Pharo 9 by Example

Stéphane Ducasse, Sebastijan Kaplar, Gordana Rakic and Quentin Ducasse

July 27, 2021

Layout and typography based on the sbabook LaTeX class by Damien Pollet.

Contents

Illustrations

CHAPTER 1

Publishing and packaging your first Pharo project

In this chapter we explain how you can publish your project on Github using Iceberg. Iceberg is a tool and library that supports the handling of git and it does more than just saving files and publishing them to a git repository. Then we will briefly show how to make sure that you or other developers can load your code without having to understand the internal dependencies.

We will not explain basic concepts like commit, push/pull, merging, or cloning, please refer to a git tutorial for this. A strong precondition before reading this chapter is that you must be able to publish from the command line to the `git` hosting service that you want to use. If you cannot, do not expect Iceberg to fix it magically for you. Now if you have some problems with SSH configuration (which is the default with Github) you can either use HTTPS or have a look in *Manage your code with Iceberg* booklet that you can find on http://books.pharo.org. Let us get started.

1.1 For the impatient

If you do not want to read everything, here is an executive summary of how to get your code published:

- Create a project on Github or any git-based platform.
- [Optional] Configure Iceberg to use custom ssh keys.
- Add a project in Iceberg.

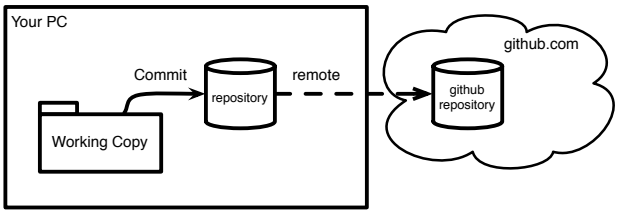

Figure 1-1 Git: A distributed versioning system.

  - Optionally but strongly recommanded, in the cloned repository, create a directory named `src` on your file system. This is a good convention.

- In Iceberg, open your project and add your packages.
- Commit your project.
- [Optional] Add a baseline to ease loading your project.
- Push your change to your remote repository.

You are done. Now we can explain such steps calmly.

1.2 Basic Architecture

As `git` is a distributed versioning system, you need a *local* clone of the repository and a *working copy*. Your working copy and local repository are usually on your machine. This is to this local repository that your changes will be commited to before being pushed to remote repositories (Figure 1-1). With Pharo, the situation is a bit more complex and Iceberg is hiding the extra complexity for us. In a nutshell, in Pharo classes and methods are objects that get modified on the fly. When you modify a class source code, the git working copy is not automatically modified. It is like if in Pharo you get two working copies: the object one and the git-file based one. The job of Iceberg is to make sure both work well together and are synchronisez and that you do not have to worry about it.

Create a new project on Github

While you can save locally first and then later create a remote repository, we first create a new project on Github – we explain later the other scenario. Figure 1-2 shows the creation of a project on Github. The order does not really matter. What is different is that you should use different options when add a repository to Iceberg as we will show later.

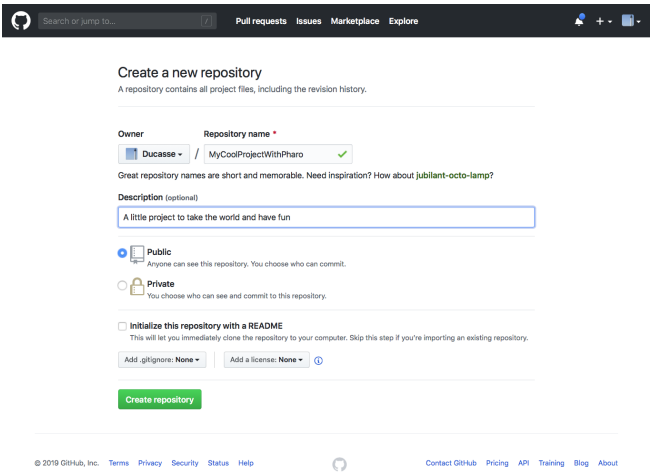

Figure 1-2 Create a new project on Github.

