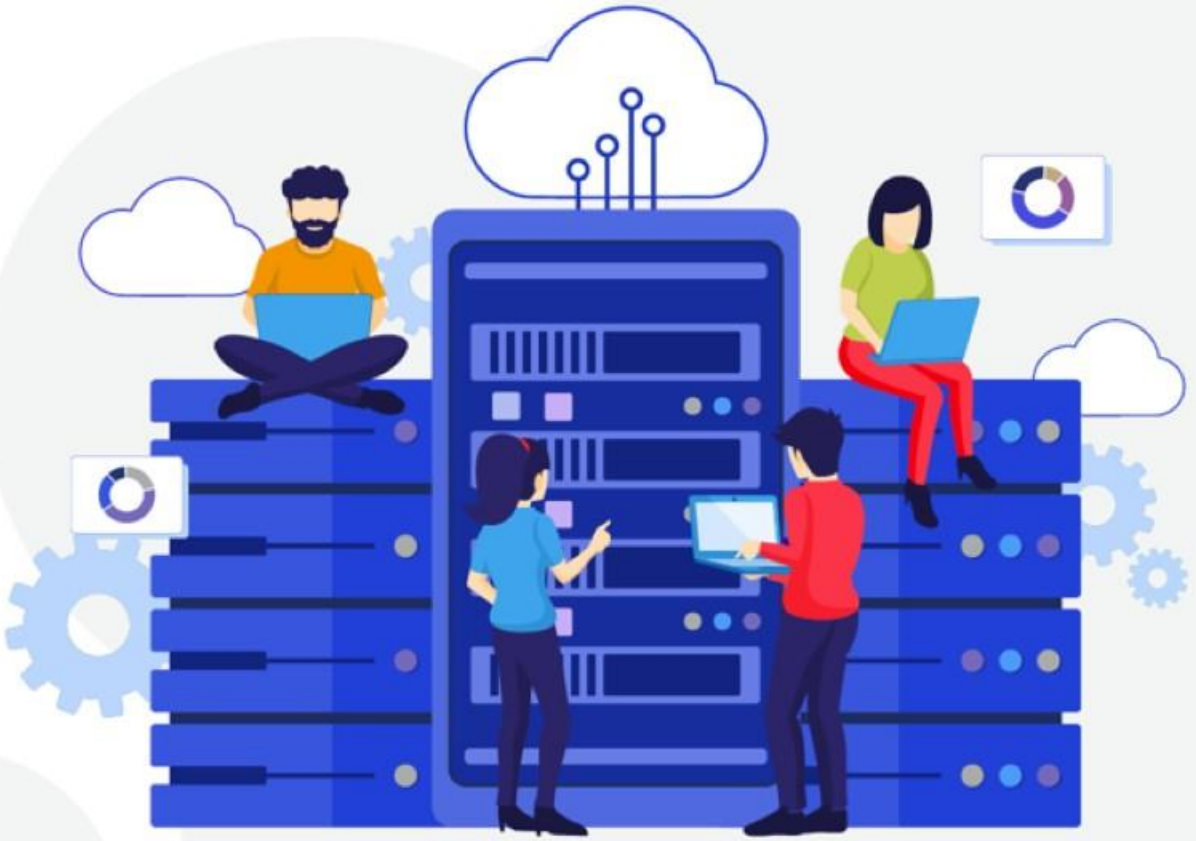
CICLO 4A

[FORMACIÓN POR CICLOS]

Profundización

Web

Gestión de la configuración



Ingeni@ Soluciones TIC

UNIVERSIDAD DE ANTIOQUIA

Facultad de Ingeniería

Agenda

- Presentación de tripulantes y docente.
- Presentación del curso
- Gestión de la configuración.

El futuro digital es de todos

MinTIC

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Misión TIC 2022

Diego Iván Oliveros Acosta

Docente Universidad de Antioquia.
Ingeniero de Sistemas.
Ingeniero de Electrónico.
Especialista en herramientas virtuales para la educación.
Magister en Arquitectura de Sistemas de Información.

https://scalapp.co/

FACULTAD DE INGENIERIA **UdeA**
Una facultad para la sociedad del aprendizaje

Presentación de beneficiarios.

