TECNOLOGÍA SUPERIOR EN TRANSFORMACIÓN DIGITAL DE EMPRESAS

PROGRAMACIÓN AVANZADA

RENÉ ROLANDO ELIZALDE SOLANO

rrelizalde@utpl.edu.ec
@reroes

Contenidos

2.1. Control de versiones

- 2.1.1. Fundamentos de control de versiones
- 2.1.2. Conceptos de Git
- 2.1.3. Comandos de Git
- 2.1.4. Creación de repositorios de versiones

Contenidos

2.1. Control de versiones

- 2.1.1. Fundamentos de control de versiones
- 2.1.2. Conceptos de Git
- 2.1.3. Comandos de Git
- 2.1.4. Creación de repositorios de versiones

https://git-scm.com/book/es/v2


sues are welcome in its GitHub repository.

1. Inicio - Sobre el Control de Versiones

1.1 Acerca del Control de Versiones
1.2 Una breve historia de Git
1.3 Fundamentos de Git
1.4 La Línea de Comandos
1.5 Instalación de Git
1.6 Configurando Git por primera vez
1.7 ¿Cómo obtener ayuda?
1.8 Resumen

2. Fundamentos de Git

2.1 Obteniendo un repositorio Git
2.2 Guardando cambios en el Repositorio
2.3 Ver el Historial de Confirmaciones
2.4 Deshacer Cosas
2.5 Trabajar con Remotos
2.6 Etiquetado
2.7 Alias de Git
2.8 Resumen

3. Ramificaciones en Git

3.1 ¿Qué es una rama?

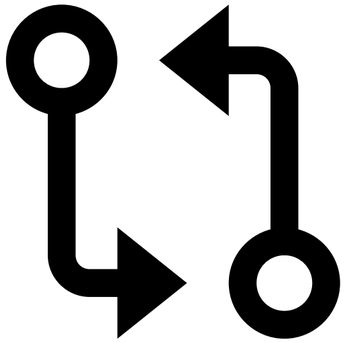
2nd Edition (2014)

Download Ebook

pdf epub

2.1.1. Fundamentos de control de versiones

En la actualidad, generar aplicaciones de código y no usar control de versiones, implica poner en riesgo la productividad del equipo o grupo de desarrollo.

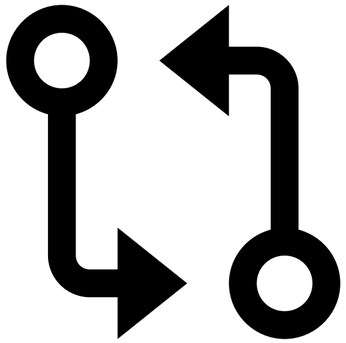


Un control de versiones, se convierte en un mecanismo o proceso que permite registrar en el tiempo el conjunto de cambios que se ha realizado para uno o varios archivos

Un control de versiones, permite que se pueda recuperar de manera exacta la versión en un día y hora específica.

2.1.1. Fundamentos de control de versiones

Ventajas

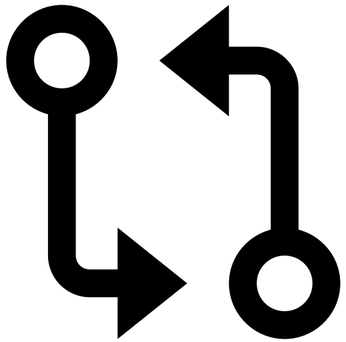


- Emplear versiones anteriores de los archivos.
- Ejecuta comparaciones de forma muy sencilla entre la versión actual de un archivo y una versión anterior.
- Identificar la persona(s) que modificó un archivo.

