DESARROLLO

WEB FULL STACK

-INTERMEDIO

Autor de contenido

Andrés Fernando Pineda Guerra

EMPEZAR

Tabla de Contenido

Presentación

En el curso de desarrollador Full Stack como componente intermedio, podrán adquirir las habilidades y lenguajes necesarios para el desarrollo web, enfocándose en sus grandes pilares, como lo son Front End, Back End, Diseño y modelamiento de aplicaciones y documentación de código.
El curso trata temas emergentes tales como, la seguridad informática, desarrollo de aplicaciones móviles, gestión de base de datos, todo esto basado en la metodología Scrum. De la misma manera, se hace énfasis en el manejo de proyectos tanto en los módulos de desarrollo como los módulos de gestión de proyectos de TI.

Objetivos del curso (competencias)

Objetivo general

Formar a los participantes en el desarrollo web en todo el ciclo de vida del software, en donde adquieran los conocimientos básicos para implementar soluciones web.

Objetivo específico

- Conocer los conceptos y teoría básica del desarrollo web.

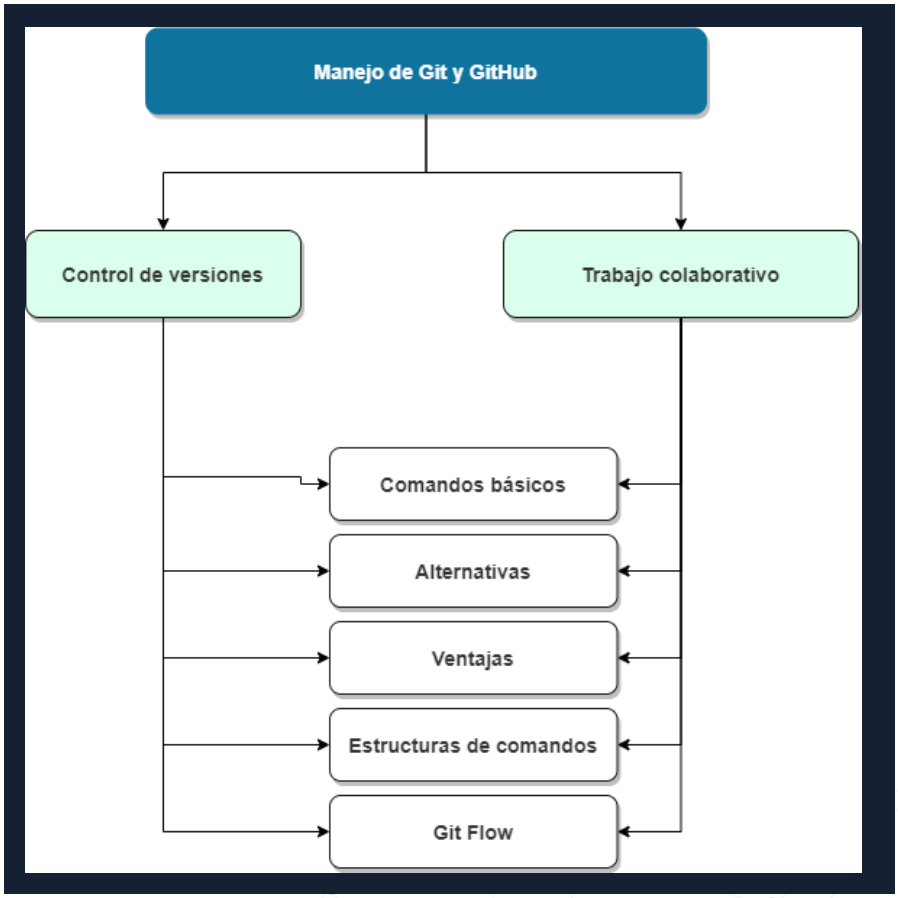
- Identificar y conocer los diferentes lenguajes de programación y herramientas para el desarrollo web.

- Aplicar las diferentes tecnologías web, tendencias y herramientas en el desarrollo de soluciones web enfocadas a proyectos.

- Diseñar, desarrollar e implementar soluciones web básicas en donde se integren los componentes de Front End, Back End, seguridad, redes y buenas prácticas utilizando metodologías ágiles.

- Identificar y conocer los conceptos básicos para el desarrollo móvil, así como aplicar su desarrollo en aplicaciones básicas.

Mapa de contenido de la unidad



Módulo 13

Manejo de Git y GitHub

Ideas clave

Manejo, introducción y principios de Git, comandos básicos.
Trabajo colaborativo en mediante Github e instrucciones de Git, ventajas, estructuras de comandos git y git flow.

