Scenarios

> ⚠ **Warning**
>
> This is a more advanced topic, if you are starting with **Typer**, feel free to skip it.
>
> It will be mostly useful for people that already work with Click and have questions around it.

Typer is powered by Click [↪]. It does all the work underneath.

Here is some more information related to using both together.

A single app with both Click and **Typer**

If you already have a Click application and want to migrate to **Typer**, or to add some Typer components, you can get a Click `Command` from your Typer application and then use Click directly.

How Click works

Before knowing how to combine Click and **Typer**, let's first check a little about how Click works.

Click `Command`

Any Click application has an object of class `Command`. That's, more or less, the most basic Click object.

A `Command` can have its own *CLI arguments* and *CLI options*, and it has a function that it calls.

For example, in this Click app:

```
{!../docs_src/using_click/tutorial001.py!}
```

The original `hello` variable is converted by Click from a function to a `Command` object. And the original `hello` function is used by that `Command` internally, but it is no longer named `hello` (as `hello` is now a Click `Command`).

Click `Group`

Then Click also has a `Group` class, it **inherits from** `Command`. So, a `Group` object is *also* a `Command`.

A `Group` can also have its own *CLI arguments* and *CLI options*.

A `Group` can have subcommands of class `Command` or sub groups of class `Group` as well.

And a `Group` can also have a function that it calls, right before calling the function for any specific subcommand.

For example:

```
{!../docs_src/using_click/tutorial002.py!}
```

The `cli` variable is converted by Click from a function to a `Group` object. And the original `cli` function is used by that `Group` internally.

Tip

The original `cli` function would be the equivalent of a Typer Callback ↪.

Then the `cli` variable, that now is a `Group` object, is used to add sub-commands.

How **Typer** works

Typer doesn't modify the functions. You create an explicit variable of class `typer.Typer` and use it to *register* those functions.

And then, when you call the app, Typer goes and creates a Click `Command` (or `Group`), and then calls it.

If your app only has one command, then when you call it, **Typer** creates a single Click `Command` object and calls it.

But **Typer** creates a Click `Group` object if your app has any of:

- More than one command.
- A callback.
- Sub-Typer apps (sub commands).

Tip

If you want to learn more about this check the section One or Multiple Commands ↪.

Combine Click and **Typer**

Typer uses an internal function `typer.main.get_command()` to generate a Click `Command` (or `Group`) from a `typer.Typer` object.

You can use it directly, and use the Click object with other Click applications.

Including a Click app in a **Typer** app

For example, you could have a **Typer** app, generate a Click `Group` from it, and then include other Click apps in it:

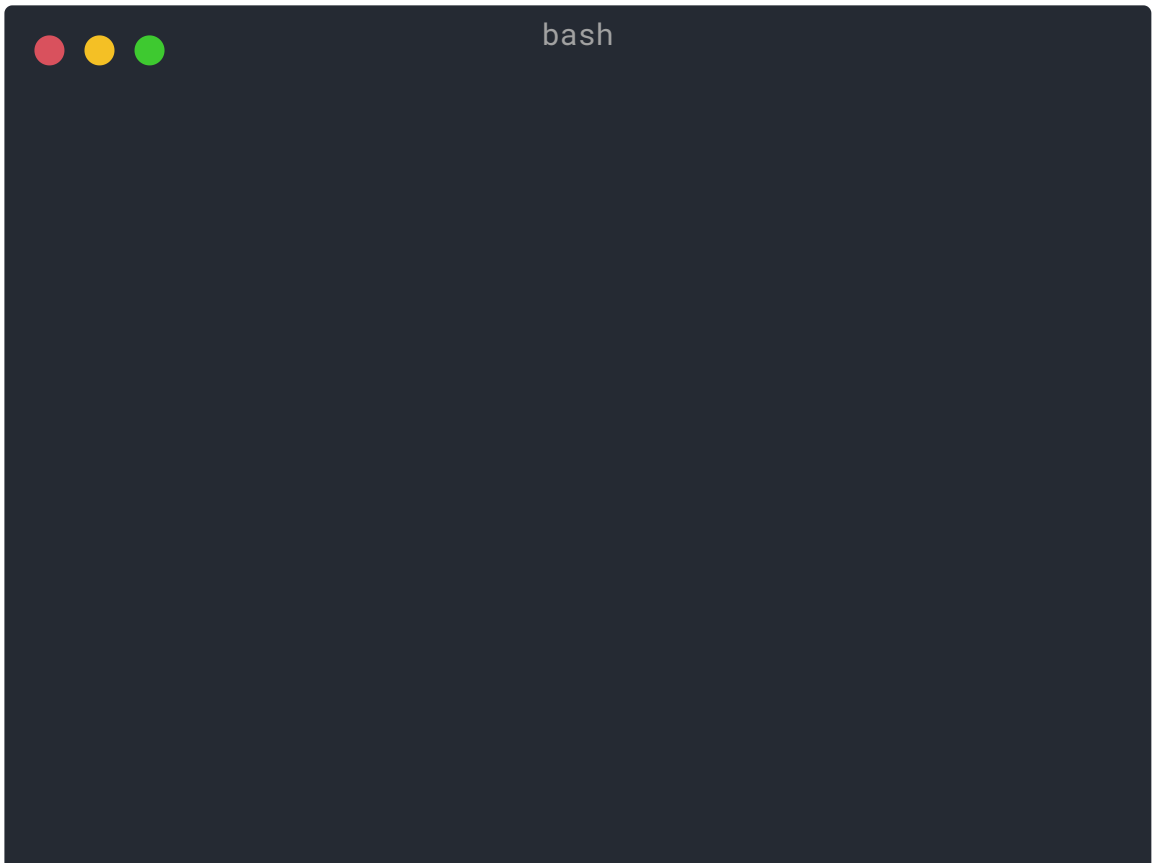
```
{!../docs_src/using_click/tutorial003.py!}
```

Notice that we add a callback that does nothing (only document the CLI program), to make sure **Typer** creates a Click `Group`. That way we can add sub-commands to that Click `Group`.