Presentación del curso

- Metodología
- Contenido
- Evaluaciones
- Tips

El ciclo 4a de *Desarrollo de aplicaciones web*

Contenido

¿Metodología?

Principios del Manifiesto Ágil

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.

2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.

4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.

5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.

6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

7. El software funcionando es la medida principal de progreso.

8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.

9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.

10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.

12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Método pedagógico

- **Aprendizaje basado en proyecto (ABP):**
- Centrado en el tripulante no en el formador.
- Involucra a los tripulantes de una manera activa en su aprendizaje al pedirles que investiguen la respuesta a alguna pregunta o problema del mundo real y luego creen una solución concreta.
- Finalmente, los tripulantes presentan sus proyectos a una audiencia mayor.

Semana 1: Procesos para integración continua

Semana	Núcleos temáticos	Horas de los encuentros sincrónicos	Horas de trabajo independiente	Total
Semana 1	**Procesos para integración continua** • Introducción a la gestión de la configuración • Herramientas libres para la gestión de configuración • Descripción de los comandos básicos Pull, Push, Commit, Clone, etc. • Control de versiones • Políticas de los repositorios • Introducción al pull request • Introducción a React js	7	17	24

Semana 2: Bases de datos NoSQL (MongoDB)

Semana 2	**Bases de datos NoSQL (MongoDB)** • Introducción a MongoDB • La consola MongoDB • Cargando datos en la consola MongoDB • Cargando ficheros en la consola MongoDB • Introducción a las operaciones de lectura • CRUD • Crear documentos • Recuperar documentos • Eliminar documentos	7	17	24

Semana 3: GraphQL Introducción

Semana 3	**GraphQL Introducción** • ¿Qué es GraphQL? • Queries GraphQL • API GraphQL	7	17	24

Semana 4: Integración de GraphQL y Nodejs

Semana 4	**Integración de GraphQL y Nodejs**	7	17	24

Semana 5

Semana 5	**Introducción a los microservicios (Spring boot – Netflix OSS)** • Servicios Rest • Microservicios y Spring Boot • Estructura de un microservicio Spring Boot • Interacción entre microservicios **Introducción a microservicios con NetfliX OSS** • Configuración distribuida (Archaius) • Registro y autorreconocimiento (Eureka) • Enrutado (Zuul) • Llamadas servicio a servicio (Feing) • Balanceo de cargas (Ribbon) • Control de ruptura de comunicaciones con los servicios (Hystrix)	7	17	24

