto como máquina del tiempo

Git nos a permitir viajar a cualquier punto de la historia de nuestro proyecto

Nuestro proyecto como máquina del tiempo

Repositorio Central

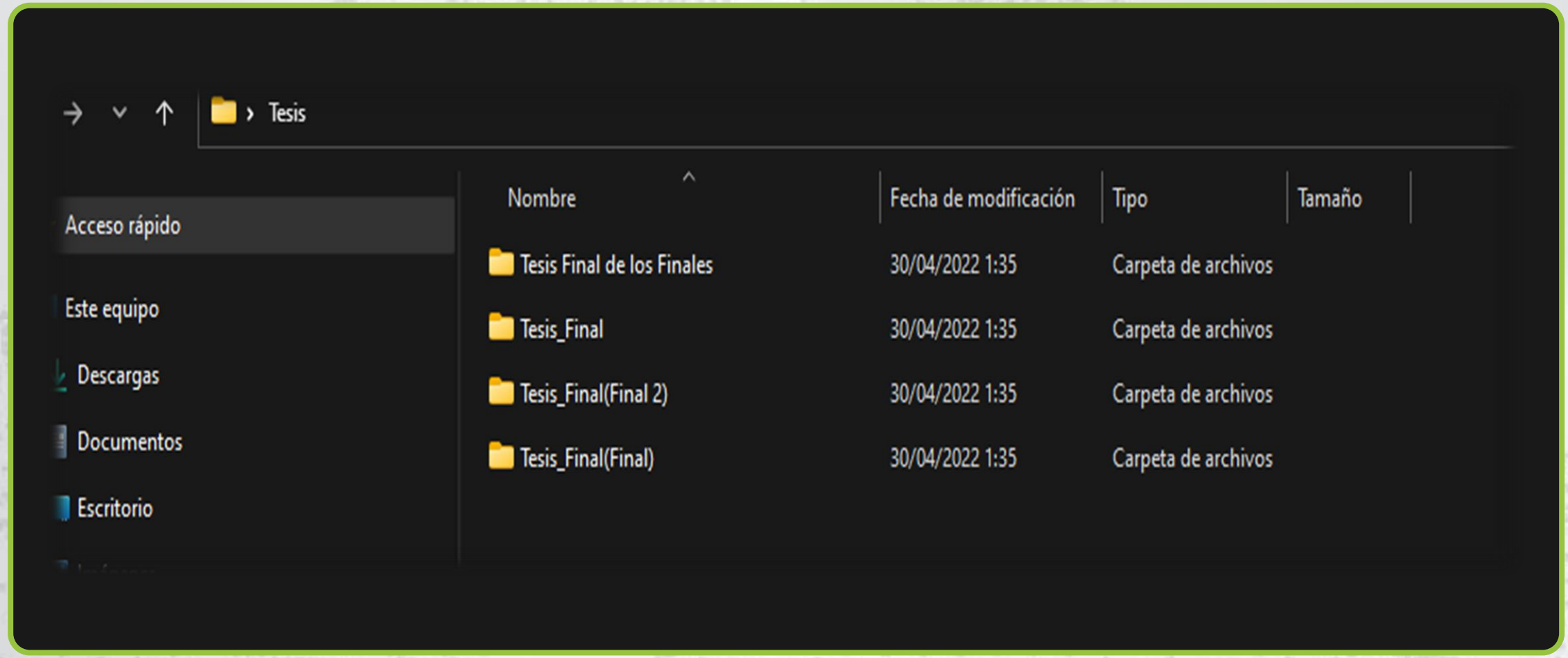
Repositorio Distribuido



Git trabaja como un repositorio distribuido

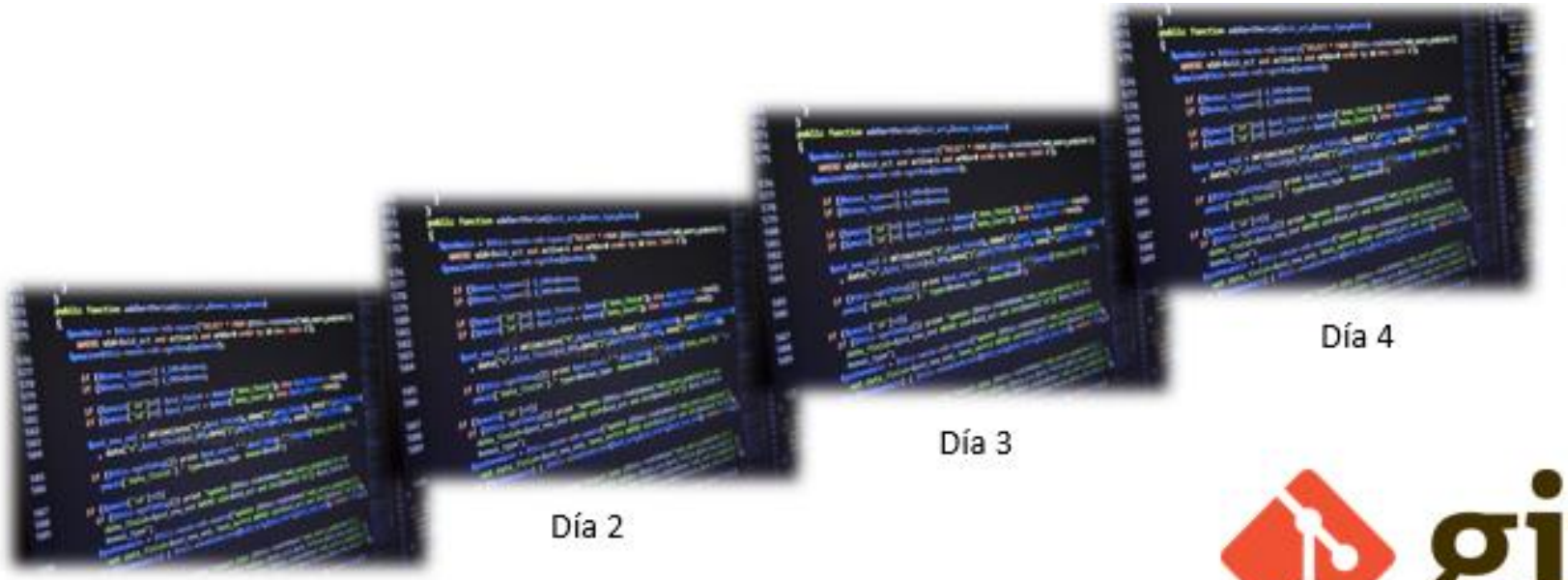
git

¿A quién no le ha pasado?



¿Cómo funciona Git?

Ya sabemos que Git es como una máquina del tiempo, donde podemos viajar a lo largo de la historia. Cuando nosotros guardamos un estado en nuestro proyecto (repositorio) lo hacemos por medio de los commit, una analogía de los commit son las fotografías (snapshot).



git

Configuración de Git

Contenido Parte 4

KIBERNUM

Configuración global

- Antes de comenzar a trabajar con Git es necesario realizar una configuración global del usuario en la máquina que tiene instalado Git, para esto vamos a necesitar ingresar en una terminal los siguientes comandos:

