# Aux 1

## Github

Prof. de Cátedra: Pablo Estévez
Prof. Auxiliar: Ignacio Reyes
Ayudantes: Germán García
Esteban Reyes
Nicolás Tapia

Este documento pretende ser una guía para la utilización de GitHub, herramienta que consideramos esencial para mantener y llevar un registro del código que realicen a lo largo de sus vidas. Esta herramienta no sólo permite almacenar proyectos de programación y tener un historial de sus modificaciones, sino que Git es un estándar al momento de crear proyecto en equipo. Muchas veces, en entrevistas laborales, se pide como 'CV' de sus habilidades de programación.

Aquí se presentan las herramientas básicas para crear un repositorio y trabajar en equipo sobre él, entregando un *Cheat Sheet* con una serie de comandos básicos.

La idea es que utilicen GitHub para realizar sus proyectos del curso, y nosotros podamos llevar registro de su forma de trabajo.

## 1. Crear una Cuenta Educativa en GitHub

Regístrense en Github con su correo @ing.uchile.cl o @ug.uchile.cl (**Es necesario usar el correo institucional**). Si ya están registrados, agreguen el correo institucional como uno de sus correos de cuenta (Para saber como hacer esto ver tutorial)

Después, han de solicitar el pack educativo de Github aquí. Con esto podrán crear repositorios privados para realizar sus proyectos.

Al postular postular **TIENEN** que usar su correo institucional (@ing.uchile.cl o @ug.uchile.cl). Para llenar el resto de los datos, pueden responder en 'Cómo planeas usar Github?' diciendo 'I need it for university projects'.

Cualquier duda que tengan publíquenla en el foro o contacten al cuerpo docente.

## 2. Instalar Git

### 2.1. Linux [Ubuntu]

```
sudo apt-get install git
```

### 2.2. Windows

- Descargar Git para Windows desde https://git-for-windows.github.io/
- Instalar fichero anterior, con opciones predeterminadas.

## 3. Crear un Repositorio

Los siguientes pasos son para todos los sistemas operativos.

- En una terminal, asociar su cuenta de GitHub a su computador:

```
git config --global user.name "Mi Nombre"
git config --global user.email "mi@correo.cl"
```

- Ir a https://www.github.com e iniciar sesión.
- Hacer click en '*New Repository*' (ubicado en barra lateral, sobre, '*Your Repositories*').
- En *repository name* escribir *EjemploGit-EL4106* y marcar repositorio como privado. Presionar '*Create Repository*'.
- Crear una carpeta donde se ubicará el proyecto. En dicha carpeta crear un archivo con nombre `.gitignore`. Este archivo sirve para que Git no suba algunos archivos al repositorio central. Por ejemplo se podría poner los nombres de las bases de datos, o archivos de compilación de python como los con terminación `*.pyc`.
- Como se indica en la página de GitHub, abrir una terminal en la carpeta donde se ubicará el proyecto y ejecutar:

```
1 echo "# EjemploGit-EL4106" >> README.md
2 git init
3 git add README.md .gitignore
4 git commit -m "first commit"
5 git remote add origin https://github.com/Nombre_Usuario_Github/Proyecto-EL4106.git
6 git push -u origin master
```

Los comandos anteriores, en orden de ejecución, significan:

1. Se crea un `Readme`.
2. Inicia Git en el directorio actual.
3. Agrega al paquete (archivo) a enviar al repositorio.
4. Sella y cierra todos los cambios realizados con el mensaje `'first commit'`, dejándolo listo para ser enviado al repositorio.
5. Se agrega repositorio de GitHub como destino de los paquetes de código.
6. Envía el paquete al repositorio de GitHub, actualizado su contenido en la página web.

## 4. Manejo Básico de un Repositorio

Con su repositorio creado, para que cada cambio que decidan realizar a su código se vea reflejado en su repositorio de GitHub, deben ejecutar lo siguiente:

```
1 git add -A
2 git commit -m "Mensaje explicativo de cambios realizados"
3 git push origin master
```

Los comandos anteriores, en orden de ejecución, significan:

1. Agregar todos los archivos que se han modificado hasta el momento.
2. Etiqueta o nombre del paquete que será subido a GitHub, generando un historial que identifique los cambios realizados.
3. Subir las modificaciones al `master` en el repositorio de GitHub.

