HERRAMIENTAS DE DESARROLLO

Profesor: *Rafael O. Castro Veramendi, Ph.D.*

Herramientas de Colaboración de Equipo

Unidad 2

Semana 09 – Sesión 2 (S09.s2)

Profesor: *Rafael O. Castro Veramendi, Ph.D.*

¿Tienen consultas sobre la clase anterior?

Temario

Herramientas de colaboración de equipo.

Logro de Aprendizaje de la Sesión

Al finalizar la sesión, el estudiante reconoce características de herramientas de colaboración de equipos, a través de participaciones grupales.

Conocimientos Previos del Tema

¿Qué es una rama en Git?

En Git, una rama (branch) es una línea de desarrollo independiente dentro de un repositorio. Permite trabajar en nuevas funcionalidades, arreglar errores o experimentar con cambios sin afectar la rama principal (main o master). Son como bifurcaciones del código, creando un nuevo camino de desarrollo paralelo.

Utilidad del Tema

¿Por qué son importantes las herramientas de colaboración de equipos?

Porque permiten un desarrollo más organizado, flexible y colaborativo del código.

Secuencia y Explicación

A continuación desarrollamos la evaluación …

Herramientas de Colaboración de Equipo

Herramientas de Colaboración para Equipos de Desarrollo con Git

Git es la base del control de versiones en el desarrollo de software, pero su potencia se multiplica al combinarse con herramientas diseñadas para facilitar la colaboración en equipo. Estas herramientas ayudan a gestionar flujos de trabajo, revisión de código, integración continua, y comunicación. A continuación, se detallan las principales categorías y herramientas:

Herramientas de Colaboración para Equipos de Desarrollo con Git

1. Plataformas de Hosting de Git (Repositorios Remotos)

Permiten almacenar repositorios Git en la nube, gestionar permisos, y colaborar mediante funcionalidades avanzadas.

- **GitHub**:
  - **Características**: Pull Requests, Issues, Projects, Wiki, GitHub Actions (CI/CD), Codespaces.
  - **Ventaja**: Ecosistema masivo de integraciones (Slack, Jira, etc.) y comunidad open source.
  - **Ejemplo**: Usado para proyectos públicos y privados; ideal para equipos que priorizan integraciones.

Herramientas de Colaboración para Equipos de Desarrollo con Git

1. Plataformas de Hosting de Git (Repositorios Remotos)

Permiten almacenar repositorios Git en la nube, gestionar permisos, y colaborar mediante funcionalidades avanzadas.

- **GitLab**:
  - **Características**: CI/CD integrado, Issues, Wiki, revisiones de código, seguridad DevSecOps.
  - **Ventaja**: Todo-en-uno (desde repositorios hasta despliegues). Versión autohospedada disponible.
- **Bitbucket (Atlassian)**:
  - **Características**: Integración nativa con Jira y Trello, pipelines de CI/CD, permisos granulares.
  - **Ventaja**: Ideal para equipos que ya usan el ecosistema Atlassian (Jira, Confluence).
- **Azure DevOps (Microsoft)**:
  - **Características**: Repositorios Git, gestión de tareas, pipelines CI/CD, tableros ágiles.
  - **Ventaja**: Integración perfecta con herramientas Microsoft y servicios en la nube Azure.

Universidad Tecnológica del Perú

Herramientas de Colaboración para Equipos de Desarrollo con Git

2. Herramientas de CI/CD (Integración Continua/Despliegue Continuo)

Automatizan pruebas, builds y despliegues al integrarse con repositorios Git.

- **Jenkins**:
  - **Uso**: Servidor de automatización open source. Se configura con Jenkinsfile en el repositorio.
  - **Integración con Git**: Ejecuta pipelines al detectar cambios en ramas específicas.
- **GitHub Actions**:
  - **Uso**: Permite definir flujos de trabajo (workflows) directamente en el repositorio de GitHub.
  - **Ejemplo**: Automatizar tests al hacer un push a `main` o desplegar en AWS tras un merge.

Herramientas de Colaboración para Equipos de Desarrollo con Git

2. Herramientas de CI/CD (Integración Continua/Despliegue Continuo)

Automatizan pruebas, builds y despliegues al integrarse con repositorios Git.

- **GitLab CI/CD**:
  - **Uso**: Configuración mediante `.gitlab-ci.yml` en el repositorio. Ejecuta jobs en runners auto-gestionados o compartidos.
- **CircleCI**:
  - **Uso**: Soporta integración con GitHub, GitLab y Bitbucket. Configuración mediante `config.yml`.

Herramientas de Colaboración para Equipos de Desarrollo con Git

3. Gestión de Proyectos y Tareas

Vinculan issues de Git con tareas, sprints y métricas del equipo.