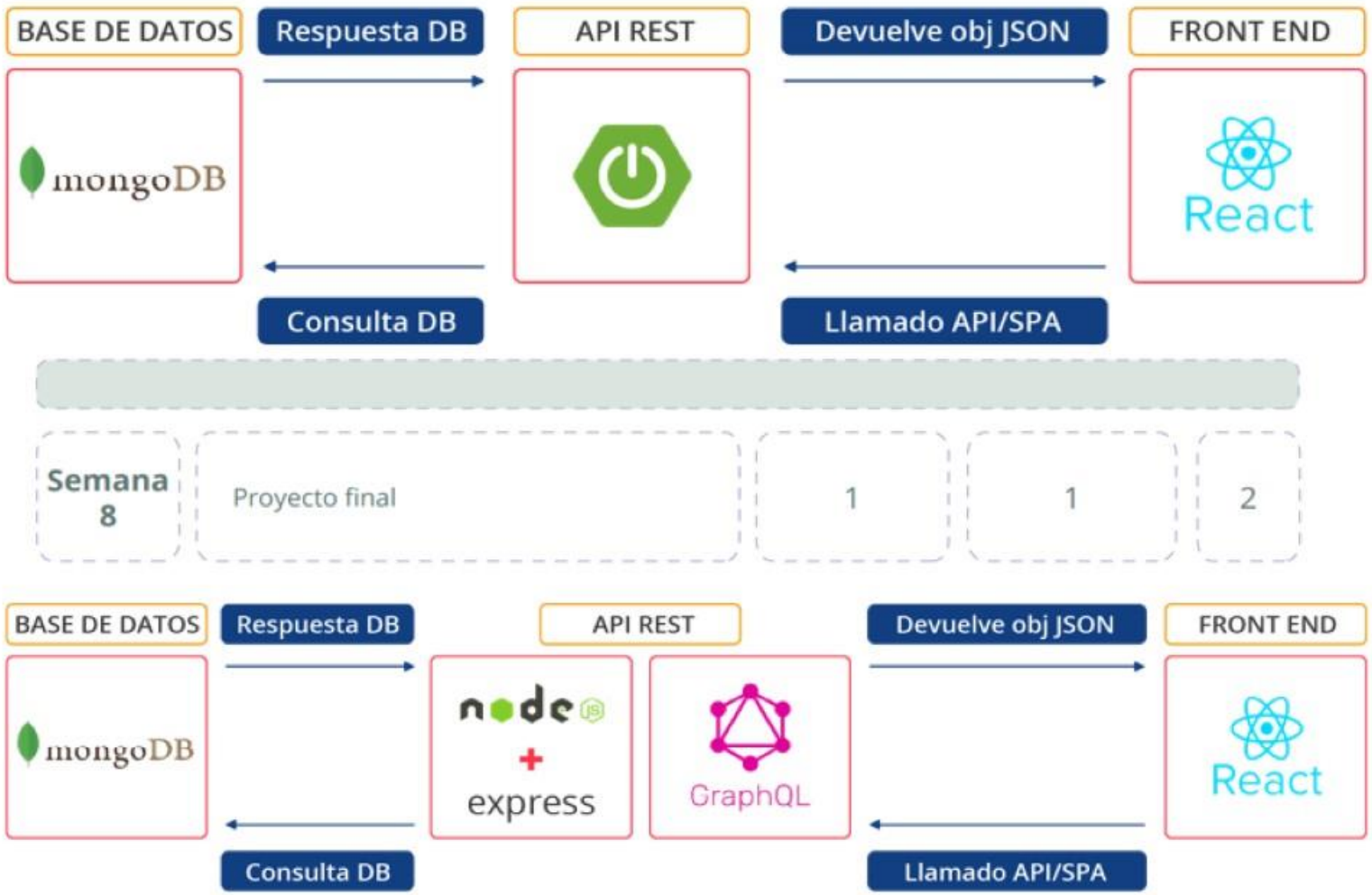
Semana 6: Contenedores para Apps (Docker)

Semana 6	**Contenedores para Apps (Docker)** • Introducción de Docker • Gestión de contenedores • Seguridad en contenedores	7	17	24

Semana 7: Introducción a DevOps

Semana 7	**Introducción a DevOps** • Introducción • Filosofía de DevOps • Comunicación entre los equipos de desarrollo • Aplicación de la cultura DevOps en una empresa • Prácticas de desarrollo con DevOps • Introducción a los pilares DevOps CI/CD/CT/RM • Configuración de CI con Azure DevOps	7	17	24

Semana 8: Proyecto final



Tema 1: Gestión de la configuración.

Fundamentos

Gestión de la configuración, se define como:

- **Definición, control, despliegue, liberación, cambio, documentación y reporte de la configuración de las partes del un sistema.**
- Conjunto de actividades diseñadas para identificar y definir los elementos en el sistema que cambia y controlar el cambio de los artefactos a lo largo del ciclo de vida del desarrollo.
- Establece mecanismos para gestionar las diferentes versiones de los ítems para auditar e informar los cambios.
- El propósito es establecer la integridad del producto software a través del proceso de software.
- Los cambios no controlados en un desarrollo de software hacen que se fracase en el proyecto.