2.1.1. Fundamentos de control de versiones

Tipos

- **Sistema de control de versiones locales**
  - No son usados en la actualidad
  - Poseen una base de datos local, donde se llevaba el control
  - Ejemplos: RCS



GNU RCS

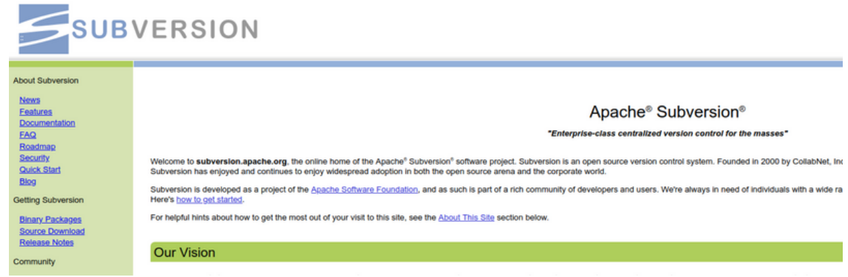
The Revision Control System (RCS) manages multiple revisions of files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, including source code, programs, documentation, graphics, papers, and form letters.

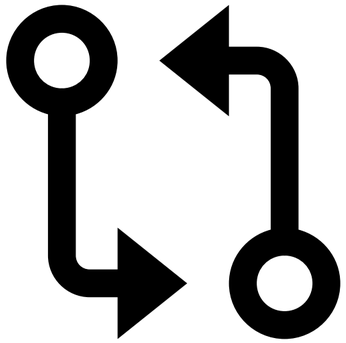
RCS was first developed by Walter F. Tichy at Purdue University in the early 1980s -- paper: RCS: A System for Version Control (1991) (troff, PostScript, PDF). See also the Purdue RCS Homepage.

RCS design is an improvement from its predecessor Source Code Control System (SCCS) (see GNU CSSC). The improvements include an easier user interface and improved storage of versions for faster retrieval. RCS improves performance by storing an entire copy of the most recent version and then stores *reverse* differences (called "deltas"). RCS uses GNU Diffutils to find the differences between versions.

2.1.1. Fundamentos de control de versiones

Tipos



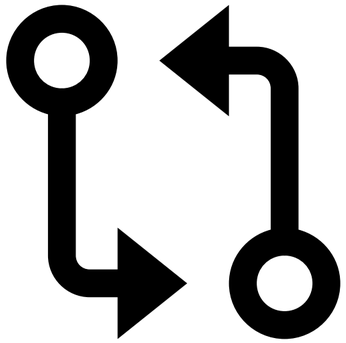


Su

- **Sistema de control de versiones centralizados**
  - Subversion / Perforce
  - Se posee colaboradores de un proyectos en varios lugares.
  - Existe un servidor único
  - El administrador, conoce a detalle el trabajo realizado y los niveles de acceso a cada parte del código.
  - Desventaja; pérdida total de la información, a causa del fallo del servidor único.

2.1.1. Fundamentos de control de versiones

Tipos



- **Sistema de control de versiones distribuidos**
  - Git / Mercurial / Bazaar
  - Cada colaborador del proyecto tiene la posibilidad de descargar en sus entornos locales la versión más reciente del proyecto.
  - Cada colaborador del proyecto hace una replica total del repositorio
  - En caso de algún problema se puede usar la réplica de unos de los colaboradores y restaurar todo.

2.1.2. Conceptos de Git

Git fue creado por Linus Torvalds, quien decidió efectuar una herramienta de control de versiones para manejar el código del núcleo de Linux en el 2005

- Velocidad.
- Diseño sencillo.
- Soporte.
- Distribuido.
- Manejo de grandes proyectos

2.1.2. Conceptos de Git

Git es un sistema que permite capturar el estado de todos los archivos del proyecto, al momento de realizar un cambio en cualquier archivo

Para Git, si un archivo no fue modificado, no es necesario generar una nueva instancia, si no usar la referencia que ya exista.