  git config --global user.name "John Doe"
  git config --global user.email johndoe@example.com

- Para confirmar la configuración escribimos:

  git config --global –e (se nos abrirá un editor, para salir “shift + tecla ‘:’ “ escriba q!)

KIBERNUM

Comandos más Usados y Uso de Ramas

Contenido Parte 5

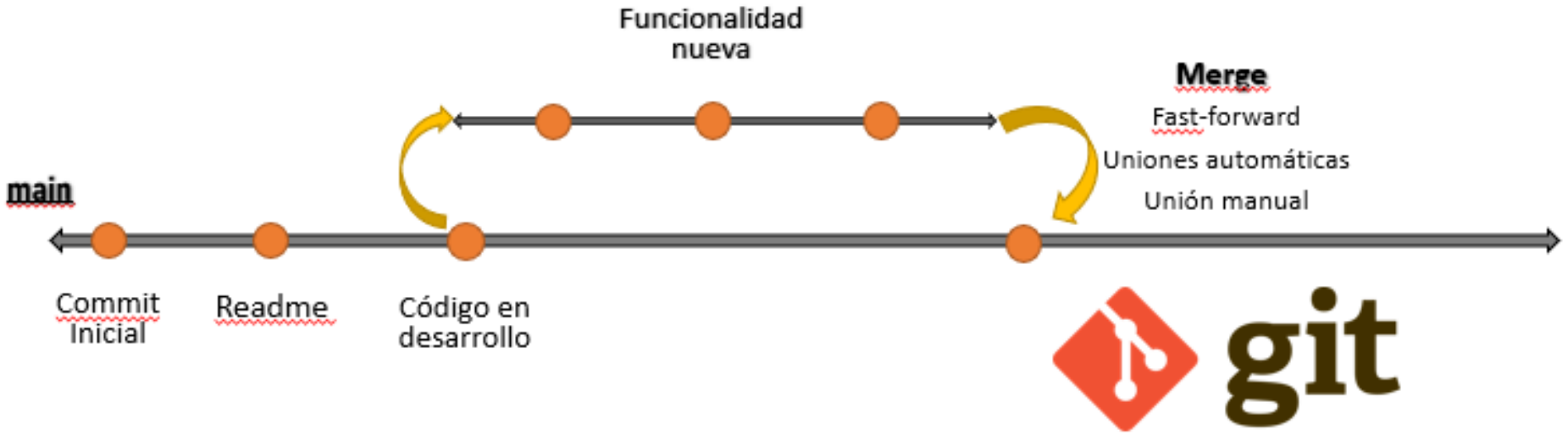
KIBERNUM

Comandos usados

- **git init** (crea el repositorio local y la carpeta .git)
- **git add index.html** (añade al stage)
- **git commit –m “mensaje significativo”** (saca un snapshot crea un registro histórico)
- **git status** (que archivos han sido modificados o creados y los que están listos para el snapshot)
- **git log** (nos permite ver el registro histórico de nuestros commit)