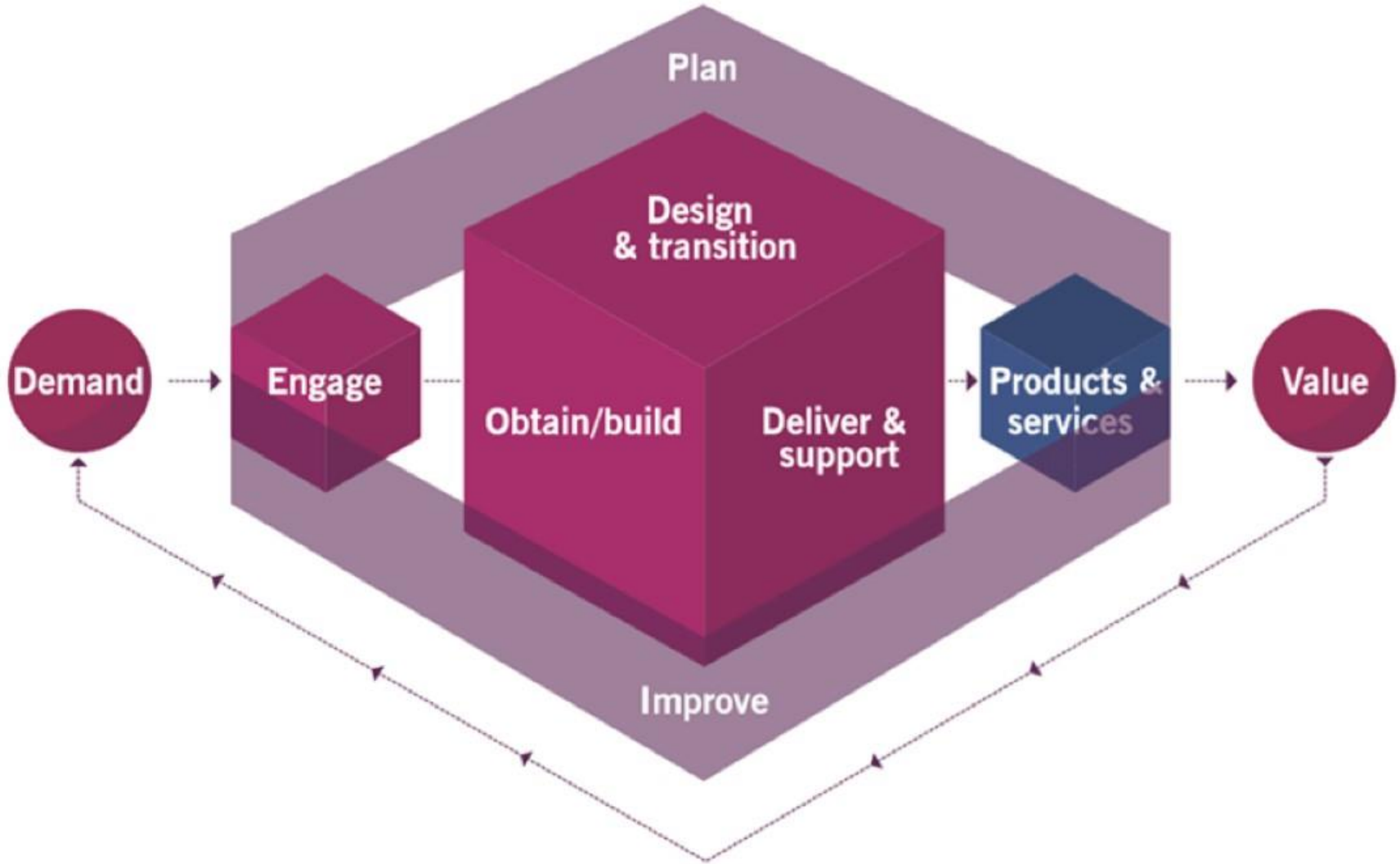
Gestión de la configuración

- "El arte de **coordinar el desarrollo de software** para minimizar la confusión se denomina gestión de configuración.
- La gestión de configuración es el arte de identificar, organizar y controlar las modificaciones que sufre el software que construye un **equipo** de programación.
- ***La meta es maximizar la productividad minimizando los errores".***

Roger S. Pressman

Propósito

- Garantizar que la información sobre la configuración de los servicios sea precisa y confiable, y los CI(elemento de configuración) que los respaldan están disponibles cuándo y dónde se necesitan.
- Esto incluye información sobre cómo se configuran los CI y las relaciones entre ellos.


