Probar los tests localmente

Descargate del Sercom el zip `testcases.zip`, anda a la carpeta raíz del repositorio y descomprimi el zip ahí.

Deberías tener algo así:

```
onboarding/
    casos/
    salidas/
    run_tests.sh
    compare_outputs.sh
```

Tanto en la carpeta `casos/` como en la carpeta `salidas/` verás varias subcarpetas. Cada una es un caso de prueba. Dentro de `casos/` están los archivos iniciales que se requieren para ejecutar el test y en `salidas/` están las salidas esperadas.

El script `run_tests.sh` es quien ejecuta todos los tests. El script **no** compila tu TP así que tenes que haber hecho un `make` **antes**. Segun el TP en el que estes lo tenes que ejecutar de una u otro manera:

TP de Sockets:

```
./run_tests.sh . casos/ single-client no-valgrind 60 10 no
```

TP de threads:

```
./run_tests.sh . casos/ multi-client no-valgrind 60 10 yes
```

El parámetro `no-valgrind` lo podes cambiar por `valgrind` para que los tests se ejecuten con Valgrind.
Los números 60 y 10 son los timeouts: el primero es el primer timeout de un test y si el programa está colgado y no responde, el segundo timeout entra en acción y hace un kill al programa. Es posible que quieras incrementar esos números si corres los tests con Valgrind.

Luego de la ejecución, las salidas generadas (las obtenidas) quedarán guardadas en las subcarpetas dentro de `casos/`

Para comparar usaras el segundo script `compare_outputs.sh`:

```
./compare_outputs.sh casos/ salidas/
```

El script te imprimirá las diferencias encontradas usando `diff`. Ten en cuenta que si algún test

te timeouteo, `tests_runner.sh` **aborta** la ejecución y hará que haya muchas más diferencias al correr `compare_outputs.sh`.

Si la salida de `diff` te confunde podes comparar vos mismo los resultados con otro programa como `meld`.

Probar un test local y manualmente

Es necesario que sepas cómo correr un caso manualmente por que te va a permitir experimentar con él y correrlo con un debugger. *¿Ya hiciste el recap de GDB?*

Cada subcarpeta dentro de `casos/` es un caso de prueba. En ellas habrá un archivo `*args__` (habrá un `*args__` para el cliente y otro para el servidor). Este archivo contiene los **parámetros de la línea de comando** para ejecutar tu programa. Así sabrás como el test ejecuta tu entrega.

Además de `*args__` habrán archivos `*stdin__`. Estos son la entrada estándar que deberás redireccionar a cada programa. *¿Ya hiciste el recap de Proceso de Building y Testing?*

En el TP de *Sockets* tendrás 2 programas, `client` y `server`, por lo que tendrás 2 `*args__` y 2 `*stdin__`.

En el TP de *Threads* tendrás también 2 programas, `client` y `server`, pero cada test lanzará **múltiples clientes** así que tendrás **múltiples** `*args__` y `*stdin__` (verás que cada uno está numerado y corresponde al número de cliente). Es muy posible que quieras solo usar algunos y debuggear un solo cliente en estos casos.

Correr los pre-commit hooks manualmente

Para correr los linters tendrás que ejecutar **manualmente** los hooks de `pre-commit`. Reemplaza `` `xxx.cpp` `` por el o los archivos `.h` y `.cpp` que tengas:

```
pre-commit run --hook-stage manual --files xxx.cpp
```

En este *onboarding* los hooks solo se ejecutan **manualmente**. En el resto de los TPs será de forma **automática en cada commit.**