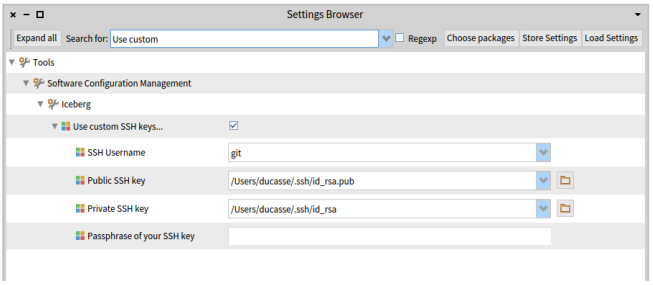

Figure 1-3 Use Custom SSH keys settings.

[Optional] SSH setup: Tell Iceberg to use your keys

To be able to commit to your `git` project, you should either use HTTPS or you will need to set up valid credentials in your system. In case you use SSH (the default way), you will need to make sure those keys are available to your Github account and also that the shell adds them for smoother communication with the server.

Go to settings browser, search for "Use custom SSH keys" and enter your data there as shown in Figure 1-3).

Alternatively, you can execute the following expressions in your image playground or add them to your Pharo system preference file (See Menu System item startup):

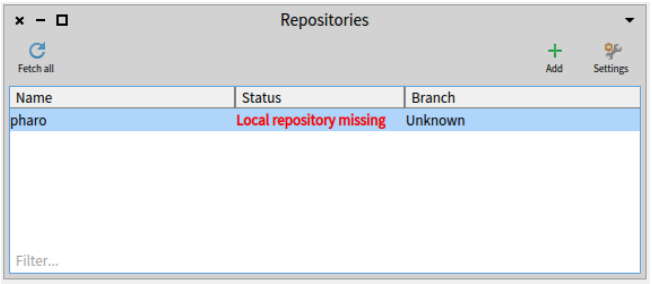

Figure 1-4 Iceberg *Repositories* browser on a fresh image indicates that if you want to version modifications to Pharo itself you will have to tell Iceberg where the Pharo clone is located. But you do not care.

```
IceCredentialsProvider useCustomSsh: true.
IceCredentialsProvider sshCredentials
  publicKey: 'path\to\ssh\id_rsa.pub';
  privateKey: 'path\to\ssh\id_rsa'
```

> **Note** **Pro Tip:** this can be used too in case you have a non-default key file. You just need to replace `id_rsa` with your file name.

Now we are ready to have a look at Iceberg the layer managing git in Pharo.

1.3 Iceberg *Repositories* browser

Figure 1-4 shows the top level Iceberg pane. It shows that for now you do not have defined nor loaded any project. It shows the Pharo project and indicates that it could not find its local repository by displaying 'Local repository missing'.

First you do not have to worry about the Pharo project or repository if you do not want to contribute to Pharo. So just go ahead. Now if you want to understand what is happening here is the explanation:s The Pharo system does not have any idea where it should look for the `git` repository corresponding to the source of the classes it contains. Indeed, the image you are executing may have been built somewhere, patched or not many times. Now Pharo is fully operational without having a local repository. You can browse system classes and methods because Pharo has its own internal source management. This warning just indicates that if you want to version Pharo system code using `git` then you should indicate to the system where the clone and working copy are located on your local machine. So if you do not plan to modify and version the Pharo system code, you do not have to worry.

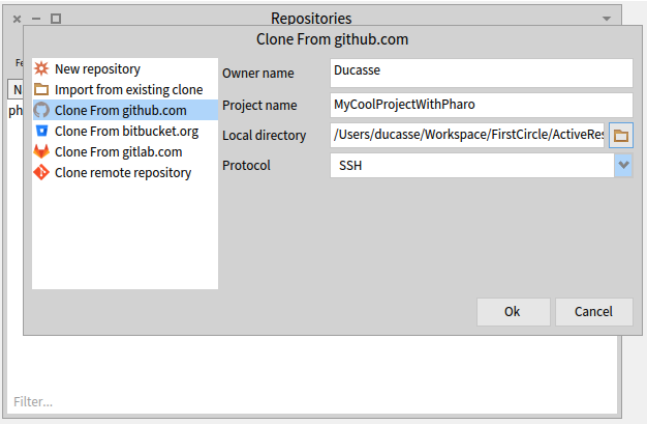

Figure 1-5 Cloning a project hosted on Github via SSH.