Si desean hacer una copia de su repositorio en otro ordenador, han de ejecutar en terminal:

```
git clone https://github.com/Nombre_Usuario_Github/Proyecto-EL4106.git
```

Como es probable que su compañer@ realice modificaciones al código, actualizando el repositorio en GitHub sin que usted lo sepa, **ES IMPORTANTE** que **siempre antes** de trabajar, usted realice una actualización de su repositorio local al hacer un `pull` del repositorio de GitHub, al ejecutar:

```
git pull
```

En caso de que usted no realice un `pull` y modifique el repositorio, existirán conflictos entre su repositorio local y el de GitHub, lo que le impedirá realizar `push` de sus modificaciones, hasta que resuelva sus conflictos o realice una nueva copia del repositorio con `git clone` del repositorio e introduzca sus modificaciones ahí.

# 5. Agregar Colaboradores a Repositorio

**Para los proyectos se solicitará como entregable un repositorio de GitHub con los códigos finales.** Debido a que los proyectos son en parejas, es necesario que algun@ de ustedes cree un repositorio privado con nombre **Proyecto-EL4106** y luego inviten a su compañer@ como colaboradora o colaborador del repositorio. También han de agregar al equipo docente para poder ver su progreso a lo largo del semestre. Para esto, sigan los siguientes pasos:

1. Acceder al url de su repositorio *Proyecto-EL4106* (La url es algo como `https://www.github.com/nombre_de_usuario/Proyecto-EL4106`).
2. Clickear en *Settings*, después en *Collaborators*.
3. Buscar por nombre de usuario a su compañero y aceptar.
4. Repetir los pasos anteriores, para agregar al cuerpo docente con nombre de usuario EL4106EquipoDocente2019

# 6. Manejo de Ramas

En Git existe una herramienta muy útil al momento de tener muchas personas trabajando sobre un mismo proyecto, o simplemente si se desea mantener la integridad de un proyecto. Estas son las ramas o `branches`, que permiten crear *'copias'* o *'versiones'* de un proyecto, que normalmente tiene como rama principal la rama `master`. La idea es que siempre que se quiera probar ideas o realizar modificaciones a un proyecto principal, se cree una rama con una copia de este, se realicen los cambios, y una vez que se esta segur@ de que los cambios han sido realizados tal y como se quiere y de forma correcta, esta rama se junte o haga `merge` con la rama `master`. De este modo no se compromete la integridad del proyecto, y un@ se asegura de modificar la rama `master` de forma consciente.

A continuación se ejemplifica el proceso de crear una rama, modificar su contenido y luego hacerle `push` a la rama `master`.

## 6.1. Crear y Acceder a Rama

```
1 git checkout -b nombre_de_rama
2 git checkout nombre_de_rama
```

En orden de ejecución, el código anterior realiza:

1. Se crea una rama llamada `nombre de rama`, cuyo contenido será una copia de los archivos de la rama actual.

2. Se cambia el directorio de trabajo a la rama `nombre_de_rama`, es decir, las modificaciones que se realicen en la carpeta del repositorio sólo quedarán registradas en la rama `nombre_de_rama`.

Si se desea saber en que rama se está trabajando, ejecutar:

```
git branch
```

### 6.2. Modificar y hacer `push` en Rama

Al igual que en la sección 4, se ejecuta:

```
1 git add -A
2 git commit -m "Mensaje explicativo de cambios realizados"
3 git push origin nombre_de_rama
```

Pero esta vez, el push se realiza sobre la rama *nombre_de_rama*

### 6.3. Hacer `merge` con `master`

Una vez se esta segur@ de que las modificaciones en la rama funcionan apropiadamente y desean ser juntadas con la rama `master`, ejecutar:

```
git checkout master
git merge nombre_de_rama
git push origin master
```

Si es que existen conflictos entre master y la rama que se esta haciendo `merge`, estos han de resolverse manualmente.

## 7. Recomendaciones

- No subir archivos pesados o bases de datos a su repositorio, si un archivo excede los 50MB no será subido a GitHub, y si son subidos, cada vez que clonen el repositorio tomará mucho tiempo. Si se tienen archivos pesados o bases de datos en carpetas dentro del repositorio, es recomendable agregarlas al `.gitignore` (se pueden ignorar carpetas completas).
- Evitar subir resultados como gráficos o imágenes a un repositorio, ya que pesan mucho. Si se tienen en carpetas dentro de este, agregarlas al `.gitignore`.
- Ejecutar `git status` para saber el estado actual de un repositorio y poder saber qué archivos serán modificados o subidos al repositorio de GitHub.
- Ejecutar `git diff` para visualizar las modificaciones realizadas al repositorio local.

Agradecimientos al Cuerpo Docente del curso

CC3002-1 Metodologías de Diseño y Programación 2017, Primavera,

Ya que esta guía es una adaptación de sus Auxiliares 1 y 2