



Actividad 00

Parte 1: Bienvenid@ a IIC2233

Bienvenid@ a tu primera actividad en IIC2233. Esta actividad **no será evaluada**. Nos servirá de práctica para prepararnos para la actividad del jueves siguiente.

Nota: Si seguiste los pasos mostrados por tu profesor/a en Zoom, deberías tener lista la Parte 1 de la actividad, por lo que puedes avanzar hasta la **Parte 2.5**.

El objetivo de esta primera parte es entender cuáles son las principales páginas relacionadas con el curso. Los requisitos para empezar son:

1. Haber instalado Git en tu computador (más info en [este enlace](#)). Si tienes macOS o Linux, muy probablemente ya lo tendrás instalado.
2. Haberte registrado en GitHub y en el curso en el formulario enviado ([este formulario](#)).
3. Una vez registrado, debió llegarte un correo de invitación de GitHub con una invitación a un repositorio en la organización del curso en GitHub (revisa bien tu correo).

1.1 *Syllabus* (syllabus)

El *syllabus* es un **repositorio** de GitHub donde, como equipo docente, subiremos todos los enunciados de las tareas, actividades y ayudantías. Puedes ver el *syllabus* en github.com/IIC2233/syllabus.

El *syllabus* tiene asociado un foro de *issues*. En este foro puedes preguntar sobre la materia o algo sobre los enunciados de las tareas. También los ayudantes podrán colocar avisos importantes, ya sean detalles administrativos o aclaraciones de enunciados, **por lo que es tu deber prestar atención constante al contenido de este foro**¹.

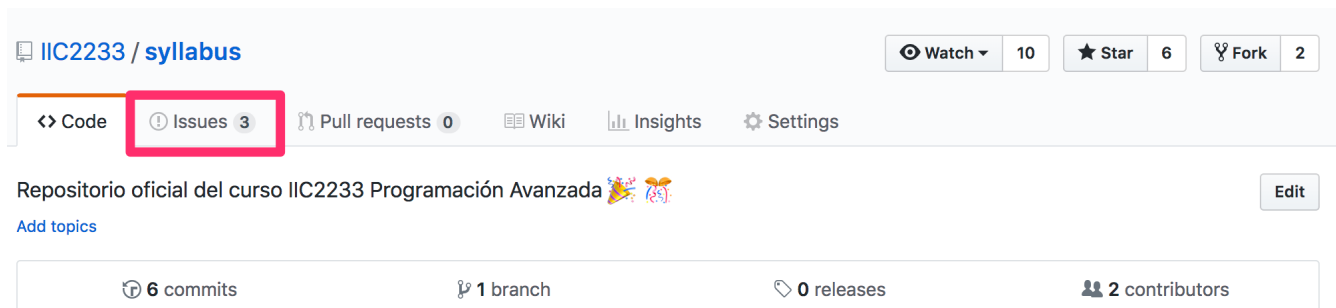


Figura 1: Llegar al foro a través del *syllabus*.

¹Existe un canal de [Telegram](#) (un cliente de mensajería, mejor que WhatsApp, que utilizamos mucho en el DCC) que avisa cuando una *issue* es marcada como importante. El canal es [Avanzada Channel 2020-1](#) y existe gracias a Enzo Tamburini.

1.2. Repositorio de apuntes (contenidos)

Los apuntes con los contenidos del curso se encontrarán en un repositorio llamado **contenidos**, al que puedes llegar haciendo clic en un vínculo ubicado en la página principal del curso, o yendo directamente a github.com/IIC2233/contenidos.

1.3. Tu repositorio personal

Luego de registrarte en el curso, se te ha creado un repositorio **personal y privado** donde deberás trabajar y entregar tus actividades y tareas. Este se ubica en github.com/IIC2233/usuario-iic2233-2020-2, donde debes reemplazar **usuario** por tu usuario de GitHub.