KIBERNUM

Ramas en Git

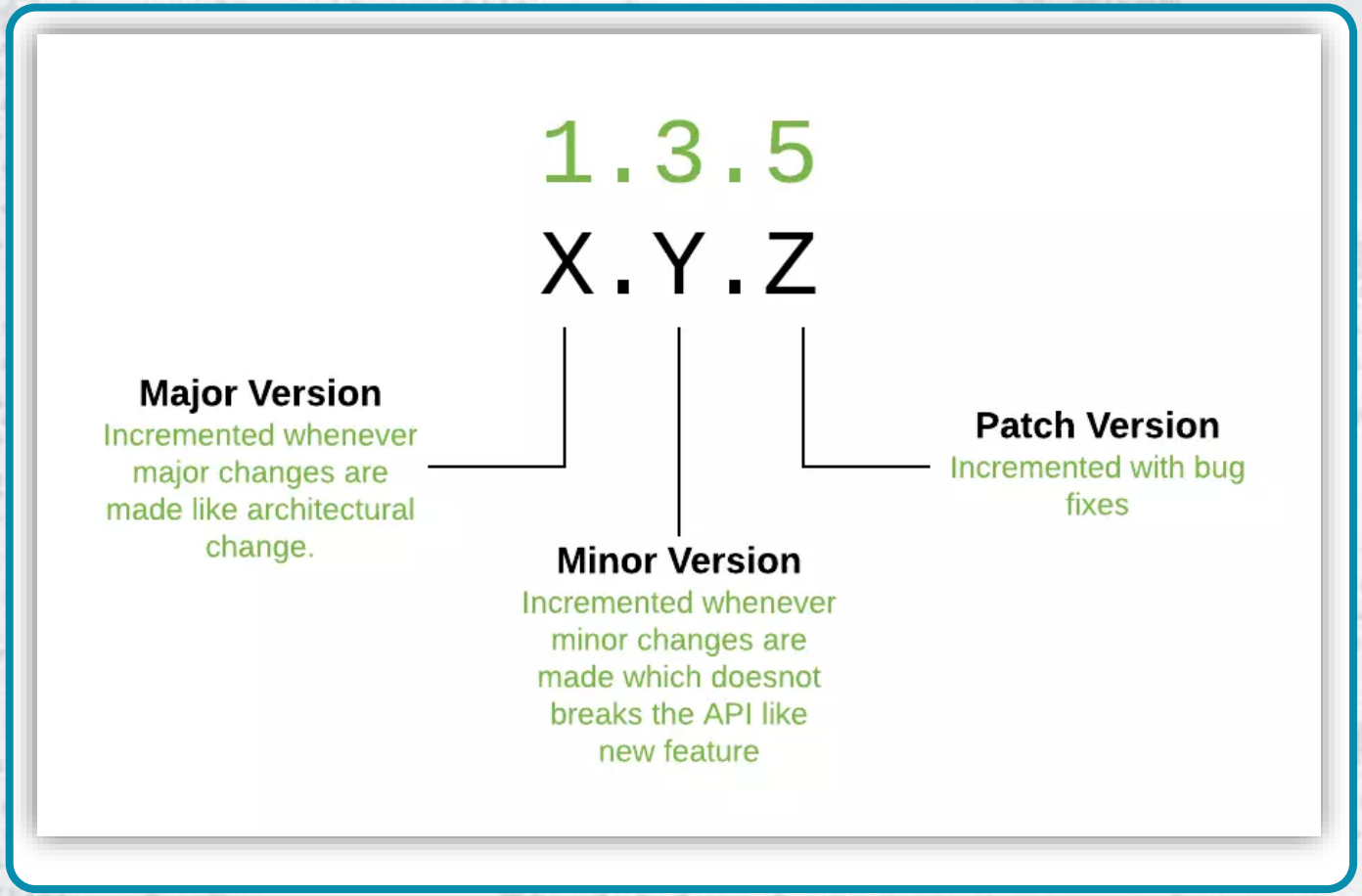

Funcionalidad nueva
Merge
Fast-forward
Uniones automáticas
Unión manual
main
Commit Inicial
Readme
Código en desarrollo
git

Versionado Semántico

Contenido Parte 6

KIBERNUM

Versionado semántico



¿Qué es GitHub?

