Gittern Documentation

Release 0.8

E-butik i Norden AB

May 24, 2013

CONTENTS

Making Git like music for PHP's ears.

Contents:

CHAPTER

ONE

USAGE

Contents:

1.1 Using Gittern with the Gaufrette Adapters

1.1.1 GitternCommitishReadOnlyAdapter

Code speaks louder than words:

```
<?php

use Gittern\Repository,
    Gittern\Transport\NativeTransport,
    Gittern\Configurator,
    Gittern\Gaufrette\GitternCommitishReadOnlyAdapter;

use Gaufrette\Filesystem;

$repo = new Repository;
$repo->setTransport(new NativeTransport($repo_path));

$configurator = new Configurator;
$configurator->defaultConfigure($repo);

$filesystem = new Filesystem(new GitternCommitishReadOnlyAdapter($repo, "master"));
```

After this, you can use the filesystem like any other Gaufrette Filesystem. Just bear in mind that it's read-only, and will throw exceptions if you try to modify it.

1.1.2 GitternIndexAdapter

Again, code speaks louder than words:

```
<?php

use Gittern\Repository,
    Gittern\Transport\NativeTransport,
    Gittern\Configurator,
    Gittern\Gaufrette\GitternIndexAdapter;

use Gaufrette\Filesystem;
```

```
$repo = new Repository;
$repo->setTransport(new NativeTransport($repo_path));

$configurator = new Configurator;
$configurator->defaultConfigure($repo);

$filesystem = new Filesystem(new GitternIndexAdapter($repo));
```

After this, you can use the filesystem like any other Gaufrette Filesystem.

Committing

The Git Index contains everything necessary to create a tree. Once you have a tree, creating a commit is a fairly straight-forward deal, but additional convenience is under consideration.

```
<?php

use Gittern\Entity\GitObject\Commit,
    Gittern\Entity\GitObject\User;

use DateTime;

$parent = $repo->getObject('master');

$tree = $repo->getIndex()->createTree();
$commit = new Commit;
$commit->setTree($tree);
$commit->addParent($parent);
$commit->setMessage("Added another file");
$commit->setAuthor(new User("Tessie Testson", "tessie.testson@example.com"));
$commit->setCommitter(new User("Tessie Testson", "tessie.testson@example.com"));
$commit->setAuthorTime(new DateTime);
$commit->setCommitTime(new DateTime);

$repo->desiccateGitObject($commit);
$repo->setBranch('master', $commit);

$repo->flush();
```

1.2 The Low Level interface

This section is very much incomplete. Currently, take a look at the test suite and the code to gain insight into how Gittern works.

The low level interface is where the magic happens. This is also where you'll absolutely need to be familiar with the git model of blobs, trees, commits, and how the fit together. Seriously. If you don't e.g. know what a tree is (in the context of git, of course), probably won't understand how to use this, and theoretically you might be able to break your repos when using the low level interface. Proceed with caution.

If you want to learn how git works, there's plenty of resources. I'd suggest the following, in order:

- Think like (a) Git
- The Internals and Plumbing chapters in the Git Community Book

CHAPTER

TWO

TECHNICAL DOCS

Contents:

2.1 What does the different kind of classes do?

2.1.1 Entities

The Git object model has quite a few entities, and Gittern divides them into two categories. GitObjects and "other". Any kind of entity which is persisted in the Git object store is a GitObject. That means Commits, trees (and their different node types), blobs and annotated tags (not yet supported). The index and it's entries are the current "other" entities.

All of these objects represent a concept present in the Git object model.

2.1.2 Proxy

When you fetch a Git entity (unless it's a blob), it's probably going to have relations to other entities. To make it easier to work with the objects, Gittern creates proxy objects for these relations. When a method requires data that the proxy object doesn't already have (pretty much anything but the SHA), the proxy lazily loads the data from the repository.

Proxies are decorators of the class they're proxying, and thus will pass a type check.

2.1.3 Hydrator

In a normal git repository, the files are stored according to a certain file specification. The role of the hydrator is to a RawObject (which is basically just the file data and it's sha), and create an entity from it.

2.1.4 Desiccator

A de-hydrator. Where a hydrator takes a RawObject and creates an entity from it, the desiccator takes and entity and creates a RawObject from it.

2.1.5 Transport

In order for Gittern to be modular, the Repository class doesn't actually know how to read and write your RawObjects et c. from/to the disk. Maybe you don't even care that much about keeping compatibility with the git binary, and want to store your objects somewhere else. Maybe you want to cache them in e.g. Redis.

For this reason there's the Transport. The Transport knows all about how to get your objects from the disk, how to resolve references, et c.

2.1.6 Adapters

Due to it's fine representation of the Git object model, Gittern is a breeze to work with. However, sometimes you don't actually care about the Git object model. Sometimes you just want to treat it like a filesystem. Lucky for us the fine folks at KnpLabs have created Gaufrette, a file system abstraction layer. Gittern has two Gaufrette adapters, allowing you to treat a git repository like any other file system.

CHAPTER

THREE

WHAT IS GITTERN

Gittern is a PHP library for reading from and writing to Git repositories. It doesn't depend on the `git` binary, it directly acccesses the repo files.

Gittern provides several interfaces for interacting with your Git repos. Firstly, there's a low level interface, where you manually create blobs, trees and commits. This can however become cumbersome very quickly. As such, there's also two different Gaufrette adapters included.

The first one, `GitternCommitishReadOnlyAdapter` is pretty much what it sounds like. It's an adapter to which you supply a commitish, and from which you may read the files associated with that commit.

The second one, `GitternIndexAdapter` allows you to read from and write to the Git index. The Git index is the staging area in which you stage new changes for a commit. Then, when you're ready, creating a commit from the index is quite simple. Just get the tree from the index, and use it to create your commit.

In addition to all of this functionality, Gittern is created with extensibility in mind. How Gittern reads data from the repository on disk is cleanly separated using the adapter pattern, so that if you have the need you could easily implement e.g. an adapter capable of maintaining multiple indexes, or Git object caching in something like MongoDB or Redis (in fact, if you don't care about accessing the repo from the git binary, you could just use this as a fast, distributed backend).

CHAPTER

FOUR

INSTALLATION

Gittern is installed via Composer. It's available on Packagist as `e-butik/gittern`.

Create composer.json file in the project root:

```
{
    "require": {
        "e-butik/gittern": "*"
    }
}
```

Then download composer.phar and run the install command:

```
$ wget --quiet http://getcomposer.org/composer.phar
$ php composer.phar install
```

If you want to be able to run the test suite, add `--install-suggests` to the install command.