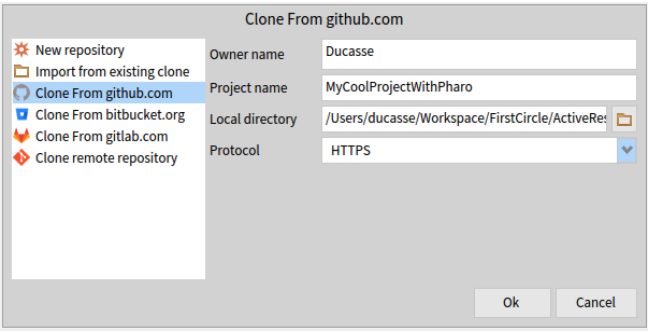

Figure 1-6 Cloning a project hosted on Github via HTTPS.

1.4 Add a new project to Iceberg

The first step is then to add a project to Iceberg:

- Press the '+' button to the right of the Iceberg main window.
- Select the source of your project. In our example, since you did not clone your project yet, choose the Github option.

Notice that you can either use SSH (Figure 1-5) or HTTPS (Figure 1-6).

Figure 1-5 and 1-6) instruct Iceberg to clone the repository we just created on Github. We specify the owner, project, and physical location where the local clone and `git` working copy will be on your disk.

Iceberg has now added your project to its list of managed projects and cloned an empty repository to your disk. You will see the status of your project, as in Figure 1-7. Here is a breakdown of what you are seeing:

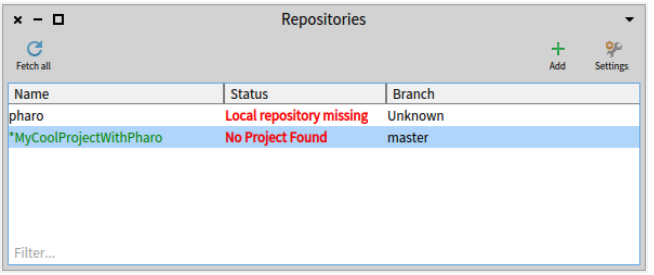

Figure 1-7 Just after cloning an empty project, Iceberg reports that the project is missing information.

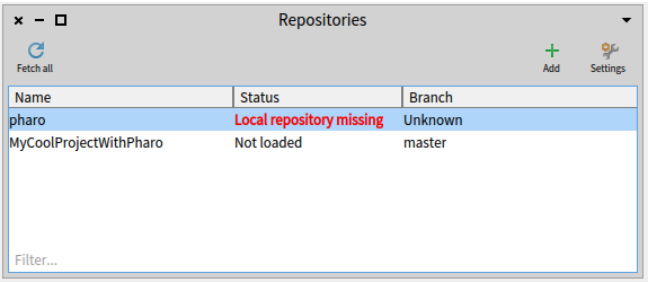

Figure 1-8 Adding a project with some contents shows that the project is not loaded - not that it is not found.

- `MyCoolProjectWithPharo` has a star and is green. This usually means that you have changes which haven't been committed yet, but may also happen in unrelated edge cases like this one. Don't worry about this for now.
- The Status of the project is `'No Project Found'` and this is more important. This is normal since the project is empty. Iceberg cannot find its metadata. We will fix this soon.

Later on, when you will have commited changes to your project and you want to load it in another image, when you will clone again, you will see that Iceberg will just report that the project is not loaded as shown in Figure 1-8.

