GitDepend Documentation

Release 0.1.0

Kevin Johnson

Mar 03, 2017

Contents:

Solves the problem of working with multiple git repositories where lower level repositories produce nuget packages that are consumed by other repositories.

CHAPTER 1

Why do I need GitDepend?

I work for a company that produces a lot of in-house nuget packages. Originally we tried maintaining the version of these packages manually. If you've ever tried doing that on a large scale you realize that it quickly gets unmanageable. It's easy to make mistakes with your versioning. It's easy to forget to bump the version. In short, versioning is something that should really be automated.

GitVersion to the rescue! With GitVersion we could just write code, and our packages magically version themselves correctly. BUT, there was still a big problem. We didn't like lower level packages having to bump their version just because another package had to change. GitVersion assumes the same version for the entire repository. So, if you need a unique version so that your code can stabalize on a version you need to split it into another git repository.

We started doing this... a LOT. Before we knew it the setup process to get our full process building involved checking out multiple repositories. Making sure that each one was on the appropriate branch for what we were doing, and writing a complicated build script that lived in the upper most repository to build all other repositories. As we added new repositories that build script was constantly changing, and getting more complicated. We needed a solution that made it easy to chain repositories together. Thus GitDepend was born.

CHAPTER 2

Installing GitDepend.CommandLine

GitDepend is available on nuget.org

```
Install-Package GitDepend.CommandLine
```

CHAPTER 3

Installing GitDepend.Portable

GitDepend is available on chocolatey.org

```
choco install GitDepend.Portable
```

CHAPTER 4

Command Line Options

```
branch      List, create, or delete branches

checkout    Switch branches

clone       Recursively clones all dependencies

config      Displays the full configuration file

init        Assists you in creating a GitDepend.json

list        Lists all repository dependencies

status      Displays git status on dependencies

sync        Sets the referenced branch to the currently checked out branch on
            dependencies.

update      Recursively builds all dependencies, and updates the current
            project to the newly built artifacts.

help        Display this help screen.
```

branch

```
-d, --delete    (Default: False) Indicates that the specified branch should
                be deleted.

--force         (Default: False) Indicates that the specified branch should
                be delete, regardless of whether it has been merged or not.

--merged        (Default: False) List all merged branches
```

```
--dir           The directory to process. The current working directory will
                be used if this option is ignored.
```

checkout

```
-b, --create    (Default: False) Create a new branch

--dir           The directory to process. The current working directory will
                be used if this option is ignored.
```

clone

Recursively clones all dependencies

```
--dir    The directory to process. The current working directory will be used
         if this option is ignored.
```

config

Shows the configuration that will be used by GitDepend.

```
--dir    The directory to process. The current working directory will be used
         if this option is ignored.
```

init

Launches the configuration wizard. This will assist you to create a GitDepend.json file

```
--dir    The directory to process. The current working directory will be used
         if this option is ignored.
```

list

Lists all dependencies for the current project

```
--dir    The directory to process. The current working directory will be used
         if this option is ignored.
```

status

This displays the status of all the dependencies which you have listed in your config files.

Basically, the same as you running `git status` in all of your directories.

sync

Sync will make sure that your current directory and dependencies are on the same branch.

If they are, nothing will be updated. If your branches are out of sync, you will have a few options to resolve:

1. Resolve the differences in your GitDepend config files.
2. Assume config is correct, and switch to that branch.
3. Abort and take care of it yourself.

update

This is used to update nuget packages across all dependencies.

Run the `sync` command if you get the error of dependencies not being on the correct branch.

```
--dir    The directory to process. The current working directory will be used
         if this option is ignored.
```

CHAPTER 5

Usage Example

In the root of your repository you will include a `GitDepend.json` file

```
{
  "name": "Lib2",
  "build": {
    "script": "make.bat"
  },
  "packages": {
    "dir": "artifacts/NuGet/Debug"
  },
  "dependencies": [
    {
      "url": "git@github.com:GitDepend/Lib1.git",
      "dir": "../Lib1",
      "branch": "develop"
    }
  ]
}
```

Normally if you are working in an upper level repository you should just be able to run the build script and rely on nuget packages. However, when you have changed code in a lower level repository you will need to have those changes cascade up the chain. This is where GitDepend shines. Run the following command

```
GitDepend.exe update
```

This will follow the chain of `GitDepend.json` files. The following things will happen

1. Check out the dependency if it has not been checked out.
2. Ensure that the repository is on the correct branch.
3. update all dependencies (this is a recursive step)
4. consume the latest nuget packages produced by dependency repositories.

At this point the upper level repository should be all up to date, targetting the latest nuget packages and be ready to build.

Try it out!

Take a look at some example projects and try it out for yourself.

- Lib1
- Lib2

Lib2 depends on Lib1

Clone Lib2

```
git clone git@github.com:GitDepend/Lib2.git
```

from the root of Lib2 run

```
make.bat update
```

This will clone and build all dependencies

build it with

```
make.bat
```

Now, make a change in Lib1 and commit that change.

```
make.bat update
```

CHAPTER 6

Indices and tables

- genindex
- search