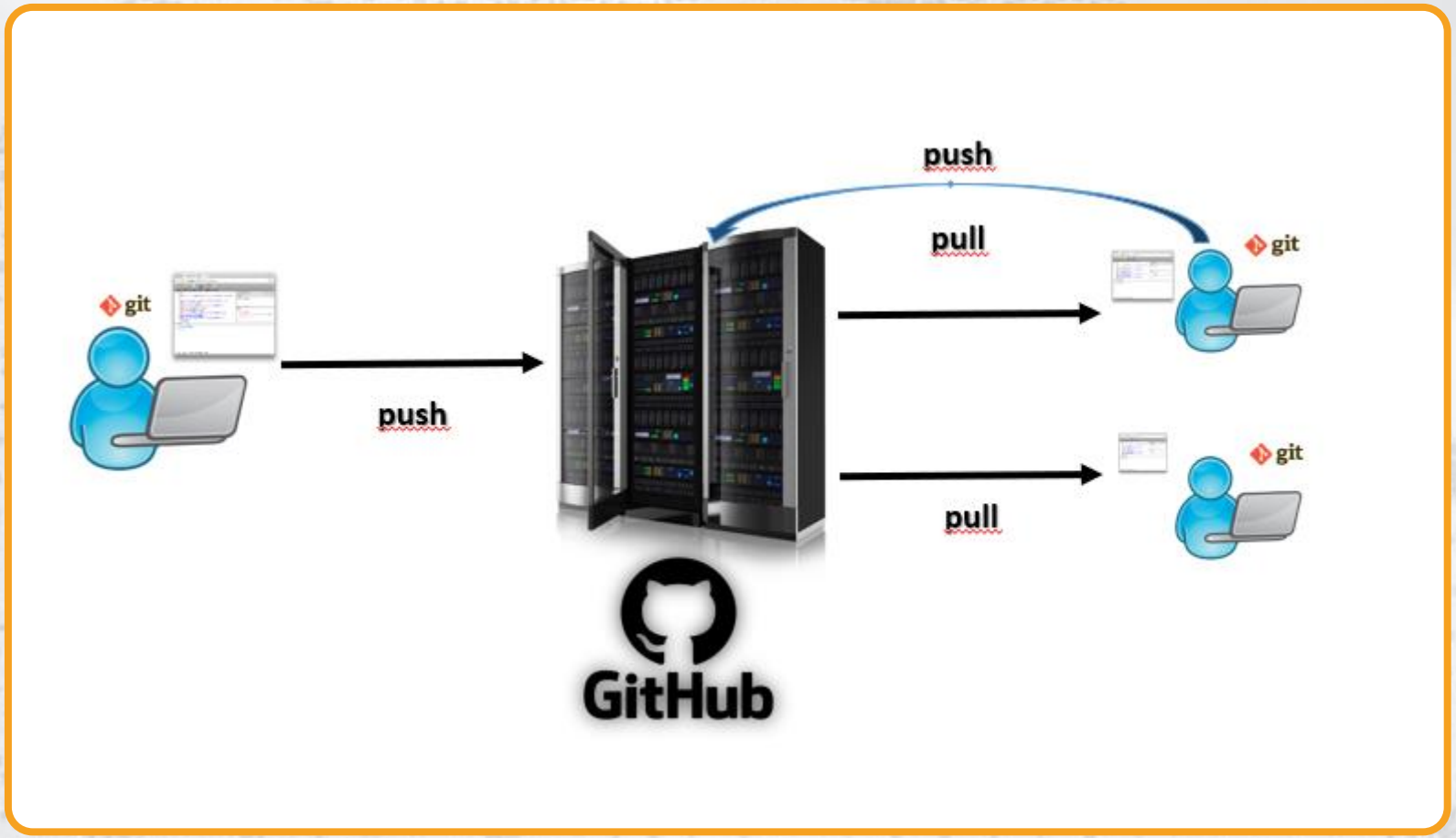
Contenido Parte 7

KIBERNUM

¿Qué es GitHub?

Es una plataforma propiedad de Microsoft, donde podemos alojar nuestros proyectos en repositorios, usando el sistema de control de versiones GIT.

