<Link>

▶ **Examples**

`<Link>` is a React component that extends the HTML `<a>` element to provide prefetching and client-side navigation between routes. It is the primary way to navigate between routes in Next.js.

For an example, consider a `pages` directory with the following files:

- `pages/index.js`
- `pages/about.js`
- `pages/blog/[slug].js`

We can have a link to each of these pages like so:

```
import Link from 'next/link';

function Home() {
  return (
    <ul>
      <li>
        <Link href="/">Home</Link>
      </li>
      <li>
        <Link href="/about">About Us</Link>
      </li>
      <li>
        <Link href="/blog/hello-world">Blog Post</Link>
      </li>
    </ul>
  );
}

export default Home;
```

Props

Here's a summary of the props available for the Link Component:

Prop	Example	Type	Required
`href`	`href="/dashboard"`	String or Object	Yes
`replace`	`replace={false}`	Boolean	-
`prefetch`	`prefetch={false}`	Boolean	-

Good to know: `<a>` tag attributes such as `className` or `target="_blank"` can be added to `<Link>` as props and will be passed to the underlying `<a>` element.

`href` (required)

The path or URL to navigate to.

```
<Link href="/dashboard">Dashboard</Link>
```

`href` can also accept an object, for example:

```
// Navigate to /about?name=test
<Link
  href={{
    pathname: '/about',
    query: { name: 'test' },
  }}
>
  About
</Link>
```

`replace`

Defaults to `false`. When `true`, `next/link` will replace the current history state instead of adding a new URL into the browser's history ↗ stack.

app/page.tsx

```
import Link from 'next/link';
```

```
export default function Page() {
  return (
    <Link href="/dashboard" replace>
      Dashboard
    </Link>
  );
}
```

`prefetch`

Defaults to `true`. When `true`, `next/link` will prefetch the page (denoted by the `href`) in the background. This is useful for improving the performance of client-side navigations. Any `<Link />` in the viewport (initially or through scroll) will be preloaded.

Prefetch can be disabled by passing `prefetch={false}`. Prefetching is only enabled in production.

app/page.tsx

```
import Link from 'next/link';

export default function Page() {
  return (
    <Link href="/dashboard" prefetch={false}>
      Dashboard
    </Link>
  );
}
```

Other Props

`legacyBehavior`

An `<a>` element is no longer required as a child of `<Link>`. Add the `legacyBehavior` prop to use the legacy behavior or remove the `<a>` to upgrade. A codemod is available to automatically upgrade your code.

Note: when `legacyBehavior` is not set to `true`, all `anchor` ↗ tag properties can be passed to `next/link` as well such as, `className`, `onClick`, etc.

`passHref`

Forces `Link` to send the `href` property to its child. Defaults to `false`

`scroll`

Scroll to the top of the page after a navigation. Defaults to `true`

`shallow`

Update the path of the current page without rerunning `getStaticProps`, `getServerSideProps` or `getInitialProps`. Defaults to `false`

`locale`

The active locale is automatically prepended. `locale` allows for providing a different locale. When `false` `href` has to include the locale as the default behavior is disabled.

Examples

Linking to Dynamic Routes

For dynamic routes, it can be handy to use template literals to create the link's path.

For example, you can generate a list of links to the dynamic route `pages/blog/[slug].js`

pages/blog/index.js

```
import Link from 'next/link';

function Posts({ posts }) {
  return (
    <ul>
      {posts.map((post) => (
        <li key={post.id}>
          <Link href={`/blog/${post.slug}`}>{post.title}</Link>
        </li>
      ))}
    </ul>
  );
}

export default Posts;
```

If the child is a custom component that wraps an `<a>` tag

If the child of `Link` is a custom component that wraps an `<a>` tag, you must add `passHref` to `Link`. This is necessary if you're using libraries like styled-components↗. Without this, the `<a>` tag will not have the `href` attribute, which hurts your site's accessibility and might affect SEO. If you're using ESLint, there is a built-in rule `next/link-passhref` to ensure correct usage of `passHref`.

```
import Link from 'next/link';
import styled from 'styled-components';

// This creates a custom component that wraps an <a> tag
const RedLink = styled.a`
  color: red;
`;

function NavLink({ href, name }) {
  return (
    <Link href={href} passHref legacyBehavior>
      <RedLink>{name}</RedLink>
    </Link>
  );
}

export default NavLink;
```

