NOTE: When I first spent time reading about REST, idempotence was a confusing concept to try to get right. I still didn't get it quite right in my original answer, as further comments (and Jason Hoetger's answer) have shown. For a while, I have resisted updating this answer extensively, to avoid effectively plagiarizing Jason, but I'm editing it now because, well, I was asked to (in the comments).

After reading my answer, I suggest you also read Jason Hoetger's excellent answer to this question, and I will try to make my answer better without simply stealing from Jason.

Why is PUT idempotent?

As you noted in your RFC 2616 citation, PUT is considered idempotent. When you PUT a resource, these two assumptions are in play:

1. You are referring to an entity, not to a collection.
2. The entity you are supplying is complete (the *entire* entity).

Let's look at one of your examples.

```
{ "username": "skwee357", "email": "skwee357@domain.com" }
```

If you POST this document to `/users`, as you suggest, then you might get back an entity such as

```
## /users/1

{
    "username": "skwee357",
    "email": "skwee357@domain.com"
}
```

If you want to modify this entity later, you choose between PUT and PATCH. A PUT might look like this:

```
PUT /users/1
{
    "username": "skwee357",
    "email": "skwee357@gmail.com"       // new email address
}
```

You can accomplish the same using PATCH. That might look like this:

```
PATCH /users/1
{
    "email": "skwee357@gmail.com"       // new email address
}
```

You'll notice a difference right away between these two. The PUT included all of the parameters on this user, but PATCH only included the one that was being modified (`email`).

When using PUT, it is assumed that you are sending the complete entity, and that complete entity *replaces* any existing entity at that URI. In the above example, the PUT and PATCH accomplish the same goal: they both change this user's email address. But PUT handles it by replacing the entire entity, while PATCH only updates the fields that were supplied, leaving the others alone.

Since PUT requests include the entire entity, if you issue the same request repeatedly, it should always have the same outcome (the data you sent is now the entire data of the entity). Therefore PUT is idempotent.

Using PUT wrong

What happens if you use the above PATCH data in a PUT request?

```
GET /users/1
{
    "username": "skwee357",
    "email": "skwee357@domain.com"
}
PUT /users/1
{
    "email": "skwee357@gmail.com"       // new email address
}

GET /users/1
{
    "email": "skwee357@gmail.com"      // new email address... and nothing else!
}
```

(I'm assuming for the purposes of this question that the server doesn't have any specific required fields, and would allow this to happen... that may not be the case in reality.)

Since we used PUT, but only supplied `email`, now that's the only thing in this entity. This has resulted in data loss.

This example is here for illustrative purposes -- don't ever actually do this. This PUT request is technically idempotent, but that doesn't mean it isn't a terrible, broken idea.

How can PATCH be idempotent?

In the above example, PATCH *was* idempotent. You made a change, but if you made the same change again and again, it would always give back the same result: you changed the email address to the new value.

```
GET /users/1
{
    "username": "skwee357",
    "email": "skwee357@domain.com"
}
PATCH /users/1
{
    "email": "skwee357@gmail.com"       // new email address
}

GET /users/1
{
    "username": "skwee357",
    "email": "skwee357@gmail.com"       // email address was changed
}
PATCH /users/1
{
    "email": "skwee357@gmail.com"       // new email address... again
}

GET /users/1
{
    "username": "skwee357",
    "email": "skwee357@gmail.com"       // nothing changed since last GET
}
```

My original example, fixed for accuracy

I originally had examples that I thought were showing non-idempotency, but they were misleading / incorrect. I am going to keep the examples, but use them to illustrate a different thing: that multiple PATCH documents against the same entity, modifying different attributes, do not make the PATCHes non-idempotent.

Let's say that at some past time, a user was added. This is the state that you are starting from.

```
{
  "id": 1,
  "name": "Sam Kwee",
  "email": "skwee357@olddomain.com",
  "address": "123 Mockingbird Lane",
  "city": "New York",
  "state": "NY",
  "zip": "10001"
}
```

After a PATCH, you have a modified entity:

```
PATCH /users/1
{"email": "skwee357@newdomain.com"}

{
  "id": 1,
  "name": "Sam Kwee",
  "email": "skwee357@newdomain.com",    // the email changed, yay!
  "address": "123 Mockingbird Lane",
  "city": "New York",
  "state": "NY",
  "zip": "10001"
}
```

If you then repeatedly apply your PATCH, you will continue to get the same result: the email was changed to the new value. A goes in, A comes out, therefore this is idempotent.

An hour later, after you have gone to make some coffee and take a break, someone else comes along with their own PATCH. It seems the Post Office has been making some changes.

```
PATCH /users/1
{"zip": "12345"}

{
  "id": 1,
  "name": "Sam Kwee",
  "email": "skwee357@newdomain.com",  // still the new email you set
  "address": "123 Mockingbird Lane",
  "city": "New York",
  "state": "NY",
  "zip": "12345"                      // and this change as well
}
```

Since this PATCH from the post office doesn't concern itself with email, only zip code, if it is repeatedly applied, it will also get the same result: the zip code is set to the new value. A goes in, A comes out, therefore this is *also* idempotent.

The next day, you decide to send your PATCH again.

```
PATCH /users/1
{"email": "skwee357@newdomain.com"}

{
  "id": 1,
  "name": "Sam Kwee",
  "email": "skwee357@newdomain.com",
  "address": "123 Mockingbird Lane",
  "city": "New York",
  "state": "NY",
  "zip": "12345"
}
```

Your patch has the same effect it had yesterday: it set the email address. A went in, A came out, therefore this is idempotent as well.

What I got wrong in my original answer

I want to draw an important distinction (something I got wrong in my original answer). Many servers will respond to your REST requests by sending back the new entity state, with your modifications (if any). So, when you get this *response* back, it is different *from the one you got back yesterday*, because the zip code is not the one you received last time. However, your request was not concerned with the zip code, only with the email. So your PATCH document is still idempotent - the email you sent in PATCH is now the email address on the entity.

So when is PATCH not idempotent, then?

For a full treatment of this question, I again refer you to Jason Hoetger's answer. I'm just going to leave it at that, because I honestly don't think I can answer this part better than he already has.