Repositorio remoto



Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/MiguelAngelRamos/curso-git.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# curso-git" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/MiguelAngelRamos/curso-git.git
git push -u origin main
```

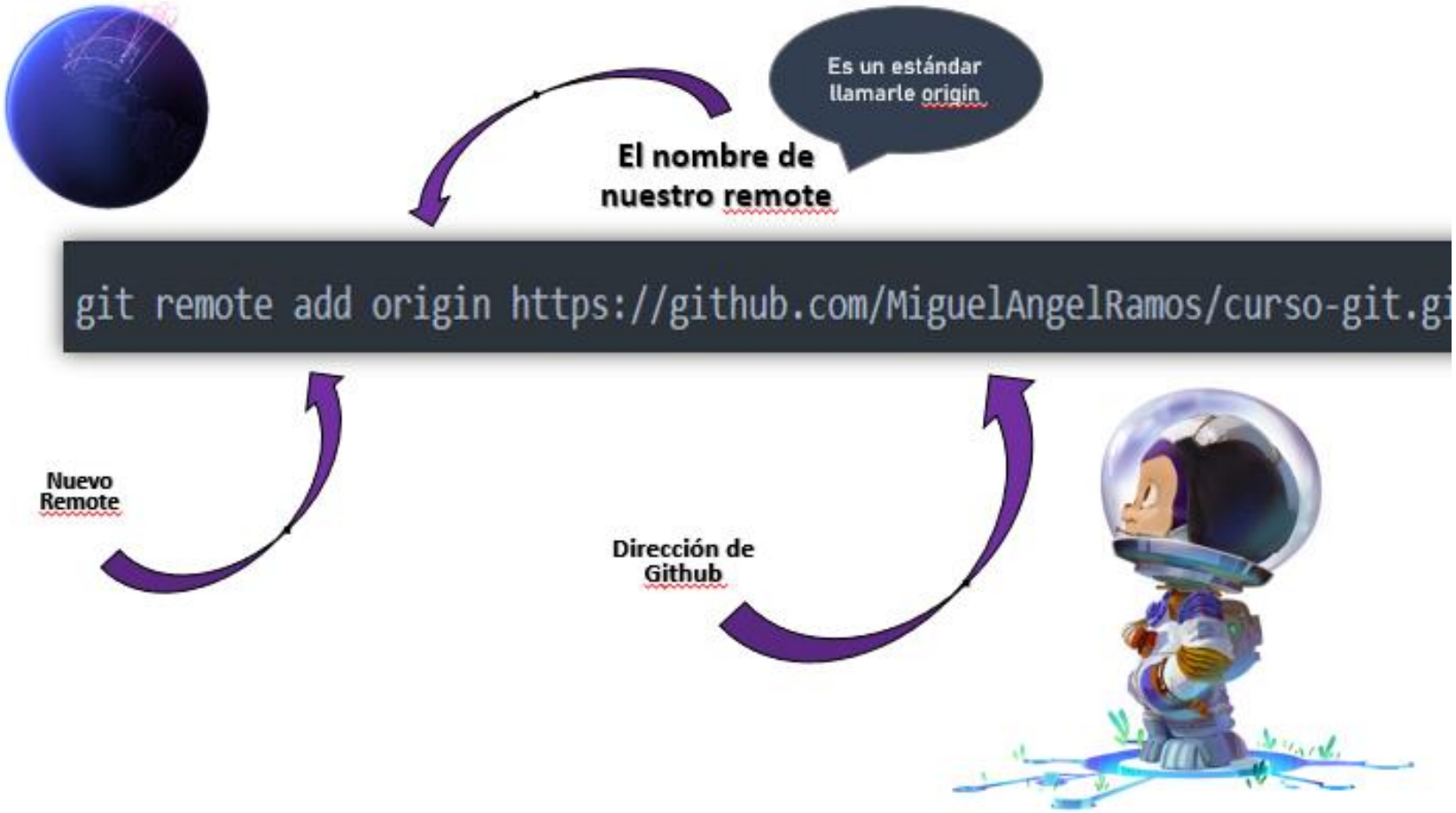
...or push an existing repository from the command line

```
git remote add origin https://github.com/MiguelAngelRamos/curso-git.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code


Es un estándar llamarle origin
El nombre de nuestro remote
git remote add origin https://github.com/MiguelAngelRamos/curso-git.gi
Nuevo Remote
Dirección de Github

Git Remote -V

Cuando queremos revisar las fuentes remotas de nuestro proyecto usar el siguiente comando:

```
PS C:\Users\MIGUEL\Desktop\discord_git\Curso_git\04-mardown> git remote -v
origin  https://github.com/MiguelAngelRamos/markdownexample.git (fetch)
origin  https://github.com/MiguelAngelRamos/markdownexample.git (push)
```

El nombre de nuestro repositorio remoto

```
git push -u origin main
```

La próxima vez que realicemos push, no será necesario especificar la rama

Rama que deseamos enviar nuestro repositorio

Comandos comunes usados de Git en GitHub

- git init : crea un nuevo repositorio local de git.
- git clone : clona un repositorio.
- git add : comando para agregar archivos al área de preparación.
- git commit : guarda todos los cambios hechos en el área de preparación, junto una breve descripción del usuario
- git config : para configurar el usuario y el email
- git status : para ver la lista de archivos que han sufrido cambios y los archivos que ya están preparados (confirmados).
- git remote : nos permite ver todos los repositorios remotos
- git checkout: nos permite crear ramas y navegar entre ellas
- git pull : comando para fusionar los cambios que están en el repositorio local.


```
git config --global user.name "su nombre"
git config --global user.email  sucorreo@gmail.com

git config --list
```

KIBERNUM

Configuración Token y SSH

Contenido Parte 9

KIBERNUM

Crear un Token

Desde la cuenta de GitHub vamos a:

- Settings => Developer Settings => Personal Access Token => Generate New Token (Give your password) => Fillup the form => click Generate token => Copy the generated Token.
- Se vera algo como: ghp_sFhFsSHhTzMDreGRLjmks4Tzuzgthdvfsrta

Para el Sistema operativo Windows.