Plan
Design
& transition
Demand
Engage
Obtain/build
Deliver &
support
Products &
services
Value
Improve

Glosario: Gestión de la configuración del repositorio

- **Rama (Branch).** Es una línea independiente de trabajo. Se usa con el fin de dejar intacta la rama principal (*branch master*) mientras se realizan las modificaciones en otras ramas y luego se pueda ir comprobando la correctitud de todos los cambios para poder mezclar (*merge*) con la rama principal y así poder desarrollar colaborativamente sin generar errores.
- **Clonar.** Cuando de repositorios/directorios de artefactos se trata, es duplicar en el equipo local todos los elementos que se encuentran en la nube organizados claramente mediante un proyecto de desarrollo de software.
- **Multiplataforma.** Es la propiedad de cualquier componente o artefacto para ser ejecutado en más de un sistema operativo.

Gestión de la configuración con GIT

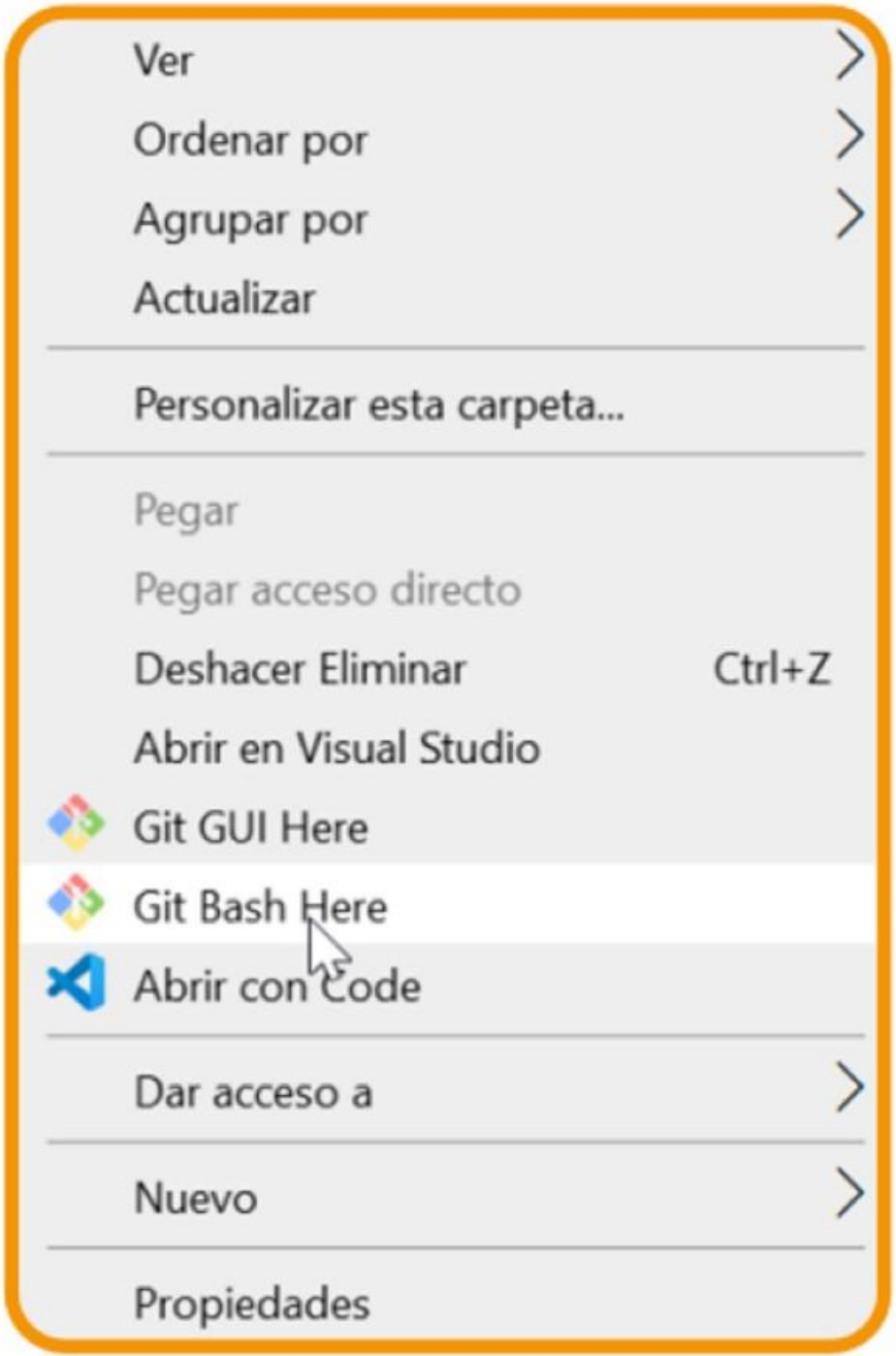
- Es un sistema de control de versiones distribuido, de código abierto, creado por Linus Torvalds.
- Cada desarrollador tiene el historial completo de su repositorio de código de manera local.
- Es multiplataforma, es decir, se pueden usar y crear repositorios locales en todos los sistemas operativos más comunes