Tu repositorio también tiene asociado un foro de *issues*. En él recibirás —en forma **automatizada**— el detalle de la corrección de tus actividades y tareas. **NO respondas los mensajes ahí, puesto que nadie leerá lo que escribas**. Si tienes dudas sobre tu corrección usa el correo de los ayudantes o solicita una recorreción mediante formulario.

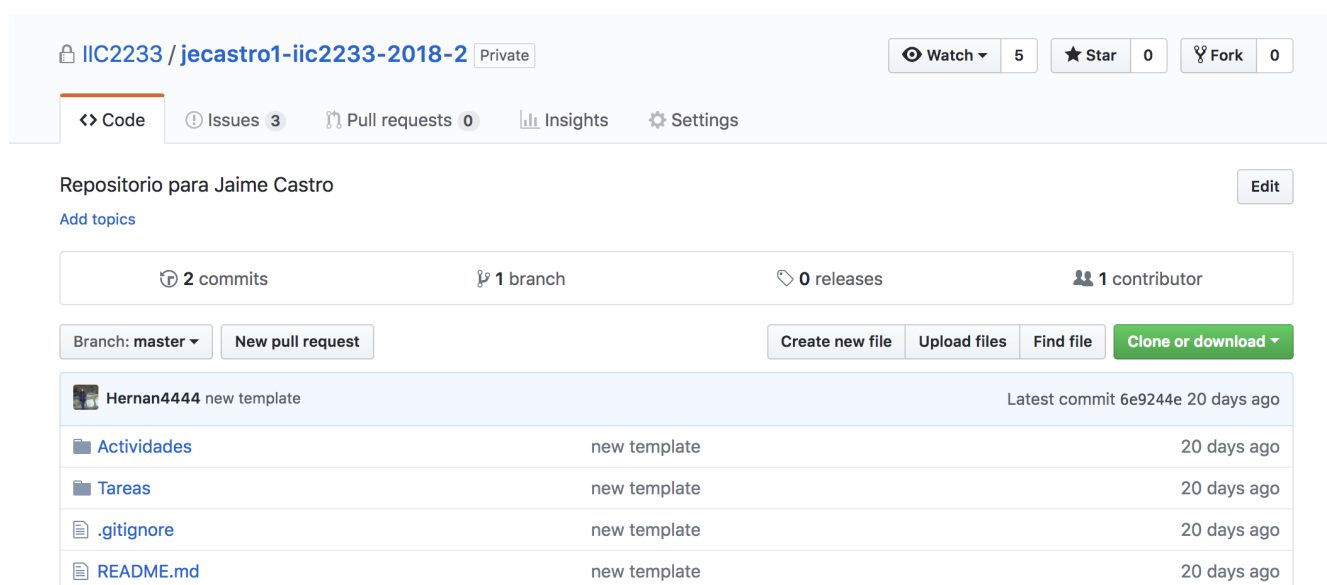


Figura 2: Ejemplo de repositorio personal (de un semestre anterior, adivinen cuál).

Parte 2: Ahora sí, bienvenid@ a IIC2233

El objetivo de esta parte es tener un primer contacto con **git**, el sistema de control de versiones que se utiliza en este curso.

Aprender lo básico de la consola

Necesitamos aprender a navegar por las carpetas del computador usando la consola. En esta parte de la guía, sigue todos los pasos y verifica que te sale lo mismo que está aquí.

Para abrir la consola:

- **macOS o Linux:** busca el programa **Terminal** o similar.
- **Windows:** busca el programa **Git Bash**. Este programa es una consola que además implementa algunos de los comandos que podrías encontrar en Linux, por lo que es mucho más amigable que el “Símbolo del Sistema”.