- **Jira (Atlassian)**:
  - **Integración con Git**: Enlaza branches, commits y pull requests con tickets de Jira.
  - **Ejemplo**: Un commit con `JIRA-123` actualiza automáticamente el estado del ticket.
- **Trello**:
  - **Power-Ups**: Integración con GitHub y GitLab para vincular tarjetas a repositorios.
- **Asana**:
  - **Integración**: Conecta tareas con commits y pull requests mediante apps como "GitHub for Asana".

Herramientas de Colaboración para Equipos de Desarrollo con Git

4. Revisión de Código y Colaboración en PR/MR

Facilitan la discusión y mejora del código antes de fusionarlo.

- **Pull Requests (GitHub) / Merge Requests (GitLab)**:
  - **Funcionalidades**: Comentarios en líneas específicas, aprobaciones requeridas, revisión diferencial (diff).
- **Gerrit**:
  - **Uso**: Herramienta especializada en revisiones de código para equipos grandes (ej: Android Open Source Project).
- **Phabricator**:
  - **Características**: Incluye Differential para revisiones, Herald para automatizar reglas de revisión.

Universidad Tecnológica del Perú

Herramientas de Colaboración para Equipos de Desarrollo con Git

5. Herramientas de Comunicación en Tiempo Real

Mantienen al equipo sincronizado sobre cambios en el repositorio.

- **Slack**:
  - **Integraciones**: Notifica nuevos pull requests, comentarios en issues o despliegues exitosos mediante bots como **GitHub Slack Integration** o **GitLab Slack Notifier**.
- **Microsoft Teams**:
  - **Conectores**: Canaliza actualizaciones de repositorios de GitHub o Azure DevOps a un canal de Teams.

Herramientas de Colaboración para Equipos de Desarrollo con Git

6. Herramientas de Monitoreo de Código y Calidad

Analizan el código en el repositorio para mantener estándares.

- **SonarQube**:
  - **Uso**: Escanea repositorios para detectar bugs, vulnerabilidades y deuda técnica. Se integra con Jenkins, GitHub Actions, etc.
- **CodeClimate**:
  - **Características**: Proporciona métricas de calidad del código y cobertura de tests. Integración directa con GitHub y GitLab.
- **Codacy**:
  - **Funcionalidad**: Revisión automática de código en cada pull request.

Herramientas de Colaboración para Equipos de Desarrollo con Git

7. Plataformas DevOps Todo-en-Uno

Combinan repositorios Git, CI/CD, y gestión de infraestructura.

- **GitLab Ultimate**:
  - Ofrece desde planificación hasta monitoreo, pasando por seguridad y despliegue.
- **Azure DevOps**:
  - Incluye repositorios Git, pipelines, y gestión de artefactos en un solo lugar.
- **AWS CodeCommit + CodePipeline**:
  - Ecosistema de AWS para equipos que priorizan integración con servicios en la nube.

8. Herramientas para Resolución de Conflictos

Ayudan a gestionar merges complejos.

- **KDiff3**:
  - **Uso**: Herramienta gráfica para resolver conflictos de merge en Git.
- **Beyond Compare**:
  - **Funcionalidad**: Compara archivos y directorios para resolver diferencias durante merges.

Herramientas de Colaboración para Equipos de Desarrollo con Git

9. Documentación Colaborativa

Mantienen documentación alineada con el código.

- **GitHub Wiki**:
  - Documentación vinculada directamente al repositorio.
- **GitBook**:
  - Permite crear documentación técnica conectada a repositorios Git.
- **Read the Docs**:
  - Genera documentación automáticamente desde archivos Markdown o Sphinx en el repositorio.

Herramientas de Colaboración para Equipos de Desarrollo con Git

10. Herramientas de Seguridad y Auditoría

Protegen repositorios y cumplen con normativas.

- **Snyk**:
  - Escanea dependencias en el repositorio para detectar vulnerabilidades.
- **GitGuardian**:
  - Detecta secretos expuestos (API keys, contraseñas) en commits y branches.

Herramientas de Colaboración para Equipos de Desarrollo con Git

Consideraciones para Elegir Herramientas

1. **Tamaño del equipo**:
   - Equipos pequeños: GitHub + Slack + GitHub Actions.
   - Equipos grandes: GitLab + Jira + SonarQube.
2. **Integraciones**:
   - ¿Necesitas conectar con herramientas de diseño (Figma) o monitoreo (Datadog)?
3. **Presupuesto**:
   - GitHub y GitLab tienen planes gratuitos para proyectos open source.
   - Herramientas como Jenkins son gratuitas pero requieren auto-gestión.
4. **Seguridad**:
   - Empresas reguladas (ej: salud, finanzas) priorizan herramientas con auditoría y permisos granulares (ej: Azure DevOps).

