⚛️ How To Use All React Hooks (Practical Guide)

We’ll go from basic → advanced → production usage.

1️⃣ `useState` – Local State Management

✅ When to Use

- Counters
- Form inputs
- Toggle UI (modal, dropdown)
- Loading flags

Example

```
import { useState } from "react";

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

  return (
    <button onClick={() => setCount(c => c + 1)}>
      {count}
    </button>
  );
}
```

👉 **Rule:** Use `useState` for simple, local state.

2️⃣ `useEffect` – Side Effects

✅ When to Use

- API calls
- Event listeners
- Timers
- Subscriptions
- DOM updates

Example (API Fetch)

```
import { useState, useEffect } from "react";

function Users() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/users")
```

```
      .then(res => res.json())
      .then(data => setUsers(data));
  }, []);

  return <div>{users.length} Users</div>;
}
```

👉 Runs once because dependency array is `[]`.

3️⃣ `useContext` – Global State (Avoid Prop Drilling)

Problem:

Passing props through 5 components.

Solution:

Create Context

```
import { createContext } from "react";

export const ThemeContext = createContext();
```

Provide Context

```
<ThemeContext.Provider value="dark">
  <App />
</ThemeContext.Provider>
```

Consume Context

```
import { useContext } from "react";

const theme = useContext(ThemeContext);
```

👉 Use when data is needed across many components.

4️⃣ `useReducer` – Complex State Logic

Use When:

- Many related states
- Complex state transitions
- Large forms
- Dashboard filters

Example

```
import { useReducer } from "react";

function reducer(state, action) {
  switch (action.type) {
    case "increment":
      return { count: state.count + 1 };
    default:
      return state;
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, { count: 0 });

  return (
    <button onClick={() => dispatch({ type: "increment" })}>
      {state.count}
    </button>
  );
}
```

👉 Think of it like Redux but inside component.

5️⃣ `useRef` – DOM & Persistent Values

Use When:

- Access DOM element
- Store previous value
- Avoid re-render

Example (Focus Input)

```
import { useRef } from "react";

function InputFocus() {
  const inputRef = useRef(null);

  return (
    <>
      <input ref={inputRef} />
      <button onClick={() => inputRef.current.focus()}>
        Focus
      </button>
    </>
  );
}
```

6️⃣ `useMemo` – Performance Optimization

Use When:

- Expensive calculation
- Large list filtering

- Avoid recalculating every render

```
import { useMemo } from "react";

const filteredItems = useMemo(() => {
  return items.filter(item => item.active);
}, [items]);
```

👉 Only recalculates when `items` changes.

7️⃣ `useCallback` – Memoize Functions

Problem:

Function recreated every render → child re-renders

Solution:

```
import { useCallback } from "react";

const handleClick = useCallback(() => {
  console.log("Clicked");
}, []);
```

👉 Useful when passing functions to memoized children.

8️⃣ `useLayoutEffect` – DOM Before Paint

Similar to `useEffect` but runs **synchronously before browser paints**.

Use for:

- Measuring DOM size
- Animation setup

```
useLayoutEffect(() => {
  console.log("Runs before paint");
}, []);
```

9️⃣ Custom Hooks – Reusable Logic

Example: useFetch

```
import { useState, useEffect } from "react";

function useFetch(url) {
  const [data, setData] = useState(null);
  useEffect(() => {
    fetch(url)
      .then(res => res.json())
      .then(setData);
  }, [url]);

  return data;
}
```

Usage:

```
const data = useFetch("/api/users");
```

👉 Extract reusable logic into custom hooks.

🔥 How To Use Them Together (Real Example)

Example: Dashboard App

Feature	Hook Used
Login form	useState
Fetch user data	useEffect
Global auth	useContext
Complex filters	useReducer
Input focus	useRef
Expensive sorting	useMemo
Stable callback	useCallback

🧠 Real Learning Order (Important)

1. Master `useState`
2. Deep dive into `useEffect`
3. Learn `useContext`
4. Practice `useReducer`
5. Understand `useRef`
6. Optimize with `useMemo` & `useCallback`
7. Build custom hooks