redirects

Redirects allow you to redirect an incoming request path to a different destination path.

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

next.config.js

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

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

- `source` is the incoming request path pattern.
- `destination` is the path you want to route to.
- `permanent` `true` or `false` - if `true` will use the 308 status code which instructs clients/search engines to cache the redirect forever, if `false` will use the 307 status code which is temporary and is not cached.

Why does Next.js use 307 and 308? Traditionally a 302 was used for a temporary redirect, and a 301 for a permanent redirect, but many browsers changed the request method of the redirect to `GET`, regardless of the original method. For example, if the browser made a request to `POST /v1/users` which returned status code `302` with location `/v2/users`, the subsequent request might be `GET /v2/users` instead of the expected `POST /v2/users`. Next.js uses the 307 temporary redirect, and 308 permanent redirect status codes to explicitly preserve the request method used.

- `basePath`: `false` or `undefined` - if false the `basePath` won't be included when matching, can be used for external redirects 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.
- `missing` is an array of missing objects with the `type`, `key` and `value` properties.

Redirects are checked before the filesystem which includes pages and `/public` files.

Redirects are not applied to client-side routing (`Link`, `router.push`), unless Middleware is present and matches the path.

When a redirect is applied, any query values provided in the request will be passed through to the redirect destination. For example, see the following redirect configuration:

```
{
  source: '/old-blog/:path*',
  destination: '/blog/:path*',
  permanent: false
}
```

When `/old-blog/post-1?hello=world` is requested, the client will be redirected to `/blog/post-1?hello=world`.

Path Matching

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

next.config.js

```
module.exports = {
  async redirects() {
    return [
      {
        source: '/old-blog/:slug',
        destination: '/news/:slug', // Matched parameters can be used in the destination
        permanent: true,
      },
    ];
  },
};
```

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`:

next.config.js

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

Regex Path Matching

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

next.config.js

```
module.exports = {
  async redirects() {
    return [
      {
        source: '/post/:slug(\\d{1,})',
        destination: '/news/:slug', // Matched parameters can be used in the destination
        permanent: false,
      },
    ];
  },
};
```

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:

next.config.js

```
module.exports = {
  async redirects() {
    return [
      {
```

```
        // this will match `/english(default)/something` being requested
        source: '/english\\(default\\)/:slug',
        destination: '/en-us/:slug',
        permanent: false,
      },
    ];
  },
};
```

Header, Cookie, and Query Matching

To only match a redirect when header, cookie, or query values also match the `has` field or don't match the `missing` field can be used. Both the `source` and all `has` items must match and all `missing` items must not match for the redirect to be applied.

`has` and `missing` items can 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`.

next.config.js

```
module.exports = {
  async redirects() {
    return [
      // if the header `x-redirect-me` is present,
      // this redirect will be applied
      {
        source: '/:path((?!another-page$).*)',
        has: [
          {
            type: 'header',
            key: 'x-redirect-me',
          },
        ],
        permanent: false,
        destination: '/another-page',
      },
      // if the header `x-dont-redirect` is present,
      // this redirect will NOT be applied
      {
        source: '/:path((?!another-page$).*)',
```

```
      missing: [
        {
          type: 'header',
          key: 'x-do-not-redirect',
        },
      ],
      permanent: false,
      destination: '/another-page',
    },
    // if the source, query, and cookie are matched,
    // this redirect 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',
        },
      ],
      permanent: false,
      destination: '/another/:path*',
    },
    // if the header `x-authorized` is present and
    // contains a matching value, this redirect will be applied
    {
      source: '/',
      has: [
        {
          type: 'header',
          key: 'x-authorized',
          value: '(?<authorized>yes|true)',
        },
      ],
      permanent: false,
      destination: '/home?authorized=:authorized',
    },
    // if the host is `example.com`,
    // this redirect will be applied
    {
      source: '/:path((?!another-page$).*)',
      has: [
        {
          type: 'host',
          value: 'example.com',
        },
      ],
      permanent: false,
```

```
      destination: '/another-page',
    },
  ];
},
};
```

Redirects with basePath support

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

next.config.js

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

  async redirects() {
    return [
      {
        source: '/with-basePath', // automatically becomes /docs/with-basePath
        destination: '/another', // automatically becomes /docs/another
        permanent: false,
      },
      {
        // does not add /docs since basePath: false is set
        source: '/without-basePath',
        destination: 'https://example.com',
        basePath: false,
        permanent: false,
      },
    ];
  },
};
```

Redirects with i18n support

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

next.config.js

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

```

```
  async redirects() {
    return [
      {
        source: '/with-locale', // automatically handles all locales
        destination: '/another', // automatically passes the locale on
        permanent: false,
      },
      {
        // does not handle locales automatically since locale: false is set
        source: '/nl/with-locale-manual',
        destination: '/nl/another',
        locale: false,
        permanent: false,
      },
      {
        // this matches '/' since `en` is the defaultLocale
        source: '/en',
        destination: '/en/another',
        locale: false,
        permanent: false,
      },
      // it's possible to match all locales even when locale: false is set
      {
        source: '/:locale/page',
        destination: '/en/newpage',
        permanent: false,
        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',
        permanent: false,
      },
    ]
  },
}
```

In some rare cases, you might need to assign a custom status code for older HTTP Clients to properly redirect. In these cases, you can use the `statusCode` property instead of the `permanent` property, but not both. To to ensure IE11 compatibility, a `Refresh` header is automatically added for the 308 status code.

Other Redirects

- Inside API Routes, you can use `res.redirect()`.
- Inside `getStaticProps` and `getServerSideProps`, you can redirect specific pages at request-time.

Version History

Version	Changes
`v13.3.0`	`missing` added.
`v10.2.0`	`has` added.
`v9.5.0`	`redirects` added.