Rewrites

▼ **Examples**

- Rewrites

▶ **Version History**

Rewrites allow you to map an incoming request path to a different destination path.

Rewrites act as a URL proxy and mask the destination path, making it appear the user hasn't changed their location on the site. In contrast, redirects will reroute to a new page and show the URL changes.

To use rewrites you can use the `rewrites` key in `next.config.js`:

```
module.exports = {
  async rewrites() {
    return [
      {
        source: '/about',
        destination: '/',
      },
    ]
  },
}
```

Rewrites are applied to client-side routing, a `<Link href="/about">` will have the rewrite applied in the above example.

`rewrites` is an async function that expects an array to be returned holding objects with `source` and `destination` properties:

- `source`: `String` - is the incoming request path pattern.
- `destination`: `String` is the path you want to route to.
- `basePath`: `false` or `undefined` - if false the basePath won't be included when matching, can be used for external rewrites only.
- `locale`: `false` or `undefined` - whether the locale should not be included when matching.
- `has` is an array of has objects with the `type`, `key` and `value` properties.

Rewrites are applied after checking the filesystem (pages and `/public` files) and before dynamic routes by default. This behavior can be changed by returning an object instead of an array from the `rewrites` function since `v10.1` of Next.js:

```
module.exports = {
  async rewrites() {
    return {
      beforeFiles: [
        // These rewrites are checked after headers/redirects
        // and before all files including _next/public files which
        // allows overriding page files
        {
          source: '/some-page',
          destination: '/somewhere-else',
```

```
          has: [{ type: 'query', key: 'overrideMe' }],
        },
      ],
      afterFiles: [
        // These rewrites are checked after pages/public files
        // are checked but before dynamic routes
        {
          source: '/non-existent',
          destination: '/somewhere-else',
        },
      ],
      fallback: [
        // These rewrites are checked after both pages/public files
        // and dynamic routes are checked
        {
          source: '/:path*',
          destination: `https://my-old-site.com/:path*`,
        },
      ],
    }
  },
}
```

Note: rewrites in `beforeFiles` do not check the filesystem/dynamic routes immediately after matching a source, they continue until all `beforeFiles` have been checked.

The order Next.js routes are checked is:

1. headers are checked/applied
2. redirects are checked/applied
3. `beforeFiles` rewrites are checked/applied
4. static files from the public directory, `_next/static` files, and non-dynamic pages are checked/served
5. `afterFiles` rewrites are checked/applied, if one of these rewrites is matched we check dynamic routes/static files after each match
6. `fallback` rewrites are checked/applied, these are applied before rendering the 404 page and after dynamic routes/all static assets have been checked.

Rewrite parameters

When using parameters in a rewrite the parameters will be passed in the query by default when none of the parameters are used in the `destination`.

```
module.exports = {
  async rewrites() {
    return [
      {
        source: '/old-about/:path*',
        destination: '/about', // The :path parameter isn't used here so will be
automatically passed in the query
      },
    ]
```

```
  },
}
```

If a parameter is used in the destination none of the parameters will be automatically passed in the query.

```
module.exports = {
  async rewrites() {
    return [
      {
        source: '/docs/:path*',
        destination: '/:path*', // The :path parameter is used here so will not be
automatically passed in the query
      },
    ]
  },
}
```

You can still pass the parameters manually in the query if one is already used in the destination by specifying the query in the `destination`.

```
module.exports = {
  async rewrites() {
    return [
      {
        source: '/:first/:second',
        destination: '/:first?second=:second',
        // Since the :first parameter is used in the destination the :second
parameter
        // will not automatically be added in the query although we can manually add
it
        // as shown above
      },
    ]
  },
}
```

Note: for static pages from the Automatic Static Optimization or prerendering params from rewrites will be parsed on the client after hydration and provided in the query.

Path Matching

Path matches are allowed, for example `/blog/:slug` will match `/blog/hello-world` (no nested paths):

```
module.exports = {
  async rewrites() {
    return [
      {
        source: '/blog/:slug',
        destination: '/news/:slug', // Matched parameters can be used in the
destination
```

```
      },
    ]
  },
}
```

Wildcard Path Matching

To match a wildcard path you can use `*` after a parameter, for example `/blog/:slug*` will match `/blog/a/b/c/d/hello-world`:

```
module.exports = {
  async rewrites() {
    return [
      {
        source: '/blog/:slug*',
        destination: '/news/:slug*', // Matched parameters can be used in the
destination
      },
    ]
  },
}
```

Regex Path Matching

To match a regex path you can wrap the regex in parenthesis after a parameter, for example `/blog/:slug(\\d{1,})` will match `/blog/123` but not `/blog/abc`:

```
module.exports = {
  async rewrites() {
    return [
      {
        source: '/old-blog/:post(\\d{1,})',
        destination: '/blog/:post', // Matched parameters can be used in the
destination
      },
    ]
  },
}
```

The following characters `(`, `)`, `{`, `}`, `:`, `*`, `+`, `?` are used for regex path matching, so when used in the `source` as non-special values they must be escaped by adding `\\` before them:

```
module.exports = {
  async rewrites() {
    return [
      {
        // this will match `/english(default)/something` being requested
        source: '/english\\(default\\)/:slug',
        destination: '/en-us/:slug',
      },
```

```
    ]
  },
}
```

Header, Cookie, and Query Matching

To only match a rewrite when header, cookie, or query values also match the `has` field can be used. Both the `source` and all `has` items must match for the rewrite to be applied.

