

git checkout master && git pull && git checkout - && git rebase master

git log -oneline

dotnet run

dotnet build -> dotnet watch run

dotnet restore (Restore Nuget Package)

git log –oneline

<img

srcset="elva-fairy-480w.jpg 480w, elva-fairy-800w.jpg 800w"

sizes="(max-width: 600px) 480px,

800px"

src="elva-fairy-800w.jpg"

alt="Elva dressed as a fairy" />

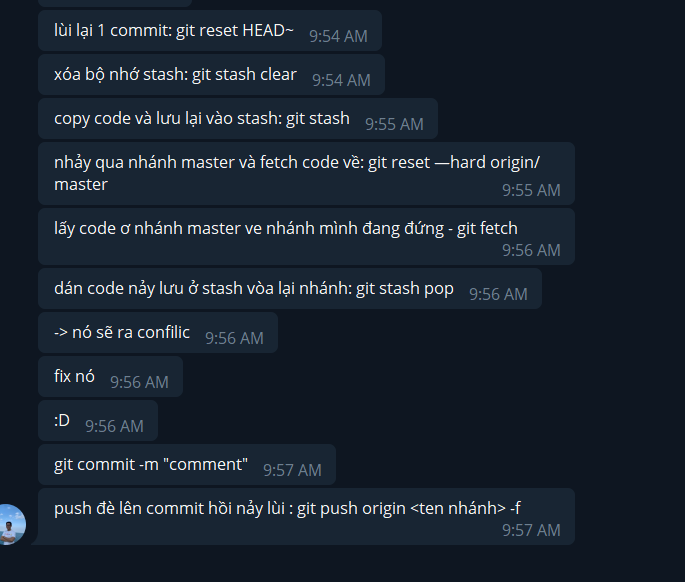
Đổi account:

git config user.name "new name"

git config credential.username "new name"

aspose words

https://www.convertapi.com/doc/dotnet-library



**Git fetch trước git reset –hard…..**

**Git push -u origin CurrentBranch -f**

****

nguyenhuukhanhphat@gmail.com / PPWAM1

string query = "Update YourTable Set ... Where ...";

context.Database.ExecuteSqlCommandAsync(query, new SqlParameter("@ColumnY", value1), new SqlParameter("@ColumnZ", value2));

<https://stackoverflow.com/questions/1774670/c-sharp-threadpool-vs-tasks>

11

# Thread

The bare metal thing, you probably don't need to use it, you probably can use a LongRunning Task and benefit from its facilities.

**Tasks**

Abstraction above the Threads. It **uses the thread pool** (unless you specify the task as a LongRunning operation, if so, a new thread is created under the hood for you).

**Thread Pool**

As the name suggests: a pool of threads. Is the .NET framework handling a limited number of threads for you. Why? Because opening 100 threads to execute expensive CPU operations on a CPU with just 8 cores definitely is not a good idea. The framework will maintain this pool for you, reusing the threads (not creating/killing them at each operation), and executing some of they in parallel in a way that your CPU will not burn.

OK, but when to use each one?

In resume: always use tasks.

Task is an abstratcion, so it is a lot easier to use. I advise you to always try to use Tasks and if you face some problem that makes you need to handle a thread by yourself (probably 1% of the time) then use threads.

BUT be aware that:

* **I/O Bound**: For I/O bound operations (database calls, read/write files, APIs calls, etc) **never use normal tasks**, use LongRunning tasks or threads if you need to, but not normal tasks. Because it would lead you to a thread pool with a few threads busy and a lot of another tasks waiting for its turn to take the pool.
* **CPU Bound**: For CPU bound operations just use the normal tasks and be happy.

Your advice on avoiding Tasks for I/O is plain wrong. Tasks work perfectly for I/O bound operations and are encouraged by async and await

**ThreadPool** is basically help to manage and reuse the free threads. In other words a threadpool is the collection of background thread.

**Task** work asynchronously manages the the unit of work. In easy words Task doesn’t create new threads. Instead it efficiently manages the threads of a threadpool.Tasks are executed by TaskScheduler, which queues tasks onto threads.