Git posee un mecanismo de integridad de información, para todos sus procesos - hash SHA-1

- 56745ee7053904d24b5a5ce7394b2bdac83c513e.

```
commit a0e2685d6bb4c9d86e9d4f8c8df6c96093b13b56 (HEAD -> main, origin/main, origin/HEAD
Author: René Elizalde <reroes799@gmail.com>
Date:   Wed Apr 19 08:21:37 2023 -0500

    ejemplos clases y objetos UML
```

2.1.2. Conceptos de Git

En Git existen tres estados posibles para los archivos de los proyectos

Confirmado (committed): en este estado los archivos con su información están almacenados de forma segura en la base de datos local.

Modificado (modified): indica que los archivos están modificados, pero aún no forman parte de la base de datos local.

Preparado (staged): cuando los archivos son marcados en su actual versión, dejando los mismos listos para la próxima confirmación.

2.1.2. Conceptos de Git

Un proyecto de Git formalmente tiene tres secciones

Directorio de Git (Git directory).

Directorio de trabajo (working directory).

Área de preparación (staging área).

```
ls -la .git

s 4096 abr 19 08:21 .
s 4096 abr 19 08:18 ..
s 4096 abr 19 08:18 branches
s   30 abr 19 08:21 COMMIT_EDITMSG
s  283 abr 19 08:18 config
s   73 abr 19 08:18 description
s   21 abr 19 08:18 HEAD
s 4096 abr 19 08:18 hooks
s 1179 abr 19 08:21 index
s 4096 abr 19 08:18 info
s 4096 abr 19 08:18 logs
s 4096 abr 19 08:21 objects
s  112 abr 19 08:18 packed-refs
s 4096 abr 19 08:18 refs
```

```
ls -la

4096 abr 19 08:18 .
4096 abr 19 08:18 ..
4096 abr 23 16:59 ejemplos-clases-uml
4096 feb 13 22:41 ejemplos-objetos-uml
4096 abr 19 08:21 .git
   9 abr 17  2022 README.md
```

2.1.2. Conceptos de Git

```
bimestre1/semana2/clase02$ git status
En la rama main
Tu rama está actualizada con 'origin/main'.

Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo)
        modificados:     ejemplos-clases-uml/ejemplo-clase-01.dia

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que se será confirmado)
        tmp/
```

Cuando se realizan modificaciones a los archivos en el directorio de trabajo

```
reroes@reroes-ubuntu:~/Dropbox/personal/clasesUTPL/CLASES_ABRIL_AGOSTO_2023/PROGRAMAC
bimestre1/semana2/clase02$ git status
En la rama main
Tu rama está actualizada con 'origin/main'.

Cambios a ser confirmados:
  (usa "git restore --staged <archivo>..." para sacar del área de stage)
        modificados:     ejemplos-clases-uml/ejemplo-clase-01.dia
        nuevos archivos: tmp/demo.html
```

En este paso se agregan los archivos al área de preparación.

```
bimestre1/semana2/clase02$ git commit -m"ejemplos"
[main 4feebe1] ejemplos
 2 files changed, 0 insertions(+), 0 deletions(-)
 rewrite ejemplos-clases-uml/ejemplo-clase-01.dia (100%)
 create mode 100644 tmp/demo.html
```

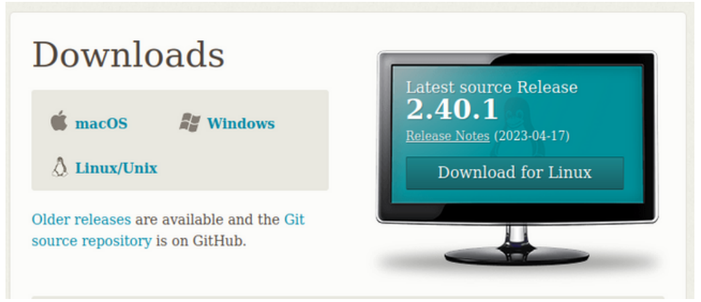
```
En la rama main
Tu rama está adelantada a 'origin/main' por 1 commit.
  (usa "git push" para publicar tus commits locales)

nada para hacer commit, el árbol de trabajo está limpio
```