`has` items have the following fields:

- `type` : `String` - must be either `header` , `cookie` , `host` , or `query` .
- `key` : `String` - the key from the selected type to match against.
- `value` : `String` or `undefined` - the value to check for, if undefined any value will match. A regex like string can be used to capture a specific part of the value, e.g. if the value `first-(?<paramName>.*)` is used for `first-second` then `second` will be usable in the destination with `:paramName` .

```
module.exports = {
  async rewrites() {
    return [
      // if the header `x-rewrite-me` is present,
      // this rewrite will be applied
      {
        source: '/:path*',
        has: [
          {
            type: 'header',
            key: 'x-rewrite-me',
          },
        ],
        destination: '/another-page',
      },
      // if the source, query, and cookie are matched,
      // this rewrite will be applied
      {
        source: '/specific/:path*',
        has: [
          {
            type: 'query',
            key: 'page',
            // the page value will not be available in the
            // destination since value is provided and doesn't
            // use a named capture group e.g. (?<page>home)
            value: 'home',
          },
          {
            type: 'cookie',
            key: 'authorized',
            value: 'true',
          },
```

```
        ],
        destination: '/:path*/home',
      },
      // if the header `x-authorized` is present and
      // contains a matching value, this rewrite will be applied
      {
        source: '/:path*',
        has: [
          {
            type: 'header',
            key: 'x-authorized',
            value: '(?<authorized>yes|true)',
          },
        ],
        destination: '/home?authorized=:authorized',
      },
      // if the host is `example.com`,
      // this rewrite will be applied
      {
        source: '/:path*',
        has: [
          {
            type: 'host',
            value: 'example.com',
          },
        ],
        destination: '/another-page',
      },
    ]
  },
}
```

Rewriting to an external URL

▶ **Examples**

Rewrites allow you to rewrite to an external url. This is especially useful for incrementally adopting Next.js. The following is an example rewrite for redirecting the `/blog` route of your main app to an external site.

```
module.exports = {
  async rewrites() {
    return [
      {
        source: '/blog',
        destination: 'https://example.com/blog',
      },
      {
        source: '/blog/:slug',
        destination: 'https://example.com/blog/:slug', // Matched parameters can be
used in the destination
      },
    ]
```

```
  },
}
```

If you're using `trailingSlash: true`, you also need to insert a trailing slash in the `source` parameter. If the destination server is also expecting a trailing slash it should be included in the `destination` parameter as well.

```
module.exports = {
  trailingSlash: true,
  async rewrites() {
    return [
      {
        source: '/blog/',
        destination: 'https://example.com/blog/',
      },
      {
        source: '/blog/:path*/',
        destination: 'https://example.com/blog/:path*/',
      },
    ]
  },
}
```

Incremental adoption of Next.js

You can also have Next.js fall back to proxying to an existing website after checking all Next.js routes.

This way you don't have to change the rewrites configuration when migrating more pages to Next.js

```
module.exports = {
  async rewrites() {
    return {
      fallback: [
        {
          source: '/:path*',
          destination: `https://custom-routes-proxying-endpoint.vercel.app/:path*`,
        },
      ],
    }
  },
}
```

See additional information on incremental adoption in the docs here.

Rewrites with basePath support

When leveraging `basePath` support with rewrites each `source` and `destination` is automatically prefixed with the `basePath` unless you add `basePath: false` to the rewrite:

```
module.exports = {
  basePath: '/docs',
```

```
  async rewrites() {
    return [
      {
        source: '/with-basePath', // automatically becomes /docs/with-basePath
        destination: '/another', // automatically becomes /docs/another
      },
      {
        // does not add /docs to /without-basePath since basePath: false is set
        // Note: this can not be used for internal rewrites e.g. `destination:
'/another'`
        source: '/without-basePath',
        destination: 'https://example.com',
        basePath: false,
      },
    ]
  },
}
```

Rewrites with i18n support

When leveraging `i18n` support with rewrites each `source` and `destination` is automatically prefixed to handle the configured `locales` unless you add `locale: false` to the rewrite. If `locale: false` is used you must prefix the `source` and `destination` with a locale for it to be matched correctly.

```
module.exports = {
  i18n: {
    locales: ['en', 'fr', 'de'],
    defaultLocale: 'en',
  },

  async rewrites() {
    return [
      {
        source: '/with-locale', // automatically handles all locales
        destination: '/another', // automatically passes the locale on
      },
      {
        // does not handle locales automatically since locale: false is set
        source: '/nl/with-locale-manual',
        destination: '/nl/another',
        locale: false,
      },
      {
        // this matches '/' since `en` is the defaultLocale
        source: '/en',
        destination: '/en/another',
        locale: false,
      },
      {
        // this gets converted to /(en|fr|de)/(.*) so will not match the top-level
        // `/` or `/fr` routes like /:path* would
```

```
        source: '/(.*)',
        destination: '/another',
      },
    ]
  },
}
```