Programación Segura

Clase 6

Pamela Bonelli

8/11/2025

AIEP de la Universidad Andrés Bello

PRESENTACIÓN DE LA CLASE

Unidad 1:

Programación segura para desarrollo – OWASP

Aprendizaje esperado:

1.-Analizan ciclos de vida SDLC asociados al desarrollo de software, considerando normativa vigente.

Criterios de evaluación:

1.1.-Caracteriza modelos de desarrollo de software tradicionales y ágiles, considerando normativa vigente para desarrollo seguro.

1.2.-Identifica componentes de documentación y productos intermedios de trabajo en SDLC.

1.3.-Relaciona SDLC con calidad y seguridad del software, considerando normativa vigente ISO-9000 e ISO-27001.

1.4.-Relaciona SDLC con control de versiones, gestión de cambios y trazabilidad, considerando uso de herramienta GIT o similar.

PRESENTACIÓN DE LA CLASE

Contenidos:

- Modelos de desarrollo de software tradicionales y ágiles.
- Cascada, Prototipado, Incremental, Espiral.
- RAD, CMMI y AGILE.
- Normativa vigente para desarrollo seguro.
- Ciclos de vida de desarrollo de software: SDLC.
- Documentación y productos intermedios de trabajo en SDLC.
- ISO-9000 e ISO-27001.
- Repositorio de control de versiones.
- Gestión de cambios de ITIL.
- Concepto de Trazabilidad en SDLC.
- **Uso de herramienta GIT.**

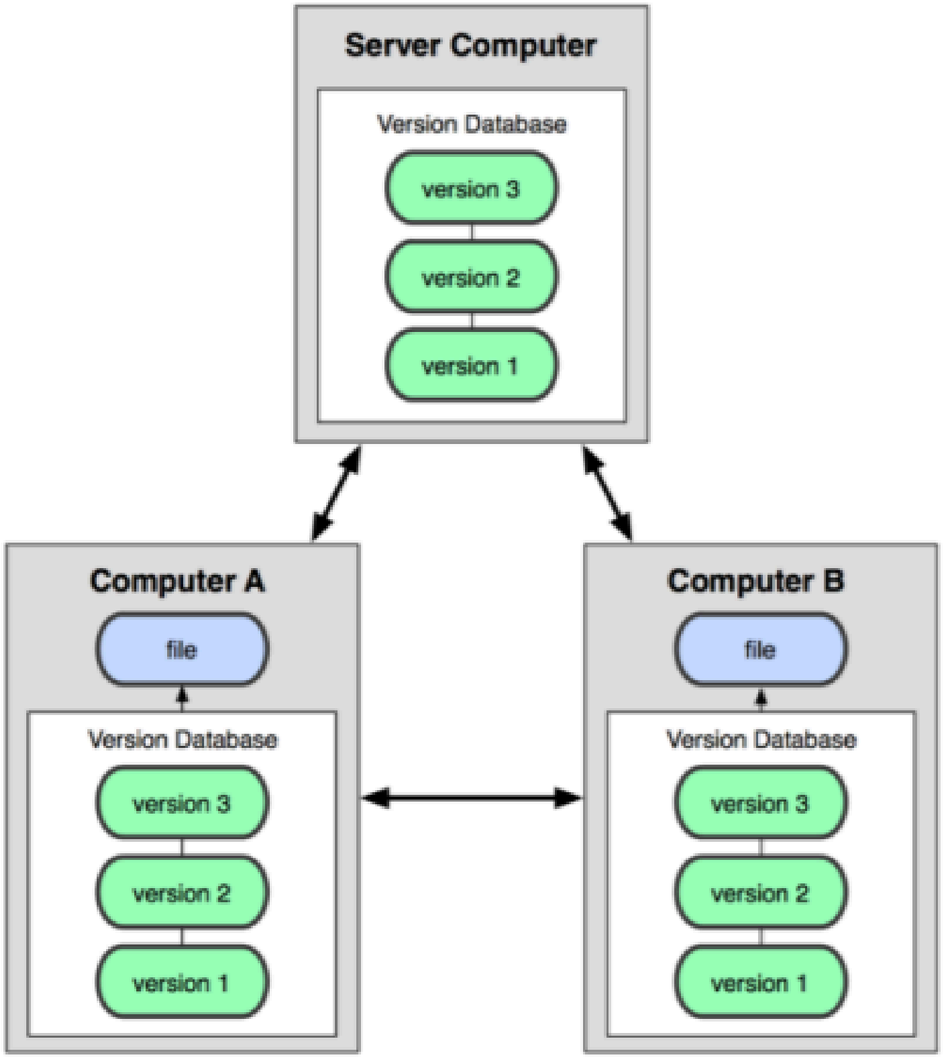
¿Qué es un sistema de control de versiones?

