



DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN  
ESCUELA DE INGENIERÍA  
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

---

# Manejo de Git

---

## 1. Introducción

¿Nunca has escuchado de Git? ¿Lo usaste pero no recuerdas nada de él? Este documento intentará ayudarte a resolver dudas básicas del sistema con el que se trabajarás durante el semestre.

**Nota:** A lo largo de este documento se usa la sigla de curso IIC2343 (Arquitectura de Computadores), dado que fué tomado de una versión antigua del mismo, pero esta información es irrelevante en cuanto al manejo mismo de GIT.

## 2. *Git*

Git es un sistema distribuido de control de versión, gratuito y open source, diseñado para manejar de pequeños a enormes proyectos de forma rápida y eficiente.

En otras palabras, permite un manejo de distintas versiones de un proyecto, manteniendo registro de las distintas etapas que ha tenido éste. Cada una de éstas etapas deben ser demarcadas manualmente, y se les llama **commits** (algo así como *checkpoints*). Gracias a esto entonces, podemos ver como ha ido evolucionando un proyecto, y a la vez tenemos la posibilidad de volver a versiones (commits) anteriores de nuestro proyecto en caso de que se necesite revisar/empezar de nuevo cierta parte.

Además, facilita el trabajo en equipo, permitiendo que dos personas trabajen en un mismo proyecto, sin editar directamente el código que está realizando la otra persona (en este curso no necesitaran hacer esto, pero vale la pena mencionarlo).

Algunas ventajas de el uso de Git son:

1. Trabajo en equipo fluido

2. Versiones disponibles en cualquier momento
3. Control de cambios efectuados a lo largo del proyecto
4. Fácil programación en paralelo, y posterior unión (*merge*) de ambas partes
5. Múltiples *backups* del proyecto

Git se basa en la línea de comandos (terminal, cmd, etc...) para realizar sus instrucciones. Primero que nada, deben tener **Git instalado** en sus computadores, pueden revisar si esta instalado corriendo la siguiente instrucción de su terminal:

```
Alfonso Irarrázaval :: ~ » git --version
git version 2.20.1.windows.1
```

Una vez ya instalado, podemos empezar a manejar nuestro repositorio de GitHub con Git a través de nuestra terminal, pero antes debemos introducir:

## 2.1. GitHub & Git

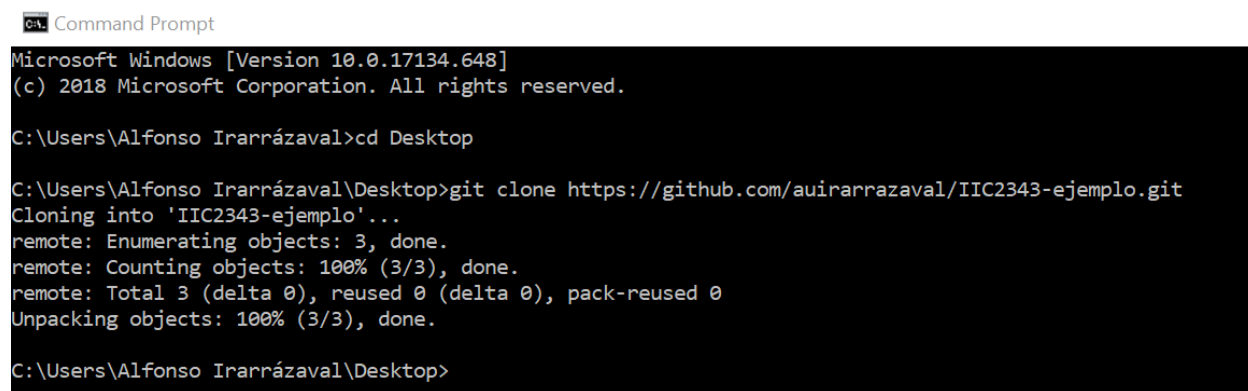
Es una plataforma para alojar proyectos, usando el sistema de control de versiones Git. GitHub nos simplifica la vida entregando una interfaz gráfica donde ver nuestros cambios y distintos estados de nuestro proyecto. Para ejemplificar como funciona GitHub y Git, vamos a partir con un repositorio vacío, como el que recibirá cada uno al inicio del curso. Al ingresar al link que les entregaremos, se encontraran con algo similar a lo siguiente:

The screenshot shows a GitHub repository page for 'IIC2343-ejemplo' by user 'airarrazaval'. The repository is empty, with 1 commit, 1 branch, 0 releases, and 1 contributor. The 'README.md' file is listed, and its content is displayed below. Red arrows point to the repository name, the 'README.md' file, the commit message 'Initial commit', and the content of the README file, which is 'IIC2343-ejemplo'.

