eslint-plugin-react-hooks

`eslint-plugin-react-hooks` provides ESLint rules to enforce the Rules of React.

This plugin helps you catch violations of React's rules at build time, ensuring your components and hooks follow React's rules for correctness and performance. The lints cover both fundamental React patterns (exhaustive-deps and rules-of-hooks) and issues flagged by React Compiler. React Compiler diagnostics are automatically surfaced by this ESLint plugin, and can be used even if your app hasn't adopted the compiler yet.

중요합니다!

When the compiler reports a diagnostic, it means that the compiler was able to statically detect a pattern that is not supported or breaks the Rules of React. When it detects this, it **automatically** skips over those components and hooks, while keeping the rest of your app compiled. This ensures optimal coverage of safe optimizations that won't break your app.

What this means for linting, is that you don't need to fix all violations immediately. Address them at your own pace to gradually increase the number of optimized components.

Recommended Rules

These rules are included in the `recommended` preset in `eslint-plugin-react-hooks`:

- `exhaustive-deps` - Validates that dependency arrays for React hooks contain all necessary dependencies

- `rules-of-hooks` - Validates that components and hooks follow the Rules of Hooks
- `component-hook-factories` - Validates higher order functions defining nested components or hooks
- `config` - Validates the compiler configuration options
- `error-boundaries` - Validates usage of Error Boundaries instead of try/catch for child errors
- `gating` - Validates configuration of gating mode
- `globals` - Validates against assignment/mutation of globals during render
- `immutability` - Validates against mutating props, state, and other immutable values
- `incompatible-library` - Validates against usage of libraries which are incompatible with memoization
- `preserve-manual-memoization` - Validates that existing manual memoization is preserved by the compiler
- `purity` - Validates that components/hooks are pure by checking known-impure functions
- `refs` - Validates correct usage of refs, not reading/writing during render
- `set-state-in-effect` - Validates against calling setState synchronously in an effect
- `set-state-in-render` - Validates against setting state during render
- `static-components` - Validates that components are static, not recreated every render
- `unsupported-syntax` - Validates against syntax that React Compiler does not support
- `use-memo` - Validates usage of the `useMemo` hook without a return value

다음
exhaustive-deps