exhaustive-deps

React Hook의 의존성 배열에 필요한 모든 의존성이 포함되어 있는지 검증합니다.

규칙 자세히보기

`useEffect`, `useMemo`, `useCallback` 과 같은 React Hook은 의존성 배열을 받습니다. 이 Hook들 안에서 참조한 값이 의존성 배열에 포함되지 않으면 그 값이 바뀌어도 React가 effect를 다시 실행하거나 값을 다시 계산하지 않습니다. 그 결과 Hook이 예전 값을 계속 붙잡고 사용하는 “stale closure(오래된 클로저)” 문제가 생겨 최신 값이 아닌 상태로 동작하게 됩니다.

흔한 위반 사항

이 오류는 effect 실행 시점을 조절하려고 의존성을 의도적으로 누락할 때 자주 발생합니다. Effect는 컴포넌트를 외부 시스템과 동기화하기 위한 용도여야 합니다. 의존성 배열은 effect가 어떤 값을 사용하고 있는지 React에게 알려주며, React는 이를 바탕으로 언제 다시 동기화해야 하는지 판단합니다.

린터 경고를 계속 피하려고 하거나 맞추기 어렵다고 느낀다면, 코드 구조를 다시 구성해야 할 가능성이 큽니다. 방법은 Effect의 의존성 제거하기 문서를 참고하세요.

유효하지 않음

이 규칙에 대한 잘못된 코드 예시

```
// ❌ Missing dependency
useEffect(() => {
  console.log(count);
}, []); // Missing 'count'

// ❌ Missing prop
useEffect(() => {
```

```
  fetchUser(userId);
}, []); // Missing 'userId'

// ❌ Incomplete dependencies
useMemo(() => {
  return items.sort(sortOrder);
}, [items]); // Missing 'sortOrder'
```

유효

이 규칙에 대한 올바른 예시

```
// ✅ All dependencies included
useEffect(() => {
  console.log(count);
}, [count]);

// ✅ All dependencies included
useEffect(() => {
  fetchUser(userId);
}, [userId]);
```

문제 해결

함수를 의존성으로 추가하면 무한 루프가 발생할 수 있습니다

effect를 사용하고 있지만, 렌더링이 일어날 때마다 새로운 함수를 매번 생성하고 있습니다.

```
// ❌ Causes infinite loop
const logItems = () => {
  console.log(items);
};

useEffect(() => {
  logItems();
}, [logItems]); // Infinite loop!
```

대부분의 경우 effect는 필요하지 않습니다. 대신 그 동작이 실제로 발생하는 지점에서 함수를 호출하세요.

```
// ✅ Call it from the event handler
const logItems = () => {
  console.log(items);
};

return <button onClick={logItems}>Log</button>;

// ✅ Or derive during render if there's no side effect
items.forEach(item => {
  console.log(item);
});
```

정말로 effect가 필요한 경우(예: 외부 무언가를 구독해야 하는 경우)에는, 의존성이 안정적이도록 만드세요.

```
// ✅ useCallback keeps the function reference stable
const logItems = useCallback(() => {
  console.log(items);
}, [items]);

useEffect(() => {
  logItems();
}, [logItems]);

// ✅ Or move the logic straight into the effect
useEffect(() => {
  console.log(items);
}, [items]);
```

effect를 한 번만 실행하기

마운트 시점에 effect를 한 번만 실행하고 싶지만, 린터가 누락된 의존성에 대해 경고합니다.

```
// ❌ Missing dependency
useEffect(() => {
  sendAnalytics(userId);
}, []); // Missing 'userId'
```

의존성을 포함하는 것이 권장되며, 정말로 한 번만 실행해야 한다면 Ref를 사용하세요.

```
// ✅ Include dependency
useEffect(() => {
  sendAnalytics(userId);
}, [userId]);

// ✅ Or use a ref guard inside an effect
const sent = useRef(false);

useEffect(() => {
  if (sent.current) {
    return;
  }

  sent.current = true;
  sendAnalytics(userId);
}, [userId]);
```

옵션

공유 ESLint 설정을 사용해 커스텀 Effect Hook을 설정할 수 있습니다(`eslint-plugin-react-hooks` 6.1.1 이상에서 지원).

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

- `additionalEffectHooks` : 철저한 의존성 검사를 적용해야 하는 커스텀 Hook을 정규식 패턴으로 지정합니다. 이 설정은 모든 `react-hooks` 규칙에서 공통으로 사용됩니다.

하위 호환성을 위해 이 규칙은 규칙 단위 옵션도 함께 지원합니다.

```
{
  "rules": {
    "react-hooks/exhaustive-deps": ["warn", {
      "additionalHooks": "(useMyCustomHook|useAnotherHook)"
    }]
  }
}
```

- `additionalHooks` : 빠짐없는 의존성 검사(exhaustive deps)를 적용해야 하는 Hook을 정규식으로 지정합니다. **참고:** 이 규칙별 옵션을 지정하면 공유 `settings` 설정보다 우선 적용됩니다.

이전
Lints

다음
rules-of-hooks

React 학습하기

빠르게 시작하기

설치하기

UI 표현하기

상호작용성 더하기

State 관리하기

탈출구

API 참고서

React APIs

React DOM APIs

커뮤니티

행동 강령

팀 소개

문서 기여자

감사의 말

더 보기

블로그

React Native

개인 정보 보호

약관