Repair to the rescue

Iceberg is a smart tool that helps you fix the problems you may encounter while working with `git`. As a general principle, each time you get a status with red text (such as "No Project Found" or "Detached Working Copy"), you should ask Iceberg to fix it using the **Repair** command.

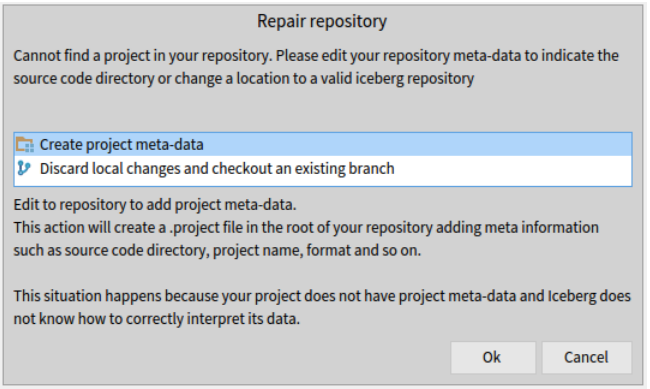

Figure 1-9 Create project metadata action and explanation.

Iceberg cannot solve all situations automatically, but it will propose and explain possible repair actions. The actions are ranked from most to least likely to be right one. Each action has a displayed explanation on the situation and the consequences of using it. It is always a good idea to read them. Setting your repository the right way makes it extremely hard to lose any piece of code with Iceberg and Pharo is general since Pharo contains its own copy of the code.

Create project metadata

Iceberg reported that it could not find the project because some meta data were missing such as the format of the code encodings and the example location inside the repository. When we activate the repair command, we get Figure 1-9. It shows the ”Create project metadata” action and its explanation.

When you choose to create the project metadata, Iceberg shows you the filesystem of your project as well as the repository format as shown in Figure 1-10. Tonel is the preferred format for Pharo projects. It has been designed to be Windows and file system friendly. Change it only if you know what you are doing!

Before accepting the changes, it is a good idea to add a source (`src`) folder to your repository. Do that by pressing the + icon. You will be prompted to specify the folder for code as shown in Figure 1-11. Do not forget to select the `src` folder once you created it. Iceberg will show you the exact structure of your project as shown in Figure 1-12.

After accepting the project details, Iceberg shows you the files that you will be committing as shown in Figure 1-13.

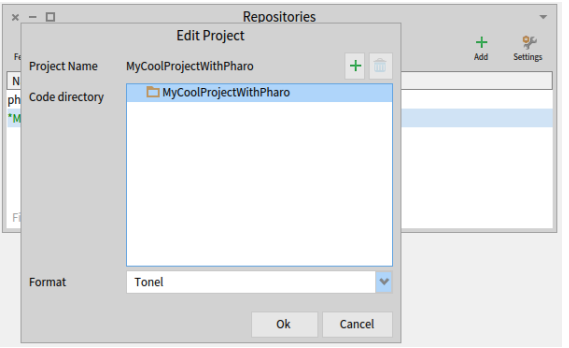

Figure 1-10 Showing where the metadata will be saved and the format encodings.

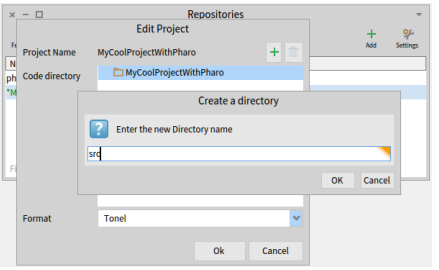

Figure 1-11 Adding a src repository for code storage.

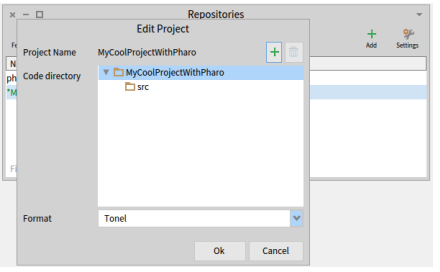

Figure 1-12 Resulting situation with a src folder: Pay attention to select it.