Un sistema de control de versiones (CVS) te permite realizar un seguimiento de la historia de una colección de archivos y además incluye la funcionalidad de revertir la colección de archivos actual hacia una versión anterior.

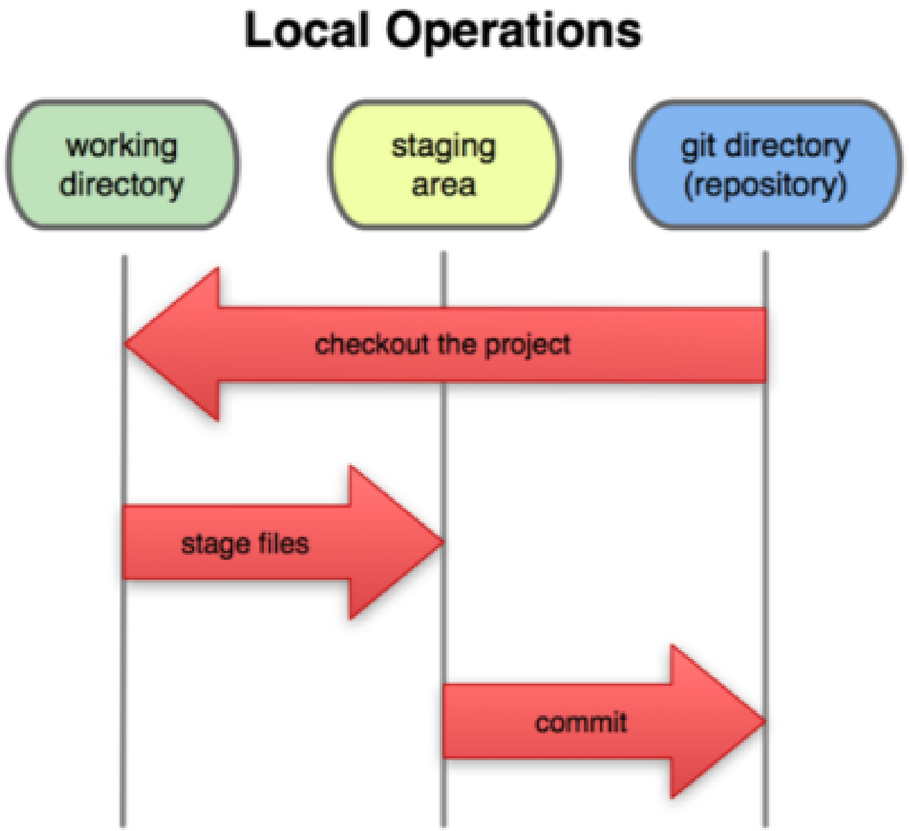
¿Qué es un sistema de control de versiones distribuida?

En un sistema de control de versiones distribuida hay un servidor central para almacenar el repositorio y cada usuario puede hacer una copia completa del repositorio central mediante un proceso llamado “clonación”.


git


Server Computer
Version Database
version 3
version 2
version 1
Computer A
file
Version Database
version 3
version 2
version 1
Computer B
file
Version Database
version 3
version 2
version 1

Repositorio Local



Flujo de trabajo básico en Git:

- Modificas una serie de archivos en tu directorio de trabajo (working directory).
- Añades instantáneas de los archivos a tu área de preparación (staging area).
- Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación, y almacena esa instantánea de manera permanente en tu directorio de Git (git directory).

Los tres estados

1. **Confirmado (committed):** los datos están almacenados de manera segura en tu base de datos local.

2. **Modificado (modified):** significa que has modificado el archivo, pero todavía no lo has confirmado a tu base de datos.