🔒 IIC2233 / jecastro1-iic2233-2018-2 Private

👁 Watch 5 ★ Star 0 🔗 Fork 0

🔗 Code 📄 Issues 3 📄 Pull requests 0 📊 Insights ⚙ Settings

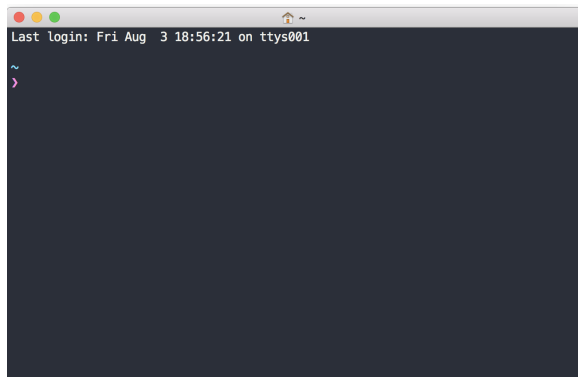
Filters 🔽 🔍 is:issue is:open Labels Milestones New issue

<input type="checkbox"/>	📌 3 Open ✓ 0 Closed	Author ▾	Labels ▾	Projects ▾	Milestones ▾	Assignee ▾	Sort ▾
<input type="checkbox"/>	📌 AC02 #3 opened 3 days ago by Hernan4444						
<input type="checkbox"/>	📌 AC01 #2 opened 3 days ago by Hernan4444						
<input type="checkbox"/>	📌 AC00 #1 opened 3 days ago by Hernan4444						

💡 ProTip! Check team mentions with [team:IIC2233/profesores](#).

Figura 3: *Issues* del repositorio personal.

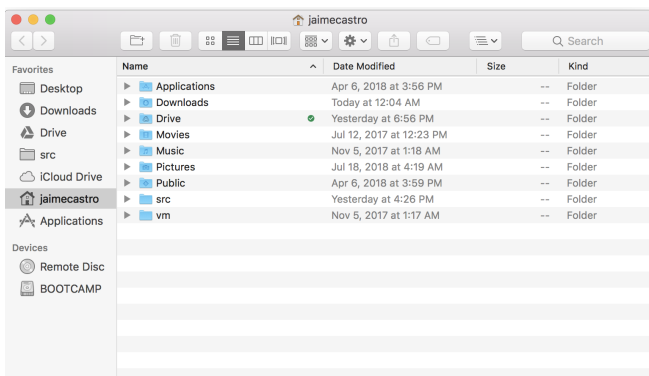
La consola en todo momento está ubicada en una carpeta (o directorio). Al abrir la consola, ésta se abre en la carpeta de tu computador que contiene tus datos. Esta carpeta suele llamarse *home*, ó *~* ó “*~*”. Dentro de ella, suele encontrarse el escritorio o la carpeta de documentos.



