Generating Bash Completions For Your cobra.Command

Please refer to Shell Completions for details.

Bash legacy dynamic completions

For backward compatibility, Cobra still supports its legacy dynamic completion solution (described below). Unlike the `ValidArgsFunction` solution, the legacy solution will only work for Bash shell-completion and not for other shells. This legacy solution can be used along-side `ValidArgsFunction` and `RegisterFlagCompletionFunc()`, as long as both solutions are not used for the same command. This provides a path to gradually migrate from the legacy solution to the new solution.

Note: Cobra's default `completion` command uses bash completion V2. If you are currently using Cobra's legacy dynamic completion solution, you should not use the default `completion` command but continue using your own.

The legacy solution allows you to inject bash functions into the bash completion script. Those bash functions are responsible for providing the completion choices for your own completions.

Some code that works in kubernetes:

```
const (
        bash_completion_func = `__kubectl_parse_get()
{
    local kubectl_output out
    if kubectl_output=$(kubectl get --no-headers "$1" 2>/dev/null); then
        out=($(echo "${kubectl_output}" | awk '{print $1}'))
        COMPREPLY=( $( compgen -W "${out[*]}" -- "$cur" ) )
    fi
}

__kubectl_get_resource()
{
    if [[ ${#nouns[@]} -eq 0 ]]; then
        return 1
    fi
    __kubectl_parse_get ${nouns[${#nouns[@]} -1]}
    if [[ $? -eq 0 ]]; then
        return 0
    fi
}

__kubectl_custom_func() {
```

```
    case ${last_command} in
        kubectl_get | kubectl_describe | kubectl_delete | kubectl_stop)
            __kubectl_get_resource
            return
            ;;
        *)
            ;;
    esac
}
`)
```

And then I set that in my command definition:

```
cmds := &cobra.Command{
    Use:   "kubectl",
    Short: "kubectl controls the Kubernetes cluster manager",
    Long: `kubectl controls the Kubernetes cluster manager.

Find more information at https://github.com/GoogleCloudPlatform/kubernetes.`,
    Run: runHelp,
    BashCompletionFunction: bash_completion_func,
}
```

The `BashCompletionFunction` option is really only valid/useful on the root command. Doing the above will cause `__kubectl_custom_func()` (`__<command-use>_custom_func()`) to be called when the built in processor was unable to find a solution. In the case of kubernetes a valid command might look something like `kubectl get pod [mypod]`. If you type `kubectl get pod [tab][tab]` the `__kubectl_customc_func()` will run because the cobra.Command only understood "kubectl" and "get." `__kubectl_custom_func()` will see that the cobra.Command is "kubectl_get" and will thus call another helper `__kubectl_get_resource()`. `__kubectl_get_resource` will look at the 'nouns' collected. In our example the only noun will be `pod`. So it will call `__kubectl_parse_get pod`. `__kubectl_parse_get` will actually call out to kubernetes and get any pods. It will then set `COMPREPLY` to valid pods!

Similarly, for flags:

```
    annotation := make(map[string][]string)
    annotation[cobra.BashCompCustom] = []string{"__kubectl_get_namespaces"}

    flag := &pflag.Flag{
        Name:        "namespace",
        Usage:       usage,
        Annotations: annotation,
    }
    cmd.Flags().AddFlag(flag)
```

In addition add the `__kubectl_get_namespaces` implementation in the

BashCompletionFunction value, e.g.:

```
__kubectl_get_namespaces()
{
    local template
    template="{{ range .items  }}{{ .metadata.name }} {{ end }}"
    local kubectl_out
    if kubectl_out=$(kubectl get -o template --template="${template}" namespace 2>/dev/null)
        COMPREPLY=( $( compgen -W "${kubectl_out}[*]" -- "$cur" ) )
    fi
}
```