`useEffect` is used in functional components to handle side effects. Side effects are operations that are not purely computational and affect or are affected by the outside world. Here are some common scenarios when you would use `useEffect`:

1. **Fetching Data**

When you need to fetch data from an API after the component mounts, use `useEffect` to perform the fetch operation.

```javascript
import React, { useEffect, useState } from 'react';

function DataFetcher() {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => {
        setData(data);
        setLoading(false);
      });
  }, []); // Empty array ensures this runs only once on mount

  if (loading) {
    return <p>Loading...</p>;
  }

  return <div>Data: {JSON.stringify(data)}</div>;
}
```

2. **Subscribing to Events**

When you need to add event listeners, use `useEffect` to add the listener on mount and clean it up on unmount.

```javascript
import React, { useEffect } from 'react';

function EventListener() {
  useEffect(() => {
    const handleResize = () => {
      console.log('Window resized');
    };

    window.addEventListener('resize', handleResize);

    return () => {
      window.removeEventListener('resize', handleResize);
    };
  }, []); // Empty array ensures this runs only once on mount

  return <div>Resize the window and check the console</div>;
}
```

3. **Updating the Document Title**

When you need to update the document title based on some state or prop changes.

```javascript
```

```
import React, { useEffect, useState } from 'react';

function DocumentTitleUpdater() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `Count: ${count}`;
  }, [count]); // Runs every time count changes

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

4. **Timers and Intervals**

When you need to set up timers or intervals and clear them when the component unmounts.

```javascript
import React, { useEffect, useState } from 'react';

function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds(prevSeconds => prevSeconds + 1);
    }, 1000);

    return () => {
      clearInterval(interval); // Cleanup interval on component unmount
    };
  }, []); // Empty array ensures this runs only once on mount

  return <div>Seconds: {seconds}</div>;
}
```

5. **Side Effects Dependent on Props or State**

When you need to perform a side effect whenever a particular prop or state changes.

```javascript
import React, { useEffect, useState } from 'react';

function FetchOnPropChange({ url }) {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch(url)
      .then(response => response.json())
      .then(data => setData(data));
  }, [url]); // Runs every time url changes

  return <div>Data: {JSON.stringify(data)}</div>;
}
```

Summary of When to Use `useEffect`

- **Fetching Data**: When you need to load data from an external source after the component mounts.
- **Subscribing to Events**: When you need to add event listeners or subscriptions and clean them up when the component unmounts.
- **Updating the DOM**: When you need to make changes to the DOM that are outside of React’s rendering process.
- **Timers and Intervals**: When you need to set up timers or intervals and clean them up when the component unmounts.
- **Effect Dependent on State or Props**: When you need to perform actions based on changes to state or props.

`useEffect` allows you to manage side effects in a functional component, ensuring that your code runs at the right times and cleans up properly when needed.