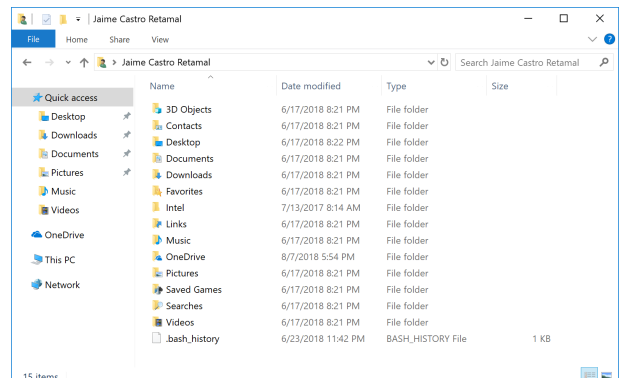
(a) Consola abierta en macOS



(b) Consola abierta en Windows

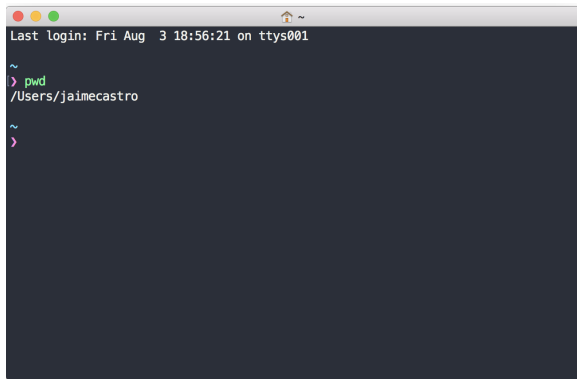


(a) “Home” en macOS



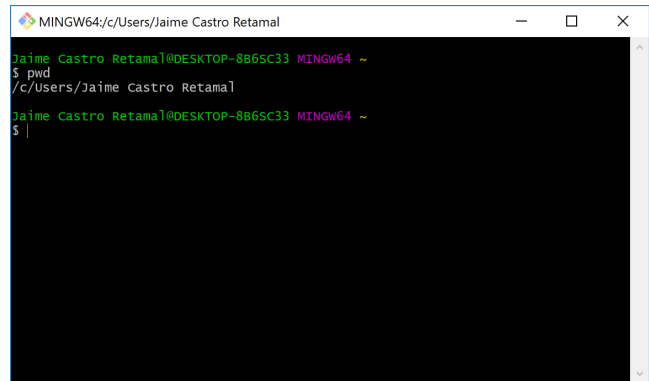
(b) “Home” en Windows

Muchas veces las consolas muestran en qué parte están ubicadas, pero el comando `pwd` también nos entrega esa información. Escríbelo en tu consola y apreta *enter*.



```
Last login: Fri Aug 3 18:56:21 on ttys001
~
> pwd
/Users/jaimecastro
~
>
```

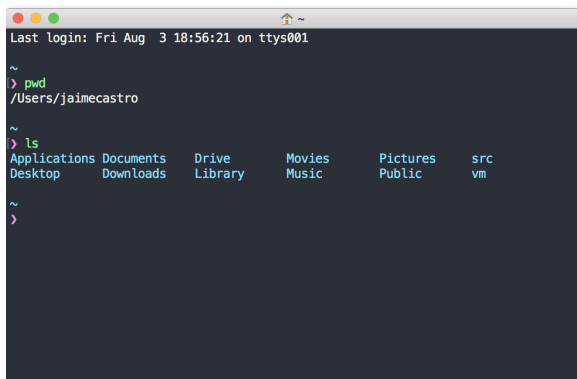
(a) `pwd` en macOS estando en el “home”



```
MINGW64:/c/Users/Jaime Castro Retamal
Jaime Castro Retamal@DESKTOP-8B65C33 MINGW64 ~
$ pwd
/c/Users/Jaime Castro Retamal
Jaime Castro Retamal@DESKTOP-8B65C33 MINGW64 ~
$ |
```

(b) `pwd` en Windows estando en el “home”

También podemos saber qué es lo que contiene el directorio actual, con el comando `ls`.



```
Last login: Fri Aug 3 18:56:21 on ttys001
~
> pwd
/Users/jaimecastro
~
> ls
Applications  Documents  Drive      Movies     Pictures   src
Desktop       Downloads  Library    Music      Public     vm
~
>
```

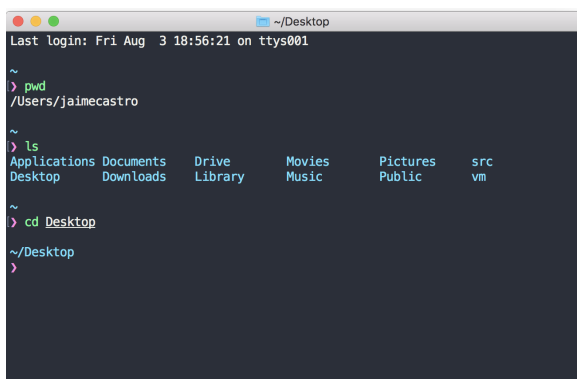
(a) `ls` en el “home” en macOS



```
MINGW64:/c/Users/Jaime Castro Retamal
My Documents'@
Nethood@
NTUSER.DAT
ntuser.dat.LOG1
ntuser.dat.LOG2
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TM.b1f
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TMContainer000000000000000000
1.regtrans-ms
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TMContainer000000000000000000
2.regtrans-ms
ntuser.ini
OneDrive/
Pictures/
PrintHood@
Recent@
'Saved Games'/
Searches/
SendTo@
'Start Menu'@
Templates@
Videos/
Jaime Castro Retamal@DESKTOP-8B65C33 MINGW64 ~
$ |
```