- Go to Credential Manager from Control Panel => Windows Credentials => find git:https://github.com => Edit => On Password replace with with your GitHub Personal Access Token => You are Done

KIBERNUM

Configurar llaves SSH en Git y GitHub

¿Qué es la criptografía asimétrica?

Cuando enviamos datos por internet, corremos el riesgo de que intercepten nuestra información y nos roben nuestros datos antes de llegar al receptor.
La criptografía asimétrica nos permite enviar y recibir datos de forma segura, de manera que, si nuestra información es interceptada, el usuario mal intencionado no podrá leerla.

¿ Cómo funciona ?

Supongamos que el “Usuario A” desea enviarle un mensaje al “Usuario B”, ambos cuentan con una llave pública y una privada que están ligadas con un algoritmo matemático, “Usuario B”, le da al “Usuario A” su llave pública y conserva la llave privada. El “Usuario A” manda un mensaje cifrado con la llave pública que le proporcionó el “Usuario B”.

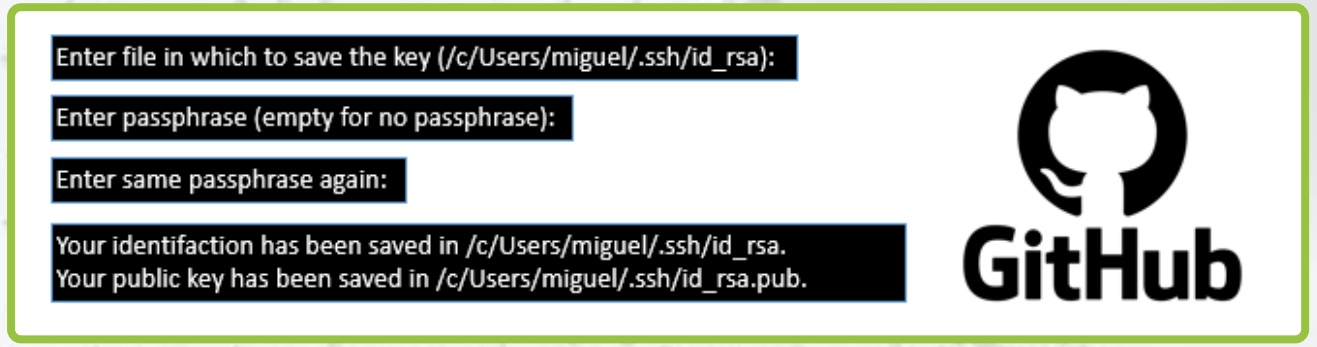
El “Usuario B” recibe el mensaje del “Usuario A”, se inicia el proceso de desencriptar el mensaje con su llave privada de esta manera el “Usuario B” podrá ver el mensaje de forma exclusiva que le envió el “Usuario A”.

¿Cómo funciona la clave privada?

En el contexto de GitHub

Cuando realizamos un push o un pull a GitHub, no estamos exentos a estos tipos de ataques, por este motivo debemos tener una conexión segura entre las dos partes.

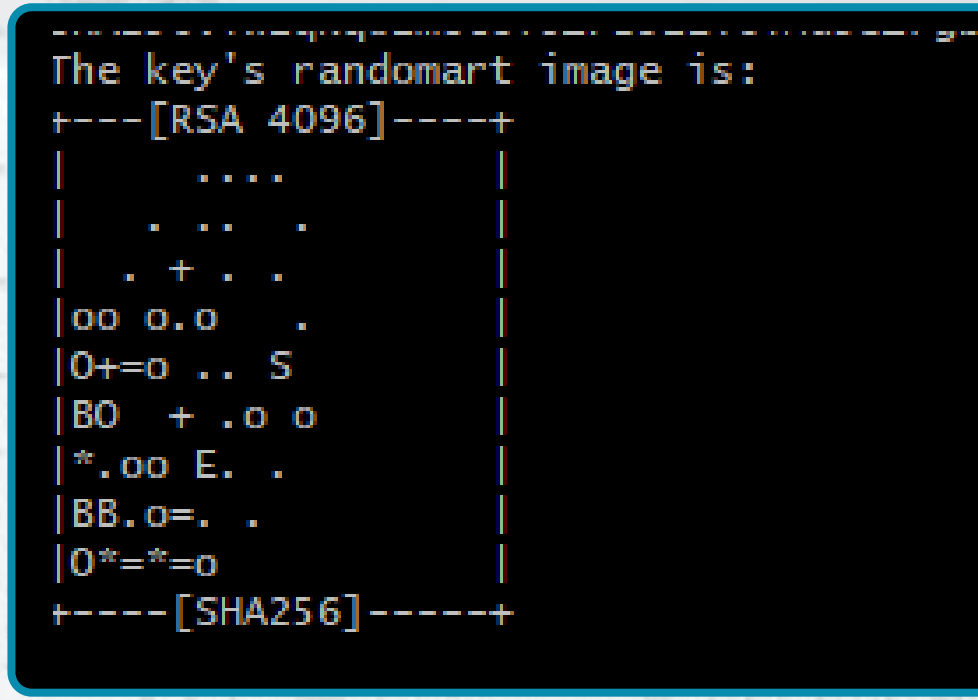
Utilizaremos el algoritmo de encriptación RSA.
ssh-keygen -t rsa -b 4096 -C “tu_email@gmail.com”