13.1. Control de versiones

Github

Es una herramienta esencial para los desarrolladores de software, y su popularidad es inigualable. Actualmente cuenta con más de 25 millones de usuarios. Se trata de un número considerable de profesionales que recurren a GitHub para mejorar el flujo de trabajo y la colaboración.

En general Github es un servicio basado en la nube, que aloja un sistema de control de versiones llamado GIT, este permite mediante unas líneas de comando mantener actualizado y controlado el ambiente de desarrollo en algún proyecto de software.

https://github.com/

Comandos básicos de git

git clone <https://link-con-nombre-del-repositorio>

Git clone básicamente realiza una copia idéntica de la última versión de un proyecto en un repositorio y la guarda en tu ordenador.

git branch <nombre-de-la-rama>

Las ramas (branch) son altamente importantes en el mundo de Git. Usando ramas, varios desarrolladores pueden trabajar en paralelo en el mismo proyecto simultáneamente. Podemos usar el comando git branch para crearlas, listarlas y eliminarlas.

git push <nombre-remoto> <nombre-rama>

Este comando creará una rama en local. Para enviar (push) la nueva rama al repositorio remoto.

git checkout <nombre-de-la-rama>

Usaremos git checkout principalmente para cambiarte de una rama a otra. También lo podemos usar para chequear archivos y commits.

git status

Podemos encontrar información como:

Si la rama actual está actualizada

Si hay algo para confirmar, enviar o recibir (pull).

Si hay archivos en preparación (staged), sin preparación(unstaged) o que no están recibiendo seguimiento (untracked)

Si hay archivos creados, modificados o eliminados

git add <archivo>

Añade los cambios para poder hacer el commit.

git commit -m “mensaje de confirmación”

Git commit es cómo establecer un punto de control en el proceso de desarrollo al cual puedes volver más tarde si es necesario.

También necesitamos escribir un mensaje corto para explicar qué hemos desarrollado o modificado en el código fuente.

git push <nombre-remoto> <nombre-de-tu-rama>

Después de haber confirmado tus cambios, el siguiente paso que quieres dar es enviar tus cambios al servidor remoto. Git push envía tus commits al repositorio remoto.

git pull <nombre-remoto>

El comando git pull se utiliza para recibir actualizaciones del repositorio remoto. Este comando es una combinación del git fetch y del git merge lo cual significa que cundo usemos el git pull recogeremos actualizaciones del repositorio remoto (git fetch) e inmediatamente aplicamos estos últimos cambios en local (git merge).

13.2. Trabajo colaborativo

Control de versiones: Introducción, ventajas, alternativas

Git

Git es un sistema de control de versiones distribuido que permite a los equipos de desarrollo de software tener varias copias locales del código base del proyecto indepen-

dientes entre sí. Estas copias, o ramas, se pueden crear, fusionar y eliminar rápidamente, lo que permite a los equipos experimentar, con un costo informático mínimo, antes de fusionarse con la rama principal. Git es conocido por su velocidad, compatibilidad de flujo de trabajo y base de código abierto.

Git y sus alternativas

Los equipos de desarrollo de software prefieren Git a otros sistemas de control de versiones, como CVS, Mercurial y Perforce, porque Git tiene la adaptabilidad, la velocidad y la estabilidad necesarias para prosperar en mercados de ritmo acelerado. No es de extrañar que el 87,2 % de los desarrolladores utilicen Git para el control de versiones.
Si las organizaciones quieren satisfacer rápidamente la demanda de los clientes y los objetivos comerciales, el control de versiones de Git es la forma más sencilla de empoderar a los desarrolladores.
La capacidad de Git para almacenar un historial completo de un proyecto localmente es un beneficio, porque las ramas de Git son livianas y el protocolo es rápido, por lo que los colaboradores pueden estar en cualquier lugar, incluso con una conexión deficiente y aun así sincronizar una copia local con cualquier otro miembro del equipo.
Ventajas

Git tiene flexibilidad de flujo de trabajo

Con Git, los equipos pueden trabajar juntos usando varias estrategias de bifurcación que no son posibles en otros sistemas de control de versiones. Las capacidades de bifurcación de Git ofrecen a los usuarios la posibilidad de seleccionar un flujo de trabajo según el tamaño de un proyecto o equipo o procesos únicos. Algunos de los flujos de trabajo de Git más populares incluyen el desarrollo centralizado, la bifurcación de funciones, el desarrollo basado en troncos y GitFlow.

Git es rápido

Los usuarios tienen un repositorio local con un historial completo, por lo que no hay retrasos en la comunicación con un servidor, que es una experiencia que a menudo encuentran los usuarios de sistemas centralizados, como CVS, Subversion y Perforce.