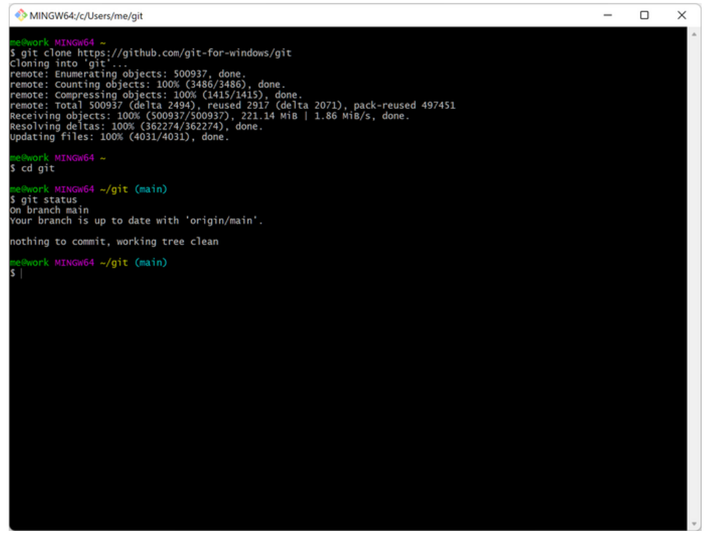
Se confirman los cambios

2.1.3. Comandos de Git

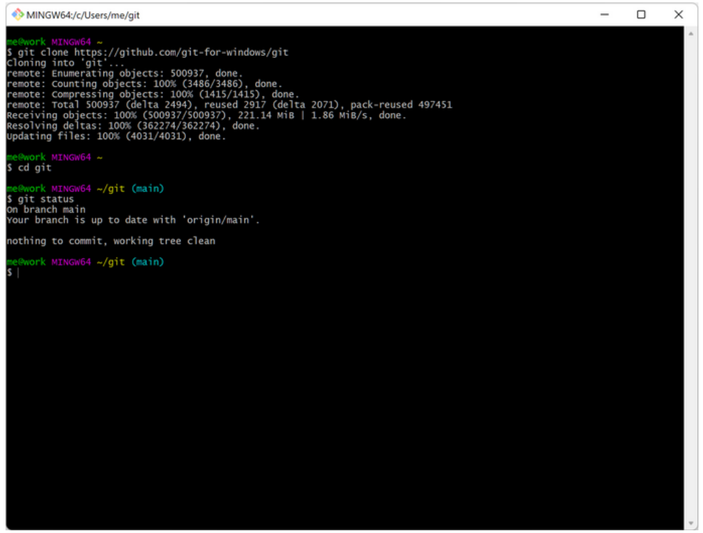
Como primer paso para poder trabajar en las computadoras personales con este sistema de control de versiones



En las etapas iniciales de formación académica, se recomienda el uso de la línea de comandos para trabajar con Git.

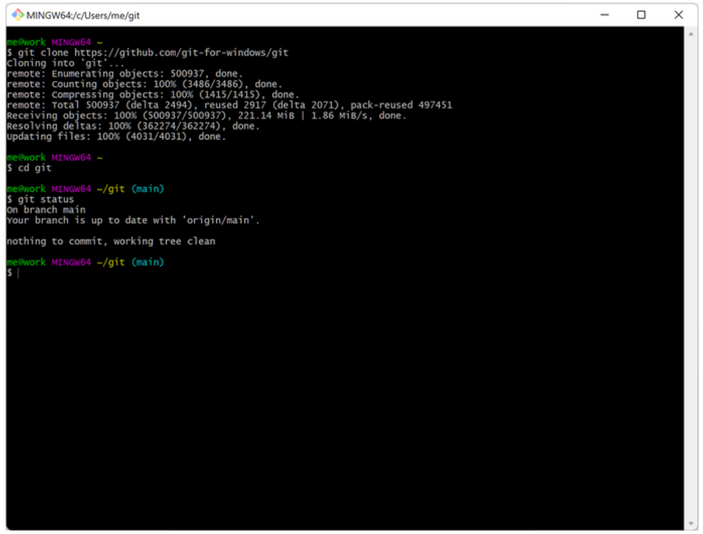


2.1.3. Comandos de Git



git config --global user.name “Nombre de Usuario”.
git config --global user.email nombre@ejemplo.com.