En el contexto de GitHub

- Una vez obtenido el resultado sabremos que hemos creado nuestras llaves (pública y privada) SSH satisfactoriamente:
- Para comprobarlo escribimos el comando:
  ls -al ~/.ssh
- Evaluar el servidor ssh ver si está funcionando
  eval $(ssh-agent -s)
- Agregar la llave (privada al servidor ssh)
  ssh-add ~/.ssh/id_rsa
- Copiar tu llave pública puedes usar editor vi
  vi ~/.ssh/id_rsa.pub

En GitHub añade tu llave pública
Ruta: GitHub → settings → SSH and GPG Keys → New SSH Key

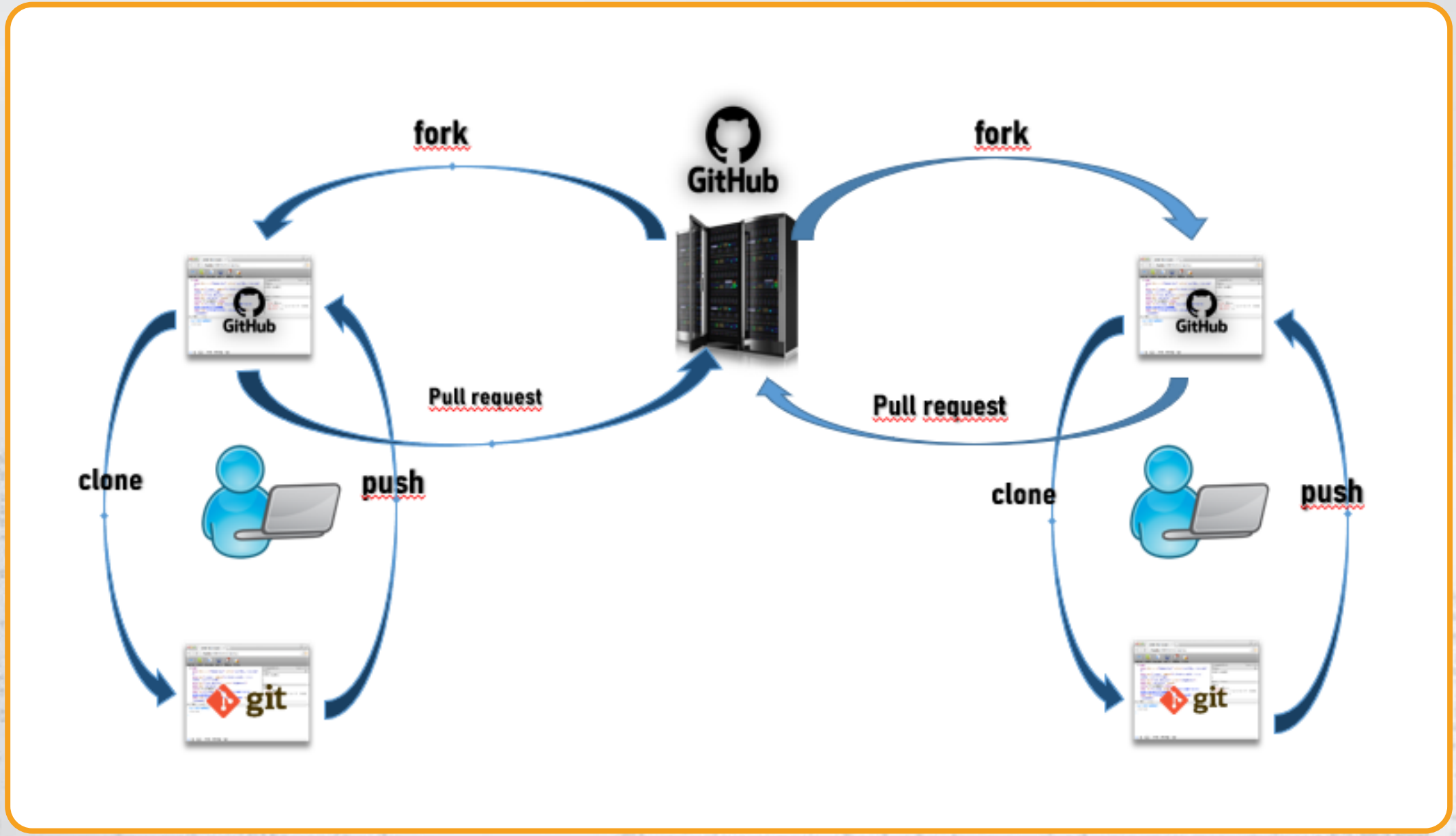


KIBERNUM

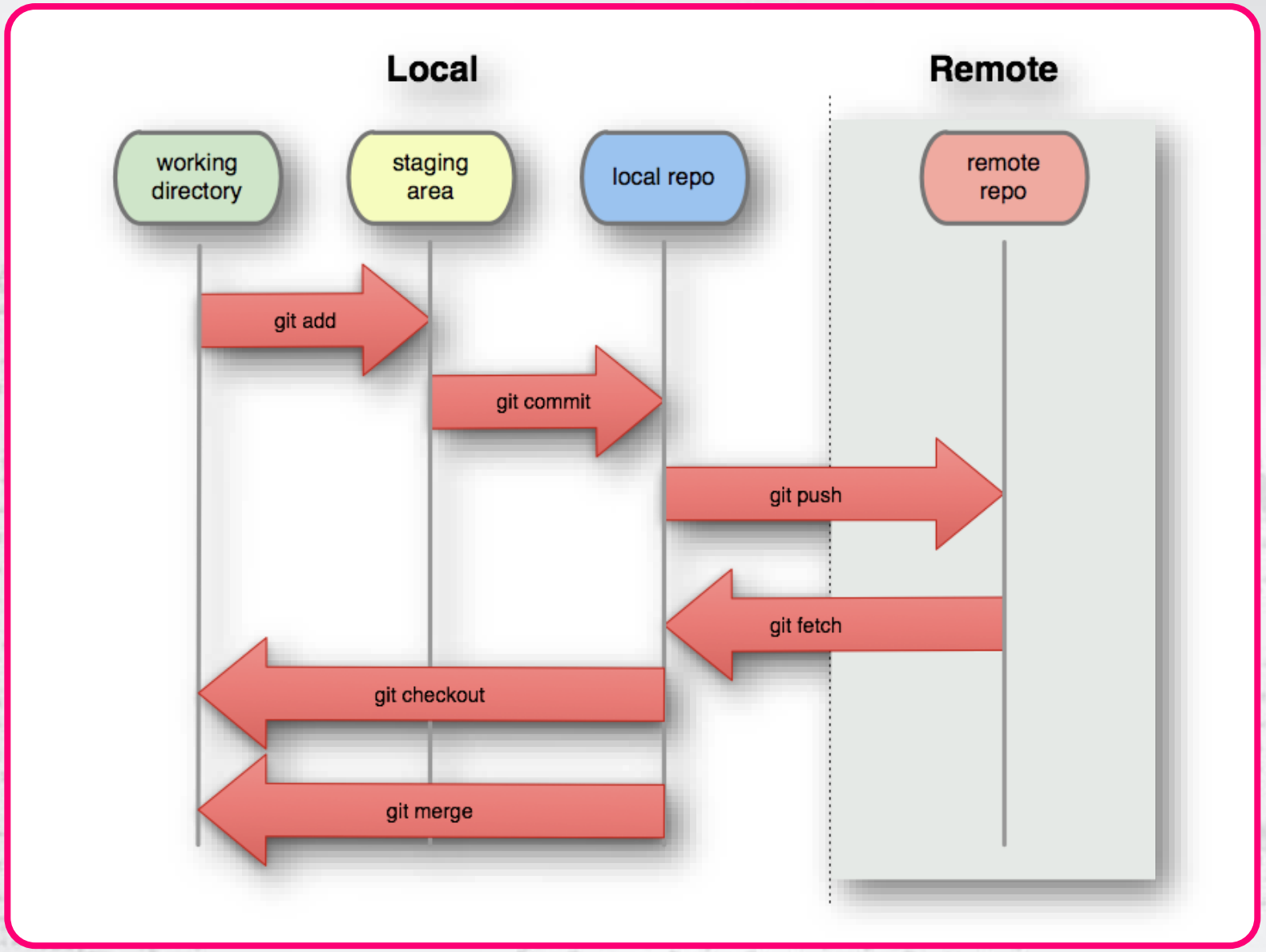
Git pull request, fork

Contenido Parte 8

KIBERNUM


fork
GitHub
fork
GitHub
GitHub
Pull request
Pull request
clone
push
clone
push
git
git

Flujo de Git


Local
Remote
working directory
staging area
local repo
remote repo
git add
git commit
git push
git fetch
git checkout
git merge

Actividad

Conociendo los Sistemas de Control de Versiones:

Permite introducir al tema de Git y GitHub y que se investigue la parte teórica para comenzar la siguiente clase con la parte práctica.

KIBERNUM

Objetivos y Conclusiones

- ¿El objetivo de la Clase se cumplió?
- ¿A qué conclusiones podemos llegar?

KIBERNUM

¿Preguntas?

Muchas Gracias

KIBERNUM