Git es confiable

Git inherentemente tiene múltiples copias de seguridad, porque cada usuario tiene un repositorio local. Si hay un bloqueo, una copia podría reemplazar el servidor principal. Otro beneficio de los repositorios locales es que los usuarios pueden continuar comprometiéndose sin conexión si están viajando o lidiando con problemas de red.

Git es colaborativo

Git facilita el desarrollo colaborativo con su modelo de ramificación. Los usuarios pueden crear una rama, experimentar y fusionar el código con la rama principal si funciona bien. Git realiza un seguimiento de los cambios durante las revisiones de código y los miembros del equipo pueden colaborar en las solicitudes de combinación.

Estructura de comandos Git

e los ca	**e los ca**
Add	git add [file]
Commit	git commit -m “[Type in the commit mes-sage]”
Diff	git diff git diff -staged git diff [first branch] [second branch]
Reset	git reset [file] git reset [commit] git reset -hard [commit]
Status	git status
Remove	git rm [file]
Log	git log git log -follow[file]
Tag	git tag [commitID]
Branch	git branch git branch [branch name]

Checkout	git checkout [branch name] git checkout -b [branch name]
Merge	git merge [branch name]
Remote	git remote add [variable name] [Remote Server Link]
Push	git push [variable name] master git push [variable name] [branch] git push -all [variable name] git push [variable name] :[branch name] git push [variable name] master git push [variable name] [branch] git push [variable name] master git push [variable name] [branch] git push -all [variable name] git push [variable name] :[branch name]
Pull	git pull [Repository Link]
Stash	git stash save git stash pop git stash list git stash drop

Git Flow

Git flow es una popular estrategia de bifurcación de Git destinada a simplificar la administración de versiones y fue presentada por el desarrollador de software Vincent Driessen en 2010. Fundamentalmente, Git flow implica aislar el trabajo en diferentes tipos de ramas de Git.
En Git Flow hay cinco tipos de ramas diferentes:

- Main:

Antes esta rama principal llevaba el nombre master pero ha quedado obsoleto. El propósito de la rama principal en el flujo de trabajo de Git es contener código listo para producción que se pueda publicar.

En Git Flow, la rama principal se crea al comienzo de un proyecto y se mantiene durante todo el proceso de desarrollo. La rama se puede etiquetar en varios commits para indicar diferentes versiones del código, y otras ramas se fusionarán con la rama principal después de que hayan sido suficientemente examinadas y probadas.

- Develop:

La rama de desarrollo se crea al comienzo de un proyecto y se mantiene a lo largo del proceso de desarrollo, y contiene código de preproducción con características recientemente desarrolladas que están en proceso de prueba.

Las funciones recién creadas deben basarse en la rama de desarrollo y luego volver a fusionarse cuando estén listas para la prueba.

- Feature:

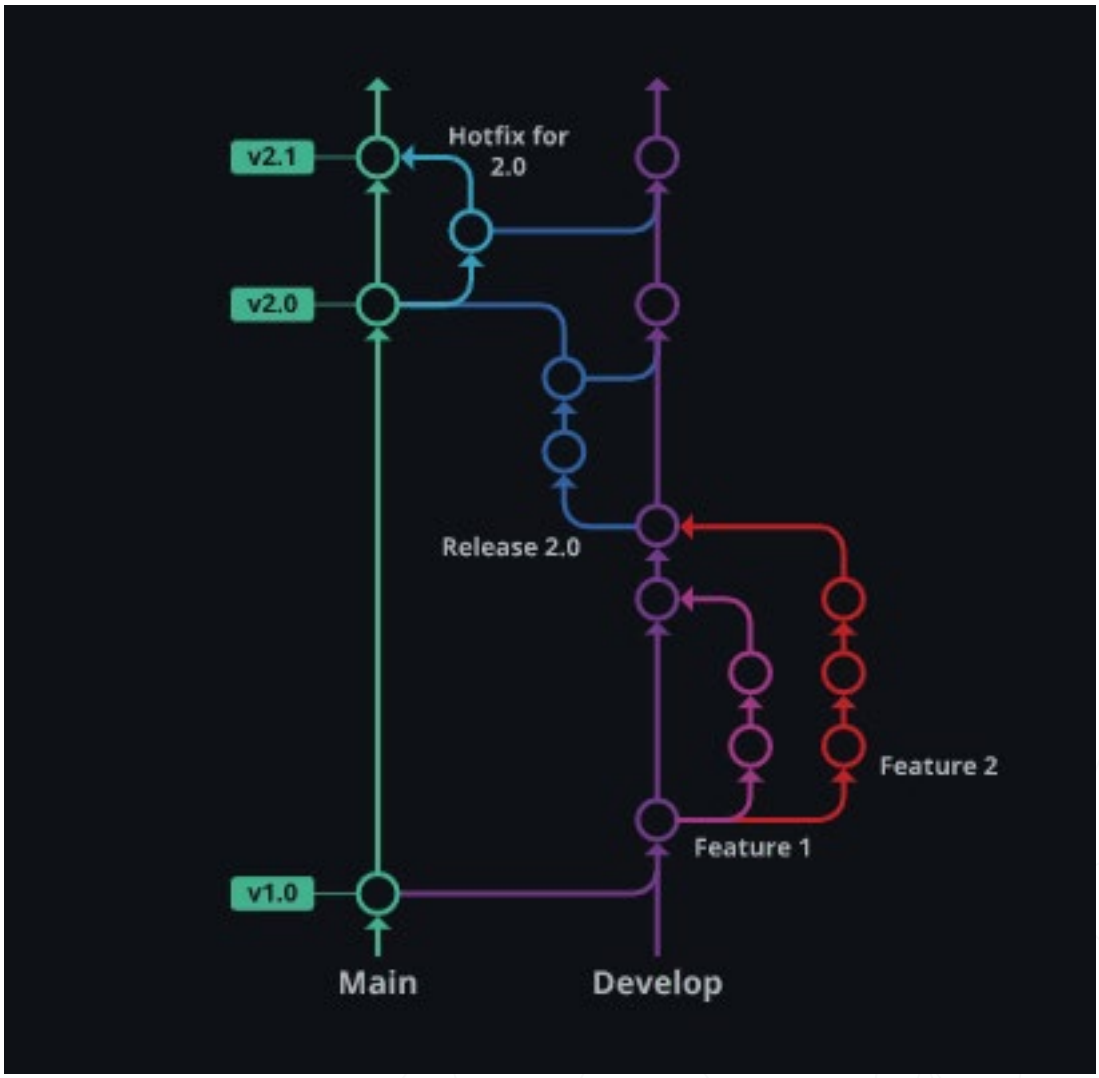
La rama feature es el tipo de rama más común en el flujo de trabajo de Git. Se utiliza al agregar nuevas características a su código.
Cuando se trabaja en una nueva característica (feature), es necesario iniciar una rama feature a partir de la rama de desarrollo y luego fusionar los cambios nuevamente en la rama de desarrollo cuando la función se complete y se revise correctamente.

- Release:

La rama de release debe usarse al preparar nuevos lanzamientos de producción. Por lo general, el trabajo que se realiza en las ramas de release se refiere a los toques finales y errores menores específicos de la liberación de código nuevo, con código que debe abordarse por separado de la rama de desarrollo principal.

- Hotfix:

En Git Flow, la rama hotfix se usa para abordar rápidamente los cambios necesarios en la rama principal.
La base de la rama hotfix debe ser su rama principal y debe fusionarse con las ramas main y de develop. La fusión de los cambios de la rama de hotfix en la rama de desarrollo es fundamental para garantizar que la corrección persista la próxima vez que se publique la rama main.


v2.1
Hotfix for
2.0
v2.0
Release 2.0
Feature 2
Feature 1
v1.0
Main
Develop

Otros materiales para profundizar

Recursos de video

Fazt (Director). (2018). Git y Github | Curso Práctico de Git y Github Desde Cero. https://www.youtube.com/watch?v=HiXLkL42tMU

Referencias bibliográficas de la unidad

GitHub.com Documentación de la Ayuda. (s. f.). GitHub Docs. https://docs.github.com/es

Atlassian. (s. f.). Resumen de. Bitbucket. https://bitbucket.org/product/es/guides/getting-started/overview

Git. (s. f.). https://git-scm.com/

Dauzon, S. (2018). Git: controle la gestión de sus versiones: conceptos, utilización y casos prácticos. Ediciones ENI..

Git and GitHub: The complete git and GitHub course. (2020). Packt Publishing.

Git: The ultimate beginner's guide to learn git step by step. (2020). Independently Published.

Git: The ultimate beginner's guide to learn git step by step. (2020b). Mem Lnc, John Bach.

Git workshop. (s. f.). ImageJ Wiki. https://imagej.net/develop/git/workshop


ALCALDÍA MAYOR
DE BOGOTÁ D.C.
SECRETARÍA DE EDUCACIÓN


ATENEA
AGENCIA DISTRITAL PARA LA EDUCACIÓN
SUPERIOR LA CIENCIA Y LA TECNOLOGÍA


BOGOTÁ


UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS
Acreditación Institucional de Alta Calidad