3. **Preparado (staged):** significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

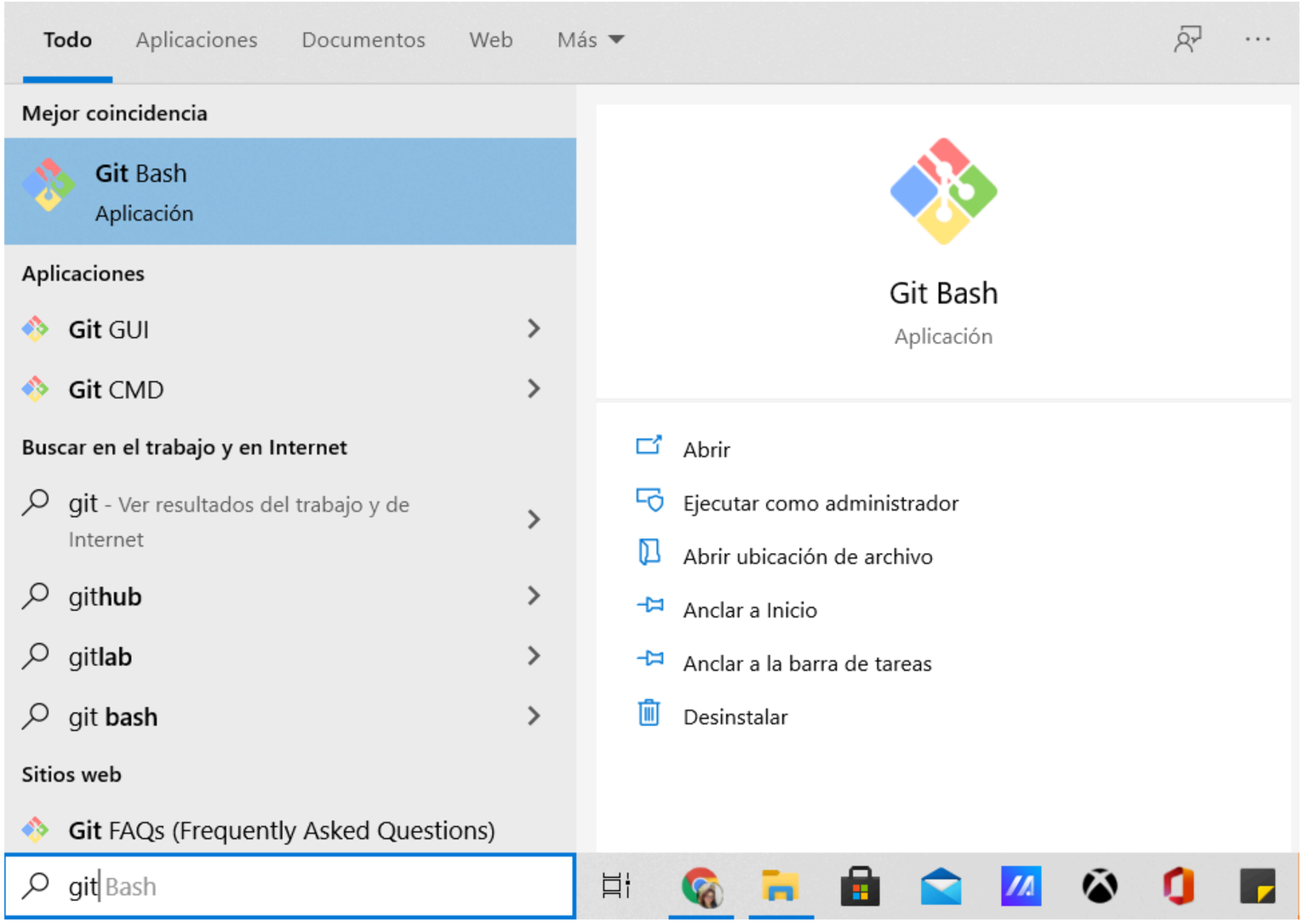