2.1.3. Comandos de Git



- git - -help.
- git config - -list .
- git status.
- git add.
- git comit.
- git push.
- git pull.
- git log.

2.1.4. Creación de repositorios de versiones

Opción 1:

ejemplo.htlm

run.py

git bash

git init

```
ayuda:  git branch -m <nombre>
Inicializado repositorio Git vacío en /home/reroes/Documentos/clasesaa2023/tec-a
a2023/demo/.git/
```

.git

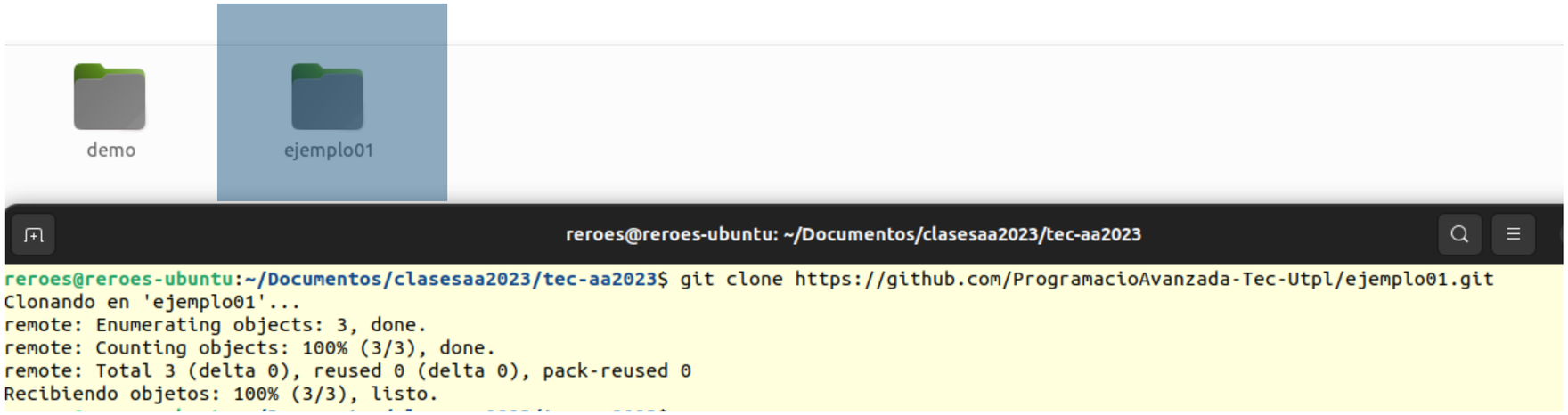
ejemplo.htlm

run.py

2.1.4. Creación de repositorios de versiones

Opción 2: git clone

git clone https://github.com/ProgramacioAvanzada-Tec-Utpl/ejemplo01.git



2.1.4. Creación de repositorios de versiones

Opción 2: git clone

git clone https://github.com/ProgramacioAvanzada-Tec-Utpl/ejemplo01.git **ejemploDemo**

2.1.4. Creación de repositorios de versiones

Uso de git add / git commit

- **git add .** Para agregar en el proceso de seguimiento a todos los nuevos archivos que son partes el directorio.
- **git add archivo.txt** Permite solo dar seguimiento al archivo específico que consta luego del comando git add.

git commit -m ‘Cadena informativa de la confirmación’

Gracias

rrelizalde@utpl.edu.ec
@reroes