- If you're using emotion↗'s JSX pragma feature (`@jsx jsx`), you must use `passHref` even if you use an `<a>` tag directly.
- The component should support `onClick` property to trigger navigation correctly

If the child is a functional component

If the child of `Link` is a functional component, in addition to using `passHref` and `legacyBehavior`, you must wrap the component in `React.forwardRef`↗:

```
import Link from 'next/link';

// `onClick`, `href`, and `ref` need to be passed to the DOM element
// for proper handling
const MyButton = React.forwardRef(({ onClick, href }, ref) => {
  return (
    <a href={href} onClick={onClick} ref={ref}>
      Click Me
    </a>
  );
});

function Home() {
  return (
    <Link href="/about" passHref legacyBehavior>
      <MyButton />
```

```
      </Link>
    );
}

export default Home;
```

With URL Object

`Link` can also receive a URL object and it will automatically format it to create the URL string. Here's how to do it:

```
import Link from 'next/link';

function Home() {
  return (
    <ul>
      <li>
        <Link
          href={{
            pathname: '/about',
            query: { name: 'test' },
          }}
        >
          About us
        </Link>
      </li>
      <li>
        <Link
          href={{
            pathname: '/blog/[slug]',
            query: { slug: 'my-post' },
          }}
        >
          Blog Post
        </Link>
      </li>
    </ul>
  );
}

export default Home;
```

The above example has a link to:

- A predefined route: `/about?name=test`
- A dynamic route: `/blog/my-post`

You can use every property as defined in the Node.js URL module documentation↗.

Replace the URL instead of push

The default behavior of the `Link` component is to `push` a new URL into the `history` stack. You can use the `replace` prop to prevent adding a new entry, as in the following example:

```
<Link href="/about" replace>
  About us
</Link>
```

Disable scrolling to the top of the page

The default behavior of `Link` is to scroll to the top of the page. When there is a hash defined it will scroll to the specific id, like a normal `<a>` tag. To prevent scrolling to the top / hash `scroll={false}` can be added to `Link`:

```
<Link href="/#hashid" scroll={false}>
  Disables scrolling to the top
</Link>
```

Middleware

It's common to use Middleware for authentication or other purposes that involve rewriting the user to a different page. In order for the `<Link />` component to properly prefetch links with rewrites via Middleware, you need to tell Next.js both the URL to display and the URL to prefetch. This is required to avoid un-necessary fetches to middleware to know the correct route to prefetch.

For example, if you have want to serve a `/dashboard` route that has authenticated and visitor views, you may add something similar to the following in your Middleware to redirect the user to the correct page:

middleware.js

```
export function middleware(req) {
  const nextUrl = req.nextUrl;
  if (nextUrl.pathname === '/dashboard') {
    if (req.cookies.authToken) {
      return NextResponse.rewrite(new URL('/auth/dashboard', req.url));
    } else {
      return NextResponse.rewrite(new URL('/public/dashboard', req.url));
    }
  }
}
```

In this case, you would want to use the following code in your `<Link />` component:

```jsx
import Link from 'next/link';
import useIsAuthed from './hooks/useIsAuthed';

export default function Page() {
  const isAuthed = useIsAuthed();
  const path = isAuthed ? '/auth/dashboard' : '/dashboard';
  return (
    <Link as="/dashboard" href={path}>
      Dashboard
    </Link>
  );
}
```

Note: If you're using Dynamic Routes, you'll need to adapt your `as` and `href` props. For example, if you have a Dynamic Route like `/dashboard/[user]` that you want to present differently via middleware, you would write:

```
<Link href={{ pathname: '/dashboard/authed/[user]', query: { user: username } }}
as="/dashboard/[user]">Profile</Link>
```

.