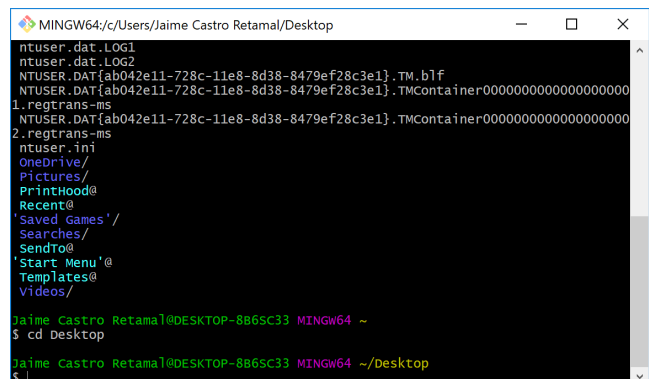
(b) `ls` en el “home” en Windows

Supongamos que queremos mover la ubicación de la consola a la carpeta “Desktop”. Para ello, usamos el comando `cd` (*change directory*). En este caso, debemos teclear `cd Desktop`. Puedes verificar que cambiaste de directorio utilizando `pwd` nuevamente.



```
~/Desktop
Last login: Fri Aug 3 18:56:21 on ttys001
~
> pwd
/Users/jaimecastro
~
> ls
Applications  Documents  Drive      Movies     Pictures   src
Desktop       Downloads  Library    Music      Public     vm
~
> cd Desktop
~/Desktop
~
>
```

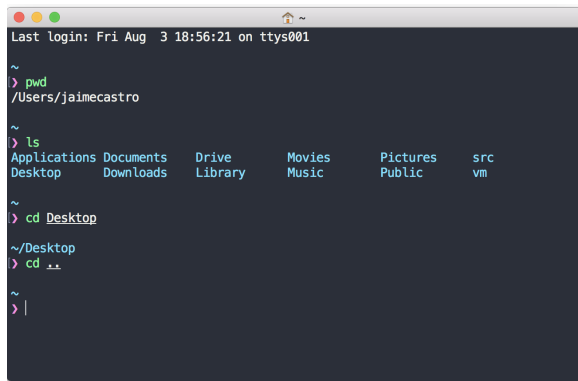
(a) `cd Desktop` en macOS



```
MINGW64:/c/Users/Jaime Castro Retamal/Desktop
ntuser.dat.LOG1
ntuser.dat.LOG2
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TM.b1f
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TMContainer000000000000000000
1.regtrans-ms
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TMContainer000000000000000000
2.regtrans-ms
ntuser.ini
OneDrive/
Pictures/
PrintHood@
Recent@
'Saved Games'/
Searches/
SendTo@
'Start Menu'@
Templates@
Videos/
Jaime Castro Retamal@DESKTOP-8B65C33 MINGW64 ~
$ cd Desktop
Jaime Castro Retamal@DESKTOP-8B65C33 MINGW64 ~/Desktop
$ |
```

(b) `cd Desktop` en Windows

Si nos queremos devolver a la carpeta que contiene a “Desktop”, usamos `cd ..` (los dos puntos significan “la carpeta que contiene la actual”).




```
Last login: Fri Aug 3 18:56:21 on ttys001

~
> pwd
/Users/jaimecastro

~
> ls
Applications  Documents  Drive      Movies     Pictures   src
Desktop       Downloads  Library    Music      Public     vm

~
> cd Desktop
~/Desktop
> cd ..
~
> |
```