Git

- Es un software para el control de versiones.
- Es eficiente y confiable en el versionamiento del código fuente.
- Su propósito es llevar el registro de los cambios en los artefactos durante el ciclo de vida del desarrollo del producto.
- Es un software libre distribuido bajo los términos de la licencia pública general GNU.
- Físicamente no es más que una carpeta del sistema de archivos con código fuente de una aplicación.

Git

- Es un sistema de control de versiones distribuido, de código abierto, creado por Linus Torvalds.
- Cada desarrollador tiene el historial completo de su repositorio de código de manera local.
- Es multiplataforma, es decir, se pueden usar y crear repositorios locales en todos los sistemas operativos más comunes.

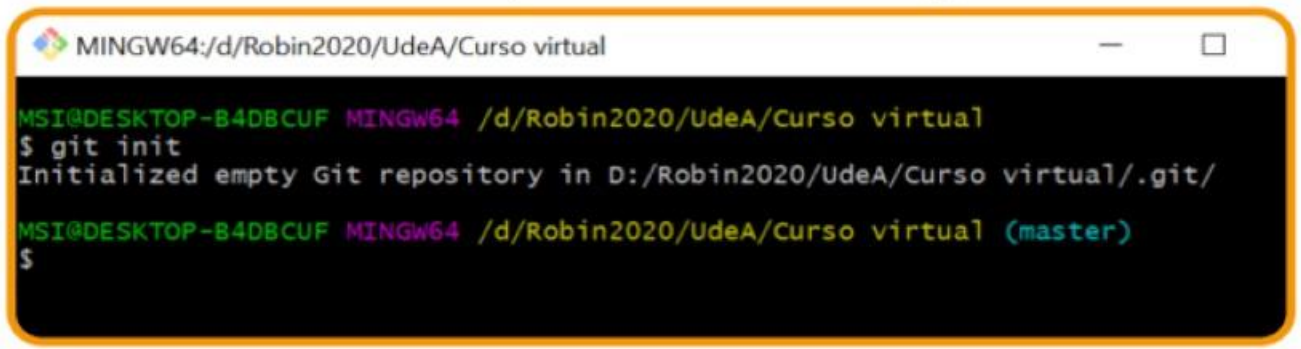


Git Bash en la práctica

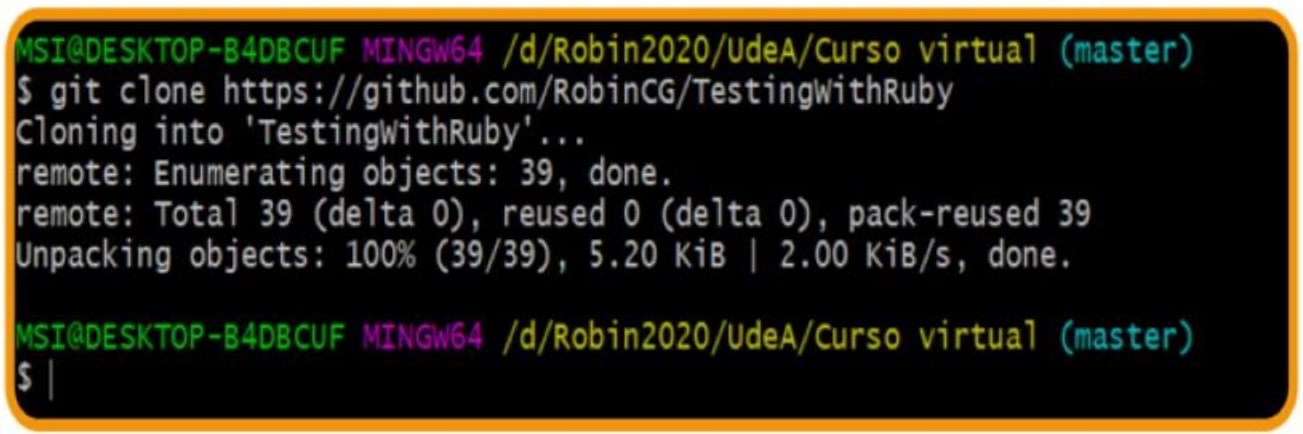
- Descargue Git Bash: https://gitforwindows.org/ Instalar con la configuración por defecto.
- En un directorio de trabajo haga clic alterno y seleccione Git Bash:

Git *Bash en la práctica

- Creamos el directorio de trabajo con el comando **git init**:



- Clonamos cualquier repositorio desde Github con el comando **git clone** "URL":



*La defin ... glosario

Git Bash en la práctica

- Listamos el contenido del directorio con el comando **ls**:
- Nos ubicamos en el nuevo directorio con el comando **cd**:
- Identificamos en qué directorio se encuentra con **pwd**:

```
MSI@DESKTOP-B4DBCUF MINGW64 /d/Robin2020/UdeA/Curso virtual (master)
$ ls
'~$Azure DevOps - Gestion de la configuracion Video 4 - copia.pptx'
'2020-00095-DEVOPS_Estructura MOOC_V1_Karenina.docx'
'2020-00095-DEVOPS_Estructura MOOC_V2_Karenina.docx'
 2020-00095-DEVOPS_Generalidades_V2_Karenina.docx
 2020-00095-DEVOPS_Semana1_V2_Karenina.docx
'Azure DevOps - Aplicacion Cultura DevOps Video 2.pptx'
'Azure DevOps - Contenido para el video Herramientas e Introduccion.pptx'
'Azure DevOps - Gestion de la configuracion Video 4 - copia.pptx'
'Azure DevOps - Introduccion Video 1.pptx'
'Azure DevOps - Pilares DevOps Video 3.pptx'
'Complementar temas DevOps.txt'
 DevOps/
'DevOps para Dummy.pdf'
'Documento muy bueno de DevOps.pdf'
 TestingWithRuby/
```

```
MSI@DESKTOP-B4DBCUF MINGW64 /d/Robin2020/UdeA/Curso virtual (master)
$ cd TestingWithRuby/
```

```
MSI@DESKTOP-B4DBCUF MINGW64 /d/Robin2020/UdeA/Curso virtual/TestingWithRuby (master)
$ pwd
/d/Robin2020/UdeA/Curso virtual/TestingWithRuby
```

Git Bash en la práctica

- Podemos usar el comando git status para identificar qué se modificó:
- Para crear una rama de trabajo usamos el comando git checkout –b :
- Para adicionar todos los cambios usamos el comando git add:
- Para confirmar los cambios usamos git commit –m “Practica de DevOps”:

```
MSI@DESKTOP-B4DBCUF MINGW64 /d/Robin2020/UdeA/Curso virtual/TestingWithRuby (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md
```

```
MSI@DESKTOP-B4DBCUF MINGW64 /d/Robin2020/UdeA/Curso virtual/TestingWithRuby (master)
$ git checkout -b feature-MyWork
Switched to a new branch 'feature-MyWork'
```

```
MSI@DESKTOP-B4DBCUF MINGW64 /d/Robin2020/UdeA/Curso virtual/TestingWithRuby (feature-MyWork)
$ git commit -m "Practica de DevOps"
[feature-MyWork befadc7] Practica de DevOps
 1 file changed, 1 insertion(+)
```

