Generating ReStructured Text Docs For Your Own cobra.Command

Generating ReST pages from a cobra command is incredibly easy. An example is as follows:

```
package main

import (
    "log"

    "github.com/spf13/cobra"
    "github.com/spf13/cobra/doc"
)

func main() {
    cmd := &cobra.Command{
        Use:   "test",
        Short: "my test program",
    }
    err := doc.GenReSTTree(cmd, "/tmp")
    if err != nil {
        log.Fatal(err)
    }
}
```

That will get you a ReST document `/tmp/test.rst`

Generate ReST docs for the entire command tree

This program can actually generate docs for the kubectl command in the kubernetes project

```
package main

import (
    "log"
    "io/ioutil"
    "os"

    "k8s.io/kubernetes/pkg/kubectl/cmd"
    cmdutil "k8s.io/kubernetes/pkg/kubectl/cmd/util"

    "github.com/spf13/cobra/doc"
)

func main() {
    kubectl := cmd.NewKubectlCommand(cmdutil.NewFactory(nil), os.Stdin,
ioutil.Discard, ioutil.Discard)
    err := doc.GenReSTTree(kubectl, "./")
    if err != nil {
```

```
        log.Fatal(err)
    }
}
```

This will generate a whole series of files, one for each command in the tree, in the directory specified (in this case "./")

Generate ReST docs for a single command

You may wish to have more control over the output, or only generate for a single command, instead of the entire command tree. If this is the case you may prefer to `GenReST` instead of `GenReSTTree`

```
    out := new(bytes.Buffer)
    err := doc.GenReST(cmd, out)
    if err != nil {
        log.Fatal(err)
    }
```

This will write the ReST doc for ONLY "cmd" into the out, buffer.

Customize the output

Both `GenReST` and `GenReSTTree` have alternate versions with callbacks to get some control of the output:

```
func GenReSTTreeCustom(cmd *Command, dir string, filePrepender func(string) string,
linkHandler func(string, string) string) error {
    //...
}
```

```
func GenReSTCustom(cmd *Command, out *bytes.Buffer, linkHandler func(string, string)
string) error {
    //...
}
```

The `filePrepender` will prepend the return value given the full filepath to the rendered ReST file. A common use case is to add front matter to use the generated documentation with Hugo:

```
const fmTemplate = `---
date: %s
title: "%s"
slug: %s
url: %s
---
`
filePrepender := func(filename string) string {
    now := time.Now().Format(time.RFC3339)
    name := filepath.Base(filename)
    base := strings.TrimSuffix(name, path.Ext(name))
    url := "/commands/" + strings.ToLower(base) + "/"
    return fmt.Sprintf(fmTemplate, now, strings.Replace(base, "_", " ", -1), base,
```

```
url)
}
```

The `linkHandler` can be used to customize the rendered links to the commands, given a command name and reference. This is useful while converting rst to html or while generating documentation with tools like Sphinx where `:ref:` is used:

```
// Sphinx cross-referencing format
linkHandler := func(name, ref string) string {
    return fmt.Sprintf(":ref:`%s <%s>`", name, ref)
}
```