# Purell

Purell is a tiny Go library to normalize URLs. It returns a pure URL. Pure-ell. Sanitizer and all. Yeah, I know...

Based on the wikipedia paper and the RFC 3986 document.

build passing

## Install

```
go get github.com/PuerkitoBio/purell
```

## Changelog

- **v1.1.1** : Fix failing test due to Go1.12 changes (thanks to @ianlancetaylor).
- **2016-11-14 (v1.1.0)** : IDN: Conform to RFC 5895: Fold character width (thanks to @beeker1121).
- **2016-07-27 (v1.0.0)** : Normalize IDN to ASCII (thanks to @zenovich).
- **2015-02-08** : Add fix for relative paths issue (PR #5) and add fix for unnecessary encoding of reserved characters (see issue #7).
- **v0.2.0** : Add benchmarks, Attempt IDN support.
- **v0.1.0** : Initial release.

## Examples

From `example_test.go` (note that in your code, you would import "github.com/PuerkitoBio/purell", and would prefix references to its methods and constants with "purell."):

```
package purell

import (
  "fmt"
  "net/url"
)

func ExampleNormalizeURLString() {
  if normalized, err :=
NormalizeURLString("hTTp://someWEBsite.com:80/Amazing%3f/url/",
    FlagLowercaseScheme|FlagLowercaseHost|FlagUppercaseEscapes); err != nil {
    panic(err)
  } else {
    fmt.Print(normalized)
  }
  // Output: http://somewebsite.com:80/Amazing%3F/url/
}

func ExampleMustNormalizeURLString() {
  normalized :=
MustNormalizeURLString("hTTpS://someWEBsite.com:443/Amazing%fa/url/",
    FlagsUnsafeGreedy)
  fmt.Print(normalized)
```

```
  // Output: http://somewebsite.com/Amazing%FA/url
}

func ExampleNormalizeURL() {
  if u, err := url.Parse("Http://SomeUrl.com:8080/a/b/.././c///g?
c=3&a=1&b=9&c=0#target"); err != nil {
    panic(err)
  } else {
    normalized := NormalizeURL(u,
FlagsUsuallySafeGreedy|FlagRemoveDuplicateSlashes|FlagRemoveFragment)
    fmt.Print(normalized)
  }

  // Output: http://someurl.com:8080/a/c/g?c=3&a=1&b=9&c=0
}
```

## API

As seen in the examples above, purell offers three methods, `NormalizeURLString(string, NormalizationFlags) (string, error)`, `MustNormalizeURLString(string, NormalizationFlags) (string)` and `NormalizeURL(*url.URL, NormalizationFlags) (string)`. They all normalize the provided URL based on the specified flags. Here are the available flags:

```
const (
    // Safe normalizations
    FlagLowercaseScheme           NormalizationFlags = 1 << iota // HTTP://host ->
http://host, applied by default in Go1.1
    FlagLowercaseHost                                            // http://HOST ->
http://host
    FlagUppercaseEscapes                                         // http://host/t%ef
-> http://host/t%EF
    FlagDecodeUnnecessaryEscapes                                 // http://host/t%41
-> http://host/tA
    FlagEncodeNecessaryEscapes                                   // http://host/!"#$
-> http://host/%21%22#$
    FlagRemoveDefaultPort                                        // http://host:80 -
> http://host
    FlagRemoveEmptyQuerySeparator                                //
http://host/path? -> http://host/path

    // Usually safe normalizations
    FlagRemoveTrailingSlash // http://host/path/ -> http://host/path
    FlagAddTrailingSlash    // http://host/path -> http://host/path/ (should choose
only one of these add/remove trailing slash flags)
    FlagRemoveDotSegments   // http://host/path/./a/b/../c -> http://host/path/a/c

    // Unsafe normalizations
    FlagRemoveDirectoryIndex   // http://host/path/index.html -> http://host/path/
    FlagRemoveFragment         // http://host/path#fragment -> http://host/path
```

```
    FlagForceHTTP              // https://host -> http://host
    FlagRemoveDuplicateSlashes // http://host/path//a///b -> http://host/path/a/b
    FlagRemoveWWW              // http://www.host/ -> http://host/
    FlagAddWWW                 // http://host/ -> http://www.host/ (should choose
only one of these add/remove WWW flags)
    FlagSortQuery              // http://host/path?c=3&b=2&a=1&b=1 ->
http://host/path?a=1&b=1&b=2&c=3

    // Normalizations not in the wikipedia article, required to cover tests cases
    // submitted by jehiah
    FlagDecodeDWORDHost           // http://1113982867 -> http://66.102.7.147
    FlagDecodeOctalHost           // http://0102.0146.07.0223 -> http://66.102.7.147
    FlagDecodeHexHost             // http://0x42660793 -> http://66.102.7.147
    FlagRemoveUnnecessaryHostDots // http://.host../path -> http://host/path
    FlagRemoveEmptyPortSeparator  // http://host:/path -> http://host/path

    // Convenience set of safe normalizations
    FlagsSafe NormalizationFlags = FlagLowercaseHost | FlagLowercaseScheme |
FlagUppercaseEscapes | FlagDecodeUnnecessaryEscapes | FlagEncodeNecessaryEscapes |
FlagRemoveDefaultPort | FlagRemoveEmptyQuerySeparator

    // For convenience sets, "greedy" uses the "remove trailing slash" and "remove
www. prefix" flags,
    // while "non-greedy" uses the "add (or keep) the trailing slash" and "add www.
prefix".

    // Convenience set of usually safe normalizations (includes FlagsSafe)
    FlagsUsuallySafeGreedy    NormalizationFlags = FlagsSafe |
FlagRemoveTrailingSlash | FlagRemoveDotSegments
    FlagsUsuallySafeNonGreedy NormalizationFlags = FlagsSafe | FlagAddTrailingSlash
| FlagRemoveDotSegments

    // Convenience set of unsafe normalizations (includes FlagsUsuallySafe)
    FlagsUnsafeGreedy    NormalizationFlags = FlagsUsuallySafeGreedy |
FlagRemoveDirectoryIndex | FlagRemoveFragment | FlagForceHTTP |
FlagRemoveDuplicateSlashes | FlagRemoveWWW | FlagSortQuery
    FlagsUnsafeNonGreedy NormalizationFlags = FlagsUsuallySafeNonGreedy |
FlagRemoveDirectoryIndex | FlagRemoveFragment | FlagForceHTTP |
FlagRemoveDuplicateSlashes | FlagAddWWW | FlagSortQuery

    // Convenience set of all available flags
    FlagsAllGreedy    = FlagsUnsafeGreedy | FlagDecodeDWORDHost |
FlagDecodeOctalHost | FlagDecodeHexHost | FlagRemoveUnnecessaryHostDots |
FlagRemoveEmptyPortSeparator
    FlagsAllNonGreedy = FlagsUnsafeNonGreedy | FlagDecodeDWORDHost |
FlagDecodeOctalHost | FlagDecodeHexHost | FlagRemoveUnnecessaryHostDots |
FlagRemoveEmptyPortSeparator
)
```