Git Bash en la práctica

- Podemos usar el comando push para publicar el nuevo cambio que se encuentra localmente hacia el servidor remoto de GitHub:
- git push origin :

```
MSI@DESKTOP-B4DBCUF MINGW64 /d/Robin2020/UdeA/Curso virtual/TestingWithRuby (feature-MyWork)
$ git push origin feature-MyWork
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 356 bytes | 356.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'feature-MyWork' on GitHub by visiting:
remote:      https://github.com/RobinCG/TestingWithRuby/pull/new/feature-MyWork
remote:
To https://github.com/RobinCG/TestingWithRuby
 * [new branch]      feature-MyWork -> feature-MyWork
```

Entendiendo Pull

- Gestión de la configuración de las fuentes de los proyectos
- El comando git pull es una forma abreviada de simplificar todos los procesos que realizamos con otros comandos, como por ejemplo git fetch o git merge y nos permite ahorrar tiempo.
- Cuando realizamos un git pull estamos sincronizando y “jalándonos” todos los cambios del repositorio remoto a la rama en la estamos trabajando localmente sin la necesidad de ejecutar ningún comando extra.

Ejemplos en los que podemos utilizar este comando:

- 1. git pull: Trae una copia remota y realiza una mezcla de fuentes desde la rama Maestra a nuestra rama remota
- 2. git pull origin : Hace una actualización en nuestra rama local desde un Branch remoto que indiquemos en el comando

Entendiendo Push

- **El comando git push básicamente se usa para cargar contenido del repositorio local a un repositorio remoto.**
- git push, “empujar” las líneas de código nuevas o modificadas en nuestras ramas locales y que ahora queremos incorporarlas en la rama Maestra.
- Estamos transfiriendo cambios **confirmados** (commit) desde el repositorio local a un remoto para sobrescribir cambios.
- Git push se usa para publicar y cargar cambios locales a un repositorio central después de modificarlo. Usando este comando se podrá compartir las modificaciones hechas localmente con otros miembros del equipo.

Ejemplos en los que podemos utilizar este comando:

- 1. git push : Envía la rama especificada a otra junto con todos los commits y objetos internos.
- 2. git push --all: Envía todas tus ramas a una rama remota especificada

Entendiendo git add & commit

- Antes de la ejecución de git commit, se utiliza el comando git add para pasar o “preparar” los cambios en el proyecto que se almacenarán en una confirmación.
- **commit** es una foto instantánea sobre los cambios en las fuentes que llevamos en el momento.

- **Permite a los desarrolladores:**
  - Acumular “confirmaciones” en los repositorios locales sin el perjuicio de ir los a perder por alguna mala práctica o apagados del sistema local.
  - Trabajar aisladamente y aplazar la integración hasta que estén completamente seguros para fusionar el trabajo con el de otros desarrolladores del proyecto.
- Estos dos comandos, git commit y git add, son dos de los que se utilizan más frecuentemente.

Ejemplos en los que podemos utilizar este comando:

- 1. git commit: Confirma la instantánea preparada
- 2. git commit -m : Confirma la instantánea y adiciona un comentario para su trazabilidad
- 3. git commit -a: Confirma una instantánea de todos los cambios del directorio de trabajo que se han añadido con git add.

Entendiendo git clone

Se usa para:

- Crear una copia de un repositorio o rama especifica dentro de un repositorio.
- Generar una copia local de lo que tengamos en nuestro repositorio central.
- Clonar un repositorio en un directorio recién creado, crea ramas de seguimiento remoto para cada rama en el repositorio clonado.
- Cuando clonamos un repositorio obtenemos todo el repositorio, los archivos, las ramas y todas las confirmaciones hechas previamente.
- Una vez realizada la clonación, no se necesitará clonarlo nuevamente, esta acción sólo se realiza una sola vez, al inicio de la interacción con el proyecto.

Ejemplos en los que podemos utilizar este comando:

- Algunos ejemplos en los que podemos utilizar el comando serían:
- 1. git clone : Baja todo el repositorio existente en la rama master
- 2. git clone –mirror: Clona un repositorio, pero sin la posibilidad de editarlo

Referencias

- https://git-scm.com/
- https://es.reactjs.org/
- https://github.com/
- https://git-scm.com/doc
- https://education.github.com/git-cheat-sheet-education.pdf
- https://rogerdudler.github.io/git-guide/index.es.html
- https://es.reactjs.org/tutorial/tutorial.html


Gracias