# rules-of-hooks

Validates that components and hooks follow the Rules of Hooks.

## Rule Details

React relies on the order in which hooks are called to correctly preserve state between renders. Each time your component renders, React expects the exact same hooks to be called in the exact same order. When hooks are called conditionally or in loops, React loses track of which state corresponds to which hook call, leading to bugs like state mismatches and “Rendered fewer/more hooks than expected” errors.

## Common Violations

These patterns violate the Rules of Hooks:

- **Hooks in conditions** ( `if` / `else` , ternary, `&&` / `||` )
- **Hooks in loops** ( `for` , `while` , `do-while` )
- **Hooks after early returns**
- **Hooks in callbacks/event handlers**
- **Hooks in async functions**
- **Hooks in class methods**
- **Hooks at module level**

### 중요합니다!

**`use` hook**

The `use` hook is different from other React hooks. You can call it conditionally and in loops:

```
// ✅ `use` can be conditional
if (shouldFetch) {
  const data = use(fetchPromise);
}

// ✅ `use` can be in loops
for (const promise of promises) {
  results.push(use(promise));
}
```

However, `use` still has restrictions:

- Can't be wrapped in try/catch
- Must be called inside a component or hook

Learn more: `use` API Reference

## Invalid

Examples of incorrect code for this rule:

```
// ❌ Hook in condition
if (isLoggedIn) {
  const [user, setUser] = useState(null);
}

// ❌ Hook after early return
if (!data) return <Loading />;
const [processed, setProcessed] = useState(data);

// ❌ Hook in callback
<button onClick={() => {
  const [clicked, setClicked] = useState(false);
}}/>
```

```
// ❌ `use` in try/catch
try {
  const data = use(promise);
} catch (e) {
  // error handling
}

// ❌ Hook at module level
const globalState = useState(0); // Outside component
```

## Valid

Examples of correct code for this rule:

```
function Component({ isSpecial, shouldFetch, fetchPromise }) {
  // ✅ Hooks at top level
  const [count, setCount] = useState(0);
  const [name, setName] = useState('');

  if (!isSpecial) {
    return null;
  }

  if (shouldFetch) {
    // ✅ `use` can be conditional
    const data = use(fetchPromise);
    return <div>{data}</div>;
  }

  return <div>{name}: {count}</div>;
}
```

# Troubleshooting

## I want to fetch data based on some condition

You're trying to conditionally call useEffect:

```
// ❌ Conditional hook
if (isLoggedIn) {
  useEffect(() => {
    fetchUserData();
  }, []);
}
```

Call the hook unconditionally, check condition inside:

```
// ✅ Condition inside hook
useEffect(() => {
  if (isLoggedIn) {
    fetchUserData();
  }
}, [isLoggedIn]);
```

### 중요합니다!

There are better ways to fetch data rather than in a useEffect. Consider using React Query, useSWR, or React Router 6.4+ for data fetching. These solutions handle deduplicating requests, caching responses, and avoiding network waterfalls.

Learn more: Fetching Data

## I need different state for different scenarios

You're trying to conditionally initialize state:

```
// ❌ Conditional state
if (userType === 'admin') {
  const [permissions, setPermissions] = useState(adminPerms);
} else {
```

```
  const [permissions, setPermissions] = useState(userPerms);
}
```

Always call useState, conditionally set the initial value:

```
// ✅ Conditional initial value
const [permissions, setPermissions] = useState(
  userType === 'admin' ? adminPerms : userPerms
);
```

## Options

You can configure custom effect hooks using shared ESLint settings (available in `eslint-plugin-react-hooks` 6.1.1 and later):

```
{
  "settings": {
    "react-hooks": {
      "additionalEffectHooks": "(useMyEffect|useCustomEffect)"
    }
  }
}
```

- `additionalEffectHooks` : Regex pattern matching custom hooks that should be treated as effects. This allows `useEffectEvent` and similar event functions to be called from your custom effect hooks.

This shared configuration is used by both `rules-of-hooks` and `exhaustive-deps` rules, ensuring consistent behavior across all hook-related linting.

이전
exhaustive-deps

다음
component-hook-factories



uwu?