Momento para conocer

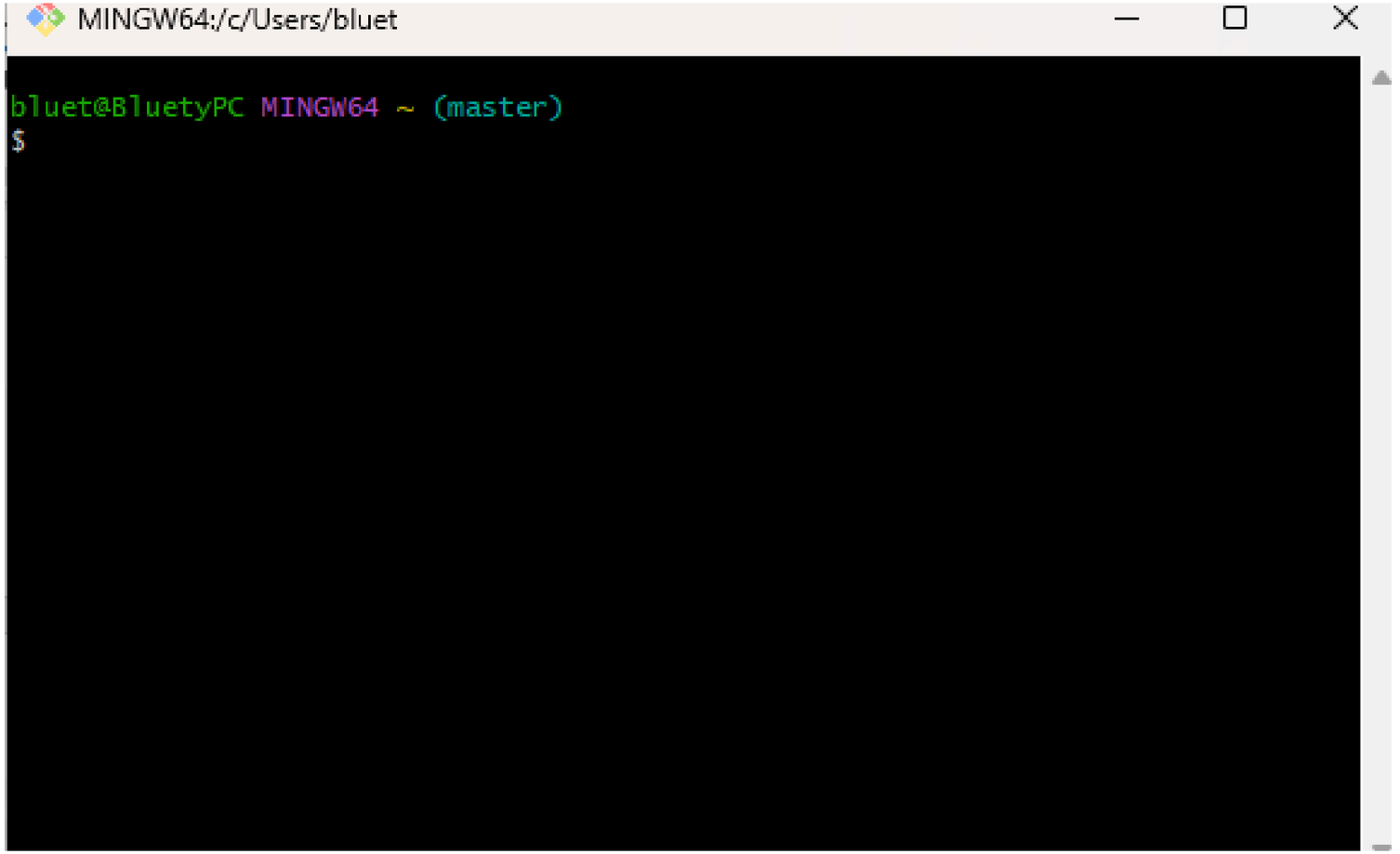
https://www.youtube.com/watch?v=Uw8SIaAK-vw

