Overall:

Both PUT and POST can be used for creating.

You have to ask "what are you performing the action to?" to distinguish what you should be using. Let's assume you're designing an API for asking questions. If you want to use POST then you would do that to a list of questions. If you want to use PUT then you would do that to a particular question.

Great both can be used, so which one should I use in my RESTful design:

You do not need to support both PUT and POST.

Which is used is left up to you. But just remember to use the right one depending on what object you are referencing in the request.

Some considerations:

- Do you name your URL objects you create explicitly, or let the server decide? If you name them then use PUT. If you let the server decide then use POST.
- PUT is idempotent, so if you PUT an object twice, it has no effect. This is a nice property, so I would use PUT when possible.
- You can update or create a resource with PUT with the same object URL
- With POST you can have 2 requests coming in at the same time making modifications to a URL, and they may update different parts of the object.

An example:

I wrote the following as part of another answer on SO regarding this:

POST:

Used to modify and update a resource

```
POST /questions/<existing_question> HTTP/1.1
Host: www.example.com/
```

Note that the following is an error:

```
POST /questions/<new_question> HTTP/1.1
Host: www.example.com/
```

If the URL is not yet created, you should not be using POST to create it while specifying the name. This should result in a 'resource not found' error because `<new_question>` does not exist yet. You should PUT the `<new_question>` resource on the server first.

You could though do something like this to create a resources using POST:

```
POST /questions HTTP/1.1
Host: www.example.com/
```

Note that in this case the resource name is not specified, the new objects URL path would be returned to you.

PUT:

Used to create a resource, or overwrite it. While you specify the resources new URL.

For a new resource:

```
PUT /questions/<new_question> HTTP/1.1
Host: www.example.com/
```

To overwrite an existing resource:

```
PUT /questions/<existing_question> HTTP/1.1
Host: www.example.com/
```

(THE MOST CORRECT ANSWER)

Summary:

Create:

Can be performed with both PUT or POST in the following way:

PUT

Creates **THE** new resource with **newResourceId** as the identifier, under the /resources URI, or **collection**.

```
PUT /resources/<newResourceId> HTTP/1.1
```

POST

Creates **A** new resource under the /resources URI, or **collection**. Usually the identifier is returned by the server.

```
POST /resources HTTP/1.1
```

Update:

Can **only** be performed with PUT in the following way:

PUT

Updates the resource with **existingResourceId** as the identifier, under the /resources URI, or **collection**.

```
PUT /resources/<existingResourceId> HTTP/1.1
```

Explanation:

When dealing with REST and URI as general, you have **generic** on the *left* and **specific** on the *right*. The **generics** are usually called **collections** and the more **specific** items can be called **resource**. Note that a **resource** can contain a **collection**.

Examples:

<-- generic -- specific -->

```
URI: website.com/users/john
website.com  - whole site
users        - collection of users
john         - item of the collection, or a resource

URI:website.com/users/john/posts/23
website.com  - whole site
users        - collection of users
john         - item of the collection, or a resource
posts        - collection of posts from john
```

```
23          - post from john with identifier 23, also a resource
```

When you use POST you are **always** refering to a **collection**, so whenever you say:

```
POST /users HTTP/1.1
```

you are posting a new user to the *users* **collection**.

If you go on and try something like this:

```
POST /users/john HTTP/1.1
```

it will work, but semantically you are saying that you want to add a resource to the *john* **collection** under the *users* **collection**.

Once you are using PUT you are refering to a **resource** or single item, possibly inside a **collection**. So when you say:

```
PUT /users/john HTTP/1.1
```

you are telling to the server update, or create if it doesn't exist, the *john* **resource** under the *users* **collection**.

Spec:

Let me highlight some important parts of the spec:

POST

The **POST** method is used to request that the origin server **accept** the entity enclosed in the request as a ***new* subordinate** of the resource identified by the Request-URI in the Request-Line

Hence, creates a new **resource** on a **collection**.

PUT

The **PUT** method requests that the enclosed entity be **stored** under the supplied Request-URI. If the Request-URI refers to an **already existing** resource, the enclosed entity SHOULD be considered as a **modified version** of the one residing on the origin server. If the Request-URI does **not point to an existing** resource, and that URI is **capable** of being defined as a ***new* resource** by the requesting user agent, the origin server can **create** the resource with that URI."

Hence, create or update based on existence of the **resource**.

Reference:

- HTTP/1.1 Spec
- Wikipedia - REST
- Uniform Resource Identifiers (URI): Generic Syntax and Semantics