Lo primero que debemos hacer, es **clonar** (crear una versión local de) nuestro repositorio en nuestro computador, para esto, primero debemos ir al botón verde en el repositorio, dónde dice *clone or download*, y se desplegara un menú con el link de nuestro repositorio (el mismo que recibieron por parte nuestra) el cual hay que copiar. Luego deben abrir su terminal, ir a la carpeta dónde quieren que esté su versión local del repositorio y ejecutar el comando:

**git clone < link >**

En éste caso, quería que el repositorio estuviera en Desktop, por lo que ejecuté *cd Desktop* para ir a esa carpeta (*cd* → Change Directory), y luego ejecuté el comando desde esa carpeta. se ve algo así:



```
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Alfonso Irarrázaval>cd Desktop

C:\Users\Alfonso Irarrázaval\Desktop>git clone https://github.com/auirarrazaval/IIC2343-ejemplo.git
Cloning into 'IIC2343-ejemplo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.

C:\Users\Alfonso Irarrázaval\Desktop>
```

Podemos ver que descargo el repositorio y todos sus contenidos, y ahora existe una carpeta en ese directorio con el nombre de nuestro repositorio, si accedemos a ella (*cd < nombre repositorio >*, o bien a través del explorador) nos podemos dar cuenta que está el README.md que tenía en un inicio.

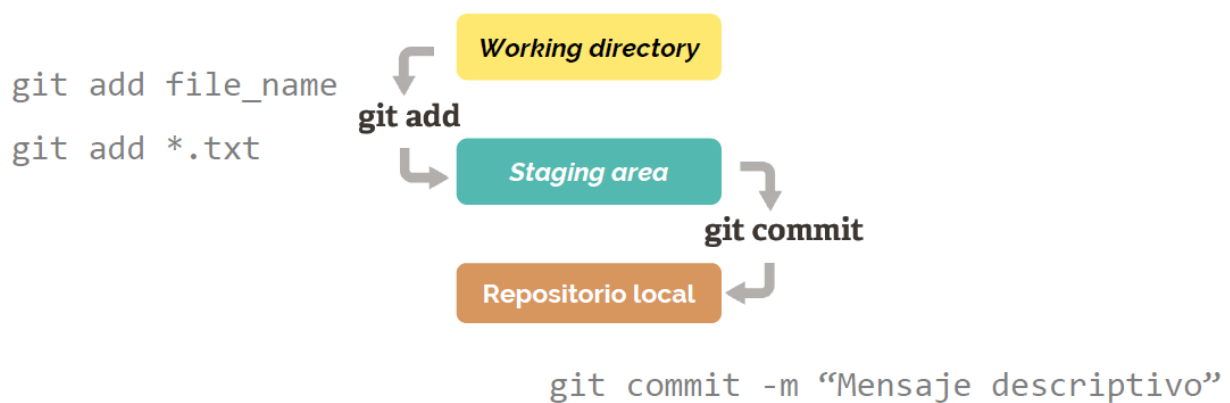
Ahora es necesario hacer una diferencia entre repositorio **local** y uno **remoto**. El local es el que vemos en nuestro computador y editamos directamente. El repositorio remoto corresponde al que está 'online' en GitHub. Lo que necesitamos saber nosotros es, en resumen, como coordinar correctamente estos dos repositorios.

Lo que queremos hacer ahora, es crear un archivo y subirlo a GitHub, para lograr eso crearemos un simple .txt (puede ser cualquier tipo: carpetas, .py, .js, .vhd, etc...) al que le pondremos ejemplo.txt y escribiremos un texto al azar en él. Este archivo en este minuto se encuentra en nuestra carpeta, pero aún no en GitHub. para agregarlo a GitHub hay que correr los siguientes comandos desde la terminal en el directorio correspondiente:

1. **git add ejemplo.txt** → Agrega nuestro archivo a la **Staging Area**, la cual contiene todos los archivos a los que se les desea hacer *commit*, pueden ser múltiples archivos separados por un espacio, o si se quiere subir todo el directorio, se usa "git add ."

2. (opcional) **git status** → Nos mostrara un resumen de los cambios que se han efectuado en este commit, en este caso, que se agrego el archivo *ejemplo.txt*
3. **git commit -m "nombre commit"** → Este comando se usa para marcar un *checkpoint* en nuestro proyecto, el cual tendrá como nombre/descripción el texto puesto dentro de las doble comillas.