Referencia adicional:

Referencias adicionales:

Learn to Use GitHub Actions: a Step-by-Step Guide

GitHub Actions Tutorial – Getting Started & Examples

Ejemplo de uso de github actions

1. Local Machine Setup

1. Local Machine Setup

Create Project

bash

```bash
# Create project directory
mkdir simple-node-actions
cd simple-node-actions

# Initialize Node.js project
npm init -y

# Create basic files
mkdir src
touch src/index.js
touch src/index.test.js
```

Ejemplo de uso de github actions

1. Local Machine Setup

Create Project

bash

```bash
# Create project directory
mkdir simple-node-actions
cd simple-node-actions

# Initialize Node.js project
npm init -y

# Create basic files
mkdir src
touch src/index.js
touch src/index.test.js
```

Add Code

src/index.js

javascript

```javascript
function add(a, b) {
  return a + b;
}

module.exports = { add };
```

src/index.test.js

javascript

```javascript
const { add } = require('./index');

test('adds 1 + 2 to equal 3', () => {
  expect(add(1, 2)).toBe(3);
});
```

Ejemplo de uso de github actions

1. Local Machine Setup

Install Dependencies

bash

```
npm install --save-dev jest
```

Update package.json

json

```
{
  "name": "simple-node-actions",
  "version": "1.0.0",
  "scripts": {
    "test": "jest"
  },
  "devDependencies": {
    "jest": "^29.0.0"
  }
}
```

Test Locally

bash

```
npm test
# Should pass with 1 test
```

Ejemplo de uso de github actions

2. GitHub Setup

Initialize Git Repository

bash

```
git init
git add .
git commit -m "Initial commit"
```

Create GitHub Repository

1. Go to GitHub.com
2. Click "+" → "New repository"
3. Name it "simple-node-actions"
4. Don't initialize with README (we already have files)

Push to GitHub

bash

```
git remote add origin https://github.com/YOUR-USERNAME/simple-node-actions.git
git branch -M main
git push -u origin main
```

Ejemplo de uso de github actions

2. GitHub Setup

Initialize Git Repository

bash

```
git init
git add .
git commit -m "Initial commit"
```

Create GitHub Repository

1. Go to GitHub.com
2. Click "+" → "New repository"
3. Name it "simple-node-actions"
4. Don't initialize with README (we already have files)

Push to GitHub

bash

```
git remote add origin https://github.com/YOUR-USERNAME/simple-node-actions.git
git branch -M main
git push -u origin main
```

Ejemplo de uso de github actions

3. Set Up GitHub Actions

bash

```
mkdir -p .github/workflows
touch .github/workflows/ci.yml
```

.github/workflows/ci.yml

yaml

```yaml
name: Node.js CI

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Use Node.js
      uses: actions/setup-node@v3
      with:
        node-version: 18

    - name: Install dependencies
      run: npm install

    - name: Run tests
      run: npm test
```

Push the Workflow

bash

```
git add .github/workflows/ci.yml
git commit -m "Add GitHub Actions workflow"
git push
```

Universidad Tecnológica del Perú

Ejemplo de uso de github actions

4. Verify on GitHub

1. Go to your repository on GitHub
2. Click on the "Actions" tab
3. You should see your workflow running (or already completed)
4. Click on it to see details

Ejemplo de uso de github actions

5. Make a Change to Trigger CI

Let's modify our test to make sure it works:

src/index.test.js (update)

javascript

```javascript
const { add } = require('./index');

test('adds 1 + 2 to equal 3', () => {
  expect(add(1, 2)).toBe(3);
});

test('adds 5 + 7 to equal 12', () => {
  expect(add(5, 7)).toBe(12);
});
```

Commit and push:

bash

```bash
git add src/index.test.js
git commit -m "Add another test"
git push
```

Check GitHub Actions again - you should see a new run with 2 passing tests.

Espacio Práctico

¿Qué conceptos recuerdas de esta sesión?

Cierre de Sesión

¿Qué se aprendió hoy?

- Características de herramientas de colaboración de equipo.

Conclusiones

La combinación de Git con herramientas de colaboración modernas permite a los equipos trabajar de manera eficiente, escalable y segura. Plataformas como GitHub o GitLab centralizan repositorios, CI/CD y gestión de tareas, mientras que integraciones con Slack, Jira o SonarQube cierran el ciclo de desarrollo. La elección depende de las necesidades específicas del equipo, pero el objetivo siempre es claro: **automatizar tareas repetitivas, mejorar la visibilidad del trabajo y fomentar la colaboración asíncrona**.

Recordatorio de cosas por hacer

- Estudiar el material de clase disponible en UTP+Class.

Gracias.