(a) `cd ..` en macOS



```
MINGW64/c/Users/Jaime Castro Retamal

NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TMContainer00000000000000000000
1.regtrans-ms
NTUSER.DAT{ab042e11-728c-11e8-8d38-8479ef28c3e1}.TMContainer00000000000000000000
2.regtrans-ms
ntuser.ini
OneDrive/
Pictures/
PrintHood@
Recent@
'Saved Games'/
Searches/
SendTo@
'Start Menu'@
Templates@
Videos/

Jaime Castro Retamal@DESKTOP-8B65C33 MINGW64 ~
$ cd Desktop

Jaime Castro Retamal@DESKTOP-8B65C33 MINGW64 ~/Desktop
$ cd ..

Jaime Castro Retamal@DESKTOP-8B65C33 MINGW64 ~
$ |
```

(b) `cd ..` en Windows

Clonar repositorio personal

En esta parte, clonaremos tu repositorio personal para que puedas entregar tus actividades y tareas.

1. Asegúrate de que la consola esté dentro de la carpeta donde quieras tener tu repo, como el escritorio o alguna carpeta propia. Si no, navega usando los comandos mencionados hasta hallarlo.
2. Ve a la página de tu repositorio y copia el vínculo que permite clonar el repositorio (busca el botón verde que resalta y que muestra la siguiente foto).
3. En la consola, ejecuta el comando para clonarlo: `git clone url_copiada`.
4. Deberías ver que se creó la carpeta con el contenido de tu repositorio personal².

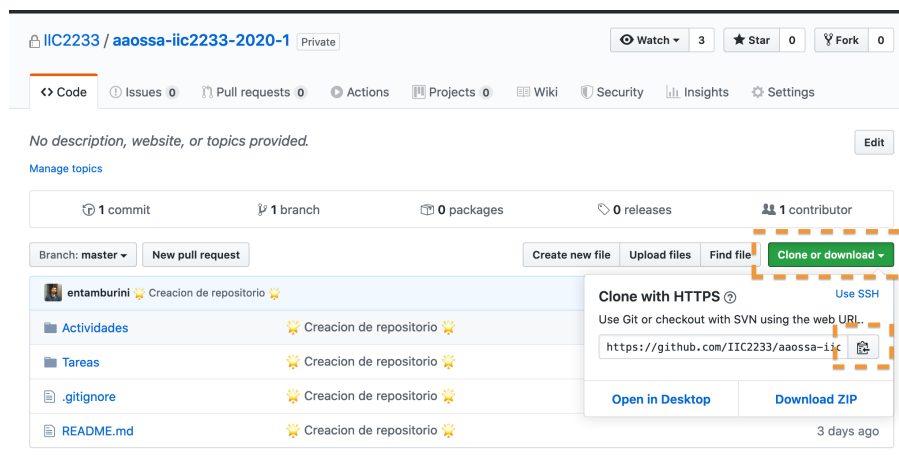


Figura 10: Como encontrar enlace de repositorio.

En caso de que ya hayas copiado tu repositorio personal en un lugar que no te guste, simplemente puedes mover la carpeta completa a un directorio que prefieras, los archivos que permiten que `git` funcione se encuentran dentro de tu carpeta y seguirán funcionando normalmente.

Clonar otros repositorio importantes del curso

Aprovechando el vuelo, repite los mismos pasos del punto anterior pero para `syllabus` y para `contenidos`, ya que tendrás que estar pendiente de estos repositorios durante el semestre.

²Si no sabes dónde se creó, recuerda usar el comando `pwd` para ver la ubicación actual de tu consola.

El primer *add*, *commit* y *push*

Haremos nuestro primer *commit* dentro del repositorio personal. Para trabajar en él, tu consola deberá estar **dentro** de la carpeta del repositorio (podrías usar el comando `cd`).