Then we generate a Click object from our `typer.Typer` app (`typer_click_object`), and then we can include another Click object (`hello`) in this Click `Group`.

And that way, our **Typer** app will have a subcommand `top` built with Typer, and a subcommand `hello` built with Click.

Check it:



Including a **Typer** app in a Click app

The same way, you can do the contrary and include a **Typer** sub app in a bigger Click app:

```
{!../docs_src/using_click/tutorial004.py!}
```

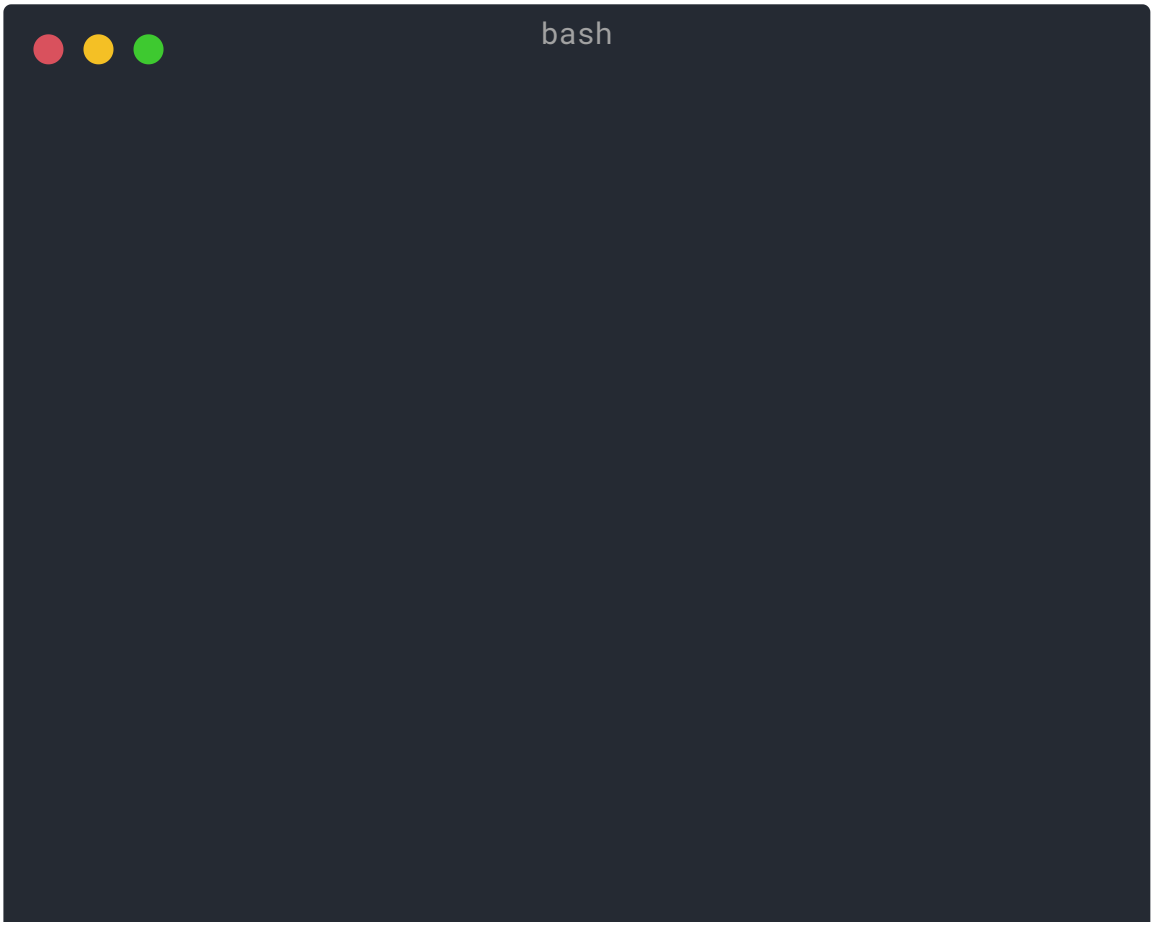
Notice that we don't have to add a callback or more commands, we can just create a **Typer** app that generates a single Click `Command`, as we don't need to include anything under the Typer app.

Then we generate a Click object from our `typer.Typer` app (`typer_click_object`), and then we use **the Click `cli` to include** our Click object from our Typer app.

In this case, the original Click app includes the **Typer** app.

And then we call the *original Click* app, not the Typer app.

Check it:



About Click decorators

Typer apps don't work with Click decorators directly.

This is because **Typer** doesn't modify functions to add metadata or to convert them to another object like Click does.

So, things like `@click.pass_context` won't work.

Most of the functionality provided by decorators in Click has an alternative way of doing it in **Typer**.

For example, to access the context, you can just declare a function parameter of type `typer.Context`.

Tip

You can read more about using the context in the docs: Commands: Using the Context ↪

But if you need to use something based on Click decorators, you can always generate a Click object using the methods described above, and use it as you would normally use Click.