\*\* (Hasta aquí, tenemos agregados los cambios en nuestro directorio **local**, pero si nos fijamos, en GitHub aún no se refleja ningún cambio)



\*\* (Ahora queremos que nuestros cambios se ejecuten en el repositorio **remoto** (GitHub))

4. **git push** → Este comando toma todos los commits pendientes (pueden ser mas de uno) y los *empuja* desde el servidor local al remoto (probablemente pedirá las credenciales username - password de GitHub)

```

C:\Users\Alfonso Irarrázaval\Desktop>cd IIC2343-ejemplo

C:\Users\Alfonso Irarrázaval\Desktop\IIC2343-ejemplo>git add ejemplo.txt

C:\Users\Alfonso Irarrázaval\Desktop\IIC2343-ejemplo>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   ejemplo.txt

C:\Users\Alfonso Irarrázaval\Desktop\IIC2343-ejemplo>git commit -m"example commit"
[master 9226e39] example commit
 1 file changed, 1 insertion(+)
 create mode 100644 ejemplo.txt

C:\Users\Alfonso Irarrázaval\Desktop\IIC2343-ejemplo>git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 321 bytes | 107.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/auirarrazaval/IIC2343-ejemplo.git
 d66633a..9226e39  master -> master

C:\Users\Alfonso Irarrázaval\Desktop\IIC2343-ejemplo>

```

Ahora, si vamos a GitHub, podremos ver que está nuestro archivo ejemplo.txt con su contenido. También podemos ver que sale un "ID" de el commit reciente.

[auirarrazaval / IIC2343-ejemplo](#) Watch 0 Star 0 Fork 0

[Code](#) [Issues 0](#) [Pull requests 0](#) [Projects 0](#) [Wiki](#) [Insights](#) [Settings](#)

No description, website, or topics provided. [Edit](#)

[Manage topics](#)

[2 commits](#) [1 branch](#) [0 releases](#) [1 contributor](#)

Branch: master [New pull request](#) [Create new file](#) [Upload files](#) [Find File](#) [Clone or download](#)

File	Commit	Message	Time
README.md	Initial commit		an hour ago
ejemplo.txt	example commit		6 minutes ago

[README.md](#)

# IIC2343-ejemplo

Si hacemos click en el nombre del commit, podemos ver todas las lineas de los archivos que cambiaron en dicho commit. (Verde es linea agregada, Rojo quitada)

Ya sabemos entonces como llevar nuestro repositorio local al remoto (git push), pero hay casos en los que será necesario llevar contenidos nuevos disponibles en el repositorio remoto a el local (Cuando se suban enunciados en la carpeta Enunciados, por ejemplo). Para esto se hace el opuesto a git push, **git pull**, este comando "descargará" todo nuevo archivo del repositorio remoto al local.

### Observaciones Importantes:

1. Si se necesita hacer pull mientras se está trabajando en un proyecto, hay que hacer commit de **todos archivos cambiados** para evitar problemas de consistencia.
2. Al momento de revisar una tarea, se tomará en cuenta el **último commit** antes de la hora de entrega, a no ser que se especifique otro commit en el README
3. Pueden practicar y aprender de cero haciendo [click aquí](#).

*\*\* Agradecimientos al material de Programación Avanzada 2018-1 por algunas de las imágenes y a los difuntos ayudantes de Arquitectura de computadores 2019-1 por hacer la versión 1.0*

## Política de Integridad Académica

Los alumnos de la Escuela de Ingeniería deben mantener un comportamiento acorde al Código de Honor de la Universidad:

*“Como miembro de la comunidad de la Pontificia Universidad Católica de Chile me comprometo a respetar los principios y normativas que la rigen. Asimismo, prometo actuar con rectitud y honestidad en las relaciones con los demás integrantes de la comunidad y en la realización de todo trabajo, particularmente en aquellas actividades vinculadas a la docencia, el aprendizaje y la creación, difusión y transferencia del conocimiento. Además, velaré por la integridad de las personas y cuidaré los bienes de la Universidad.”*

En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un procedimiento sumario. Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno (grupo) para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno (grupo), sin apoyo en material de terceros. Por “trabajo” se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno (grupo) copia un trabajo, los antecedentes serán enviados a la Dirección de Docencia de la Escuela de Ingeniería para evaluar posteriores sanciones en conjunto con la Universidad, las que pueden incluir reprobación del curso y un procedimiento sumario. Por “copia” se entiende incluir en el trabajo presentado como propio partes hechas por otra persona. Está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la cita correspondiente.