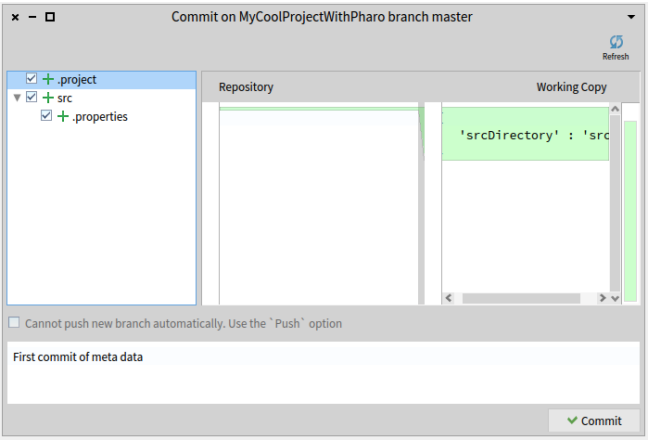

Figure 1-13 Details of metadata commit.

Once you have committed the metadata, Iceberg shows you that your project has been repaired but is not loaded as shown in Figure 1-8. This is normal since we haven't added any packages to our project yet. You can optionally push your changes to your remote repository.

Your local repository is ready, let's move on to the next part.

1.5 Add and commit your package using the *Working copy* browser

Once your project contains Iceberg metadata, Iceberg will be able to manage it easily. Double click on your project to bring a *Working copy* browser for your project. It lists all the packages that compose your project. Right now you have none. Add a package by pressing the + (Add Package) iconic button as shown by Figure 1-14.

Again, Iceberg shows that your package contains changes that are not committed using the green color and the star in front of the package name as showing in Figure 1-15.

Commit the changes

Commit the changes to your local repository using the Commit button as shown in Figure 1-16. Iceberg lets you chose the changed entities you want to commit. Here this is not needed but this is an important feature. Iceberg will show the result of the commit action by removing the star and changing the color. It now shows that the code in the image is in sync with your local repository as shown by Figure 1-17. You can commit several times if needed.

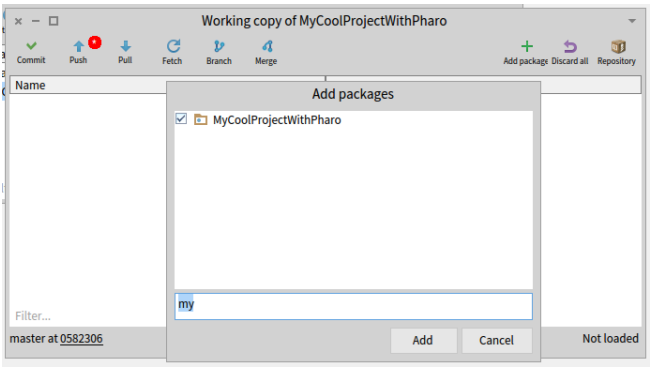

Figure 1-14 Adding a package to your project using the *Working copy* browser.

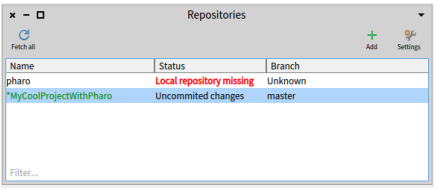

Figure 1-15 Iceberg indicates that your project has unsaved changes – indeed you just added your package.

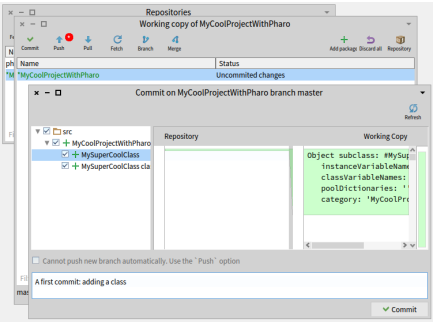

Figure 1-16 When you commit changes, Iceberg shows you the code about to be committed and you can chose the code entities that will effectively be saved.