Descargar git

https://git-scm.com

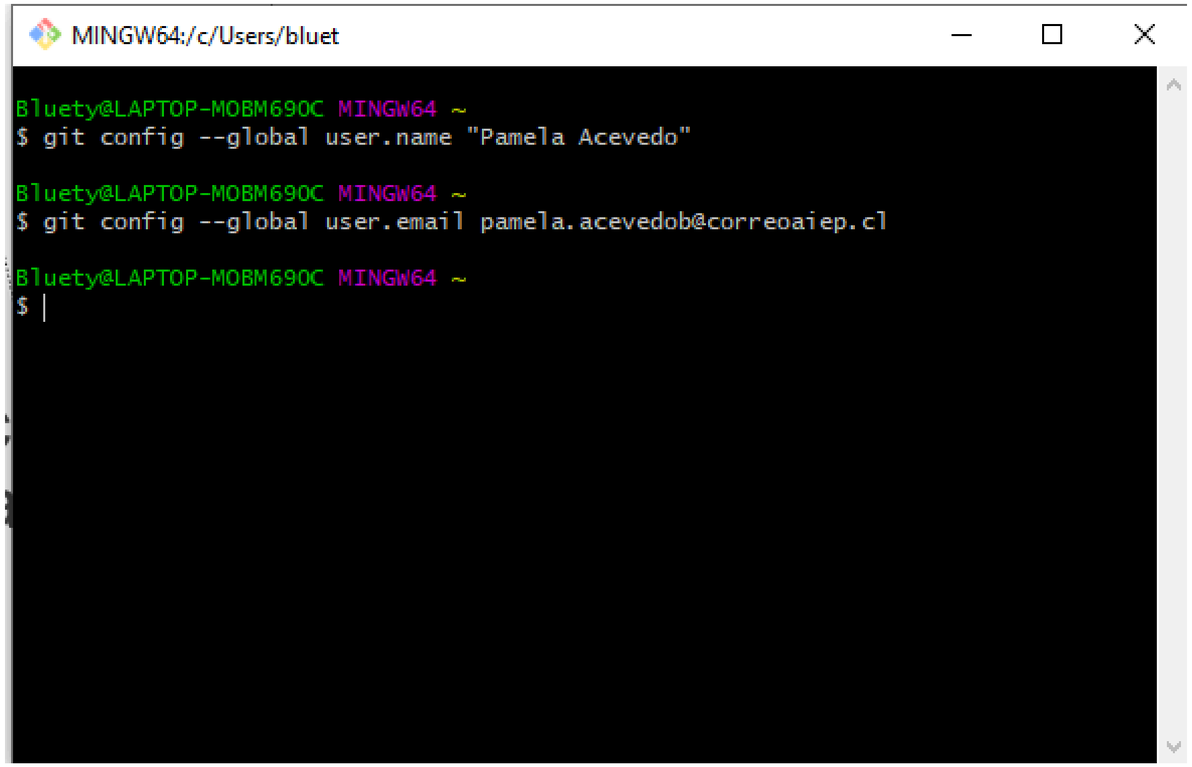
Momento para conocer




MINGW64:/c/Users/bluet
bluet@BluetyPC MINGW64 ~ (master)
$

1° paso Configurar nombre completo y correo

$ git config --global user.name “Pamela Bonelli"
$ git config --global user.email pb@correo.cl



Uso básico de Git
Crear un proyecto

1. Creamos un directorio donde agregar nuestro archivo de código

```
$ mkdir proyecto1

$ cd proyecto1
```

2. Creamos un archivo hola.html que muestre:
   Hola Mundo

Crear el repositorio

Para crear un nuevo repositorio se usa la orden git init

$ git init

Comprobar el estado del repositorio

Con la orden git status podemos ver en qué estado se encuentran los archivos de nuestro repositorio.

$ git status

Si modificamos el archivo hola.php:

```
<h1>Hola Mundo jsabsk</h1>
  <h2> Esto es prueba de git</h2>
```

Y volvemos a comprobar el estado del repositorio:

$ git status

```
$ git status
On branch 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:   hola.html

no changes added to commit (use "git add" and/or "git commit -a")
```

Añadir cambios

Con la orden git add indicamos a git que prepare los cambios para que sean almacenados.

git add hola.html

```
Bluety@LAPTOP-MOBM690C MINGW64 ~/proyecto1 (master)
$ git add hola.html

Bluety@LAPTOP-MOBM690C MINGW64 ~/proyecto1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hola.html
```

Confirmar los cambios

Con la orden git commit confirmamos los cambios definitivamente, lo que hace que se guarden permanentemente en nuestro repositorio.

git commit –m “Parametrización del programa”

```
$ git commit -m "Parametrización del programa"
[master 0f50a6e] Parametrización del programa
 1 file changed, 3 insertions(+), 2 deletions(-)

Bluety@LAPTOP-MOBM690C MINGW64 ~/proyecto1 (master)
$ git status
On branch master
nothing to commit, working tree clean

Bluety@LAPTOP-MOBM690C MINGW64 ~/proyecto1 (master)
```

Trabajando con el historial

Con la orden git log podemos ver todos los cambios que hemos hecho:

git log

```
Bluety@LAPTOP-MOBM690C MINGW64 ~/proyecto1 (master)
$ git log
commit 0f50a6e344eeed9d904aa896d15e934936589bc9 (HEAD -> master)
Author: Pamela Acevedo <pamela.acevedob@correoaiep.cl>
Date:   Sun Aug 22 22:37:11 2021 -0400

    Parametrización del programa

commit d47d5afb55b937882379cc544c40240a898aebd9
Author: Pamela Acevedo <pamela.acevedob@correoaiep.cl>
Date:   Sun Aug 22 22:23:19 2021 -0400

    Creación proyecto

Bluety@LAPTOP-MOBM690C MINGW64 ~/proyecto1 (master)
$
```

Momento para conocer

Aprender sobre Git

https://learngitbranching.js.org/

https://www.gitmastery.me/

AIEP
de la Universidad Andrés Bello

¡Muchas Gracias!