1. Dentro del repositorio, hay un archivo llamado `README.md` creado. En este, hay información sobre el uso de tu repositorio, y dejamos espacios para que rellenes con tus datos. Abre el archivo en tu computador con algún editor de texto y rellena los espacios con tus datos.
2. Ahora vuelve a la consola. Revisa el estado del repositorio utilizando `git status`. Observa que tu archivo recién editado aparece listado como *“modified”* en *“Changes not staged for commit”*.
3. Agrega el archivo `README.md` al *staging area* mediante el comando: `git add README.md`.
4. Revisa el estado del repositorio (`git status`) y verifica que tu archivo recién editado aparece listado como *“modified”* en *“Changes to be committed”*.
5. Utiliza: `git commit -m "README actualizado"`. El texto entre comillas se conoce como mensaje del *commit* y debe describir lo que fue agregado mediante `git add`.
6. Vuelve a revisar el estado (`git status`) del repositorio nuevamente. No deberías ver listado a `README.md`, pero si un mensaje que dice *“Your branch is ahead of 'origin/master' by 1 commit.”*, lo cual significa que el *commit* fue creado correctamente,
7. Ahora, para escribir tus cambios en GitHub utiliza: `git push`
8. Vuelve a revisar el estado del repositorio, debe decir: *Your branch is up to date with 'origin/master'. nothing to commit, working tree clean.*
9. Después ve el contenido del repositorio en un navegador web (GitHub) y verifica tus cambios. Si los ves, ¡lo lograste!

IMPORTANTE: Cuando escribas el `README.md` de tus tareas, debes seguir este mismo proceso, **no lo edites desde el navegador**, porque eso generará conflictos entre la versiones del archivo de tu computador y la versión del repositorio

Parte 2.5: El segundo *add*, *commit* y *push*

Haremos un segundo *commit* dentro del repositorio personal.

1. En la carpeta `Actividades/AC00/` crea un archivo `bienvenida.py` cuyo código imprima en pantalla `"Bienvenid@ a IIC2233"`.

Los siguientes pasos son iguales a la sección anterior, pero para el archivo `bienvenida.py`.
2. Revisa el estado del repositorio utilizando `git status`. Observa que tu archivo recién creado aparece bajo *“Untracked files”*.
3. Agrega el archivo `bienvenida.py` al *staging area* mediante `git add`. Recuerda que debes escribir el nombre del archivo después de este comando. Luego revisa el estado del repositorio y verifica que se haya agregado.
4. Utiliza `git commit -m "mensaje"` de los cambios realizados. Recuerda escribir un mensaje **descriptivo**. Luego revisa el estado del repositorio.
5. Ahora, utiliza `git push` para escribir tus cambios en GitHub. Luego, revisa el estado del repositorio y después ve el contenido del repositorio en un *browser* () y verifica tus cambios.

Parte 3: Actualizar tus repos locales

Ya hablamos de como agregar cambios locales (computador) y enviarlos a la versión remota tu repositorio (GitHub), pero no hemos hablado de como **traer cambios** del repositorio remoto a nuestro repositorio local.

Lo anterior, asume que posterior a que clonamos un repositorio, como *syllabus* o *contenidos*, se hayan subido cambios que queremos ver en nuestro repositorio local. **La solución no es clonar cada vez, si no que actualizar la versión ya clonada.** Para eso es: `git pull`.

Hoy (jueves 20) a las 15:30 se subirán nuevos archivos al *Syllabus* y deberás actualizarlo para poder completar la actividad, sin embargo si clonaste el repositorio luego de esa hora, este paso no será necesario, por lo que puedes pasar directamente a la **Parte 4**.

Actualizar *syllabus*

A continuación actualizaremos los contenidos del repositorio del curso *syllabus*. Para lograrlo, tu consola deberá estar ubicada dentro de la carpeta del repositorio.

- Primero, revisa el contenido de *syllabus* en tu computador, notarás que no hay muchos archivos, solo podrás ver este enunciado.
- Debemos esperar a que los ayudantes actualicen el repositorio *syllabus* del curso. Puedes revisar en github.com/IIC2233/syllabus en el navegador. Si ves que hay un archivo de extensión `.py` y otro de extensión `.csv` en la carpeta `AC00`, significa que ya es tiempo de actualizar tu repositorio local.
- Antes, en consola, revisa el estado del repositorio usando `git status`.
- Ahora, ejecuta el comando `git pull`. Este último debería descargar cambios que fueron subidos por los ayudantes a este repositorio.
- Si ves el contenido en tu computador nuevamente, te encontrarás con dos nuevos archivos llamados `main.py` y `ayudantes.csv`, lo que significa que puedes comenzar con la actividad.