For convenience, the set of flags `FlagsSafe`, `FlagsUsuallySafe[Greedy|NonGreedy]`, `FlagsUnsafe[Greedy|NonGreedy]` and `FlagsAll[Greedy|NonGreedy]` are provided for the similarly

grouped normalizations on [wikipedia's URL normalization page](). You can add (using the bitwise OR `|` operator) or remove (using the bitwise AND NOT `&^` operator) individual flags from the sets if required, to build your own custom set.

The [full godoc reference is available on gopkgdoc]().

Some things to note:

- `FlagDecodeUnnecessaryEscapes`, `FlagEncodeNecessaryEscapes`, `FlagUppercaseEscapes` and `FlagRemoveEmptyQuerySeparator` are always implicitly set, because internally, the URL string is parsed as an URL object, which automatically decodes unnecessary escapes, uppercases and encodes necessary ones, and removes empty query separators (an unnecessary `?` at the end of the url). So this operation cannot **not** be done. For this reason, `FlagRemoveEmptyQuerySeparator` (as well as the other three) has been included in the `FlagsSafe` convenience set, instead of `FlagsUnsafe`, where Wikipedia puts it.
- The `FlagDecodeUnnecessaryEscapes` decodes the following escapes (*from* -> *to*): - %24 -> $ - %26 -> & - %2B-%3B -> +,-./0123456789:; - %3D -> = - %40-%5A -> @ABCDEFGHIJKLMNOPQRSTUVWXYZ - %5F -> _ - %61-%7A -> abcdefghijklmnopqrstuvwxyz - %7E -> ~
- When the `NormalizeURL` function is used (passing an URL object), this source URL object is modified (that is, after the call, the URL object will be modified to reflect the normalization).
- The *replace IP with domain name* normalization (`http://208.77.188.166/` → `http://www.example.com/`) is obviously not possible for a library without making some network requests. This is not implemented in purell.
- The *remove unused query string parameters* and *remove default query parameters* are also not implemented, since this is a very case-specific normalization, and it is quite trivial to do with an URL object.

### Safe vs Usually Safe vs Unsafe

Purell allows you to control the level of risk you take while normalizing an URL. You can aggressively normalize, play it totally safe, or anything in between.

Consider the following URL:

`HTTPS://www.RooT.com/toto/t%45%1f///a/./b/../c/?z=3&w=2&a=4&w=1#invalid`

Normalizing with the `FlagsSafe` gives:

`https://www.root.com/toto/tE%1F///a/./b/../c/?z=3&w=2&a=4&w=1#invalid`

With the `FlagsUsuallySafeGreedy`:

`https://www.root.com/toto/tE%1F///a/c?z=3&w=2&a=4&w=1#invalid`

And with `FlagsUnsafeGreedy`:

`http://root.com/toto/tE%1F/a/c?a=4&w=1&w=2&z=3`

## TODOs

- Add a class/default instance to allow specifying custom directory index names? At the moment, removing directory index removes `(^|/)((?:default|index)\.\w{1,4})$`.

## Thanks / Contributions

@rogpeppe @jehiah @opennota @pchristopher1275 @zenovich @beeker1121

## License