# set-state-in-render

Validates against unconditionally setting state during render, which can trigger additional renders and potential infinite render loops.

## Rule Details

Calling `setState` during render unconditionally triggers another render before the current one finishes. This creates an infinite loop that crashes your app.

## Common Violations

### Invalid

```
// ❌ Unconditional setState directly in render
function Component({value}) {
  const [count, setCount] = useState(0);
  setCount(value); // Infinite loop!
  return <div>{count}</div>;
}
```

### Valid

```
// ✅ Derive during render
function Component({items}) {
  const sorted = [...items].sort(); // Just calculate it in render
  return <ul>{sorted.map(/*...*/)}</ul>;
}

// ✅ Set state in event handler
function Component() {
```

```
  const [count, setCount] = useState(0);
  return (
    <button onClick={() => setCount(count + 1)}>
      {count}
    </button>
  );
}

// ✅ Derive from props instead of setting state
function Component({user}) {
  const name = user?.name || '';
  const email = user?.email || '';
  return <div>{name}</div>;
}

// ✅ Conditionally derive state from props and state from previous renders
function Component({ items }) {
  const [isReverse, setIsReverse] = useState(false);
  const [selection, setSelection] = useState(null);

  const [prevItems, setPrevItems] = useState(items);
  if (items !== prevItems) { // This condition makes it valid
    setPrevItems(items);
    setSelection(null);
  }
  // ...
}
```

# Troubleshooting

## I want to sync state to a prop

A common problem is trying to “fix” state after it renders. Suppose you want to keep a counter from exceeding a `max` prop:

```
// ❌ Wrong: clamps during render
function Counter({max}) {
  const [count, setCount] = useState(0);

  if (count > max) {
```

```
    setCount(max);
  }

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

As soon as `count` exceeds `max`, an infinite loop is triggered.

Instead, it's often better to move this logic to the event (the place where the state is first set). For example, you can enforce the maximum at the moment you update state:

```
// ✅ Clamp when updating
function Counter({max}) {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(current => Math.min(current + 1, max));
  };

  return <button onClick={increment}>{count}</button>;
}
```

Now the setter only runs in response to the click, React finishes the render normally, and `count` never crosses `max`.

In rare cases, you may need to adjust state based on information from previous renders. For those, follow this pattern of setting state conditionally.


이전
set-state-in-effect

다음
static-components

## 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