Parte 4: Simulacro de Actividad

Ahora, deberás trabajar sobre el archivo `.py` entregado, **pero lo harás en tu repositorio personal**. Para esto, debes comenzar copiando los archivos `main.py` y `ayudantes.csv` (que encontrarás en la carpeta `Actividades/AC00/` del *syllabus* después de actualizar) a la carpeta `Actividades/AC00/` de tu repositorio.

A modo de actividad tendrás que obtener datos de los ayudantes del curso, aplicando tus conocimientos de estructuras de datos *built-in*. Para lo anterior, deberás completar las funciones del archivo `main.py` de forma que hagan lo siguiente:

- `def cargar_datos(path)`: Recibe el *path* a un archivo y retorna una lista con las líneas de este archivo.
- `def crear_ayudantes(datos)`: Recibe una lista de con las líneas del archivo de ayudantes y retorna una lista de `namedtuples` con la información de todos los ayudantes (nombre, cargo y usuario).
- `def encontrar_cargos(ayudantes)`: Recibe una lista de ayudantes y debe retornar un `set` con todos los posibles cargos que puede tener un ayudante del curso.
- `def ayudantes_por_cargo(ayudantes)`: Recibe una lista de ayudantes y retorna un diccionario que tiene como `keys` los posibles cargos de los ayudantes y como `values` la lista de ayudantes que tienen ese cargo.

Una vez que termines esto, debes subir todos tus cambios, como aprendiste en la Parte 2, a tu repositorio personal.

Notas

- Para saber si completaste correctamente cierta parte, puedes correr el código en cualquier momento de tu desarrollo. Cuando lo ejecutes imprimirá los datos que obtengas en las secciones completadas hasta el momento y un mensaje con las secciones que falta por completar.
- Puedes usar los *built-ins* `split` y `strip` al momento de leer el archivo (en `cargar_datos`) para que te sea más fácil la siguiente función.
- Para la función `ayudantes_por_cargo` puede ayudarte utilizar un default `dict`
- Haz un *commit* cada vez que una idea lógica esté terminada. En este caso te recomendamos hacerlo cada vez que completes una de las funciones.
- Al final de este enunciado se encuentra una ayuda de memoria de los comandos de *Git*, esto puede serte de utilidad para esta y las demás actividades ;)

Requerimientos

Esta sección aparecerá en todas las actividades en clase, y detalla todas las cosas que se espera realices o entregues al finalizar la actividad. Te recordamos que esta actividad **no será evaluada**. Las siguientes actividades incluirán aquí el desglose del puntaje para cada requisito.

- Actualizar `README.md` de tu repositorio personal.
- *Script* `bienvenida.py` que imprima “Bienvenid@ a IIC2233”
- Completar la función `def cargar_datos`
- Completar la función `def crear_ayudantes`
- Completar la función `def encontrar_cargos`
- Completar la función `def ayudantes_por_cargo`

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC00/`
- **Hora del *push*:** 16:50

Ayuda memoria de operaciones con Git

Comando	Descripción
<code>git clone <url del repositorio></code>	Clonar repositorio remoto
<code>git status</code>	Revisar estado actual
<code>git add file_name</code>	Agregar un archivo al <i>staging area</i>
<code>git reset HEAD file_name</code>	Deshacer cambios de un archivo en el <i>staging area</i>
<code>git commit -m "Descripción del commit"</code>	Crear <i>commit</i> con un mensaje
<code>git reset HEAD~1</code>	Deshacer último <i>commit</i>
<code>git push</code>	Llevar versiones al repositorio remoto
<code>git pull</code>	Traer versiones al repositorio local