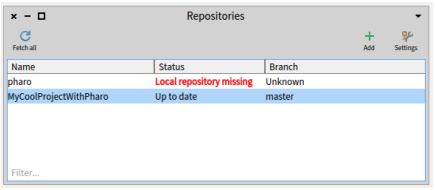

Figure 1-17 Once changes committed, Iceberg reflects that your project is in sync with the code in your local repository.

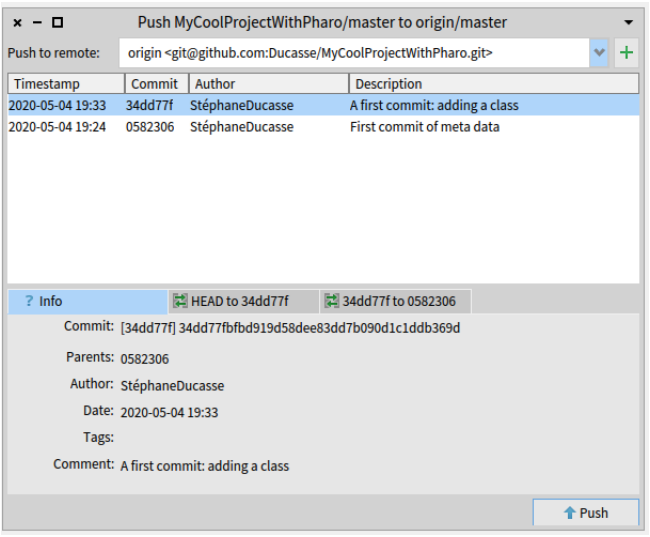

Figure 1-18 Publishing your committed changes.

Publish your changes to your remote

Now you are nearly done. Publish your changes from your local directory to your remote repository using the Push button. You may be prompted for credentials if you used HTTPS.

When you push your changes, Iceberg will show you all the commits awaiting publication and will push them to your remote repository as shown in Figure 1-18.

Now you are basically done. You now know the essential aspects of managing your code with Github. Iceberg has been designed to guide you so please listen to it unless you really know what you are doing. You are now ready to use services offered around Github to improve your code control and quality!

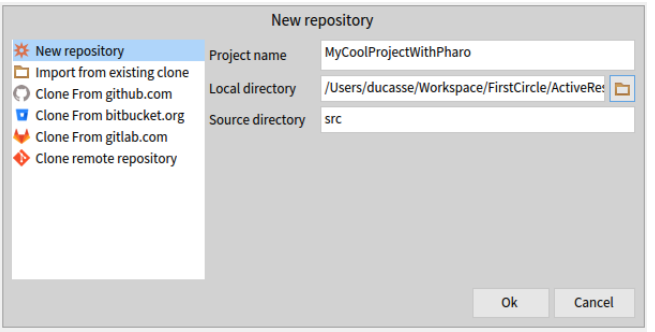

Figure 1-19 Creating a local repository without pre-existing remote repository.

1.6 What if I did not create a remote repository

We started by creating a remote repository on Github. Then we asked Iceberg to add a project by cloning it from Github. Now you may ask yourself what is the process to publish first your project locally without a pre-existing repository. Let us try it. This is actually simple.

Create a new repository

When you add a new repository use the 'New repository' option as shown in 1-19.

Add a remote

If you want to commit to a remote repository, you will have to add it using the *Repository* browser. You can access this browser through the associated menu item or the icon. The *Repository* browser gives you access to the `git` repositories associated with your project: you can access, manage branches and also add or remove remote repositories. Figure 1-21 shows the repository browser on our project.

Pressing on the 'Add remote' iconic button adds a remote by filling the needed information that you can find in your Github project. Figure 1-21 shows it for the sample project using SSH and Figure 1-22 for HTTPS.

Push to the remote

Now you can push your changes and versions to the remote repository using the Push iconic button. Once you have pushed you can see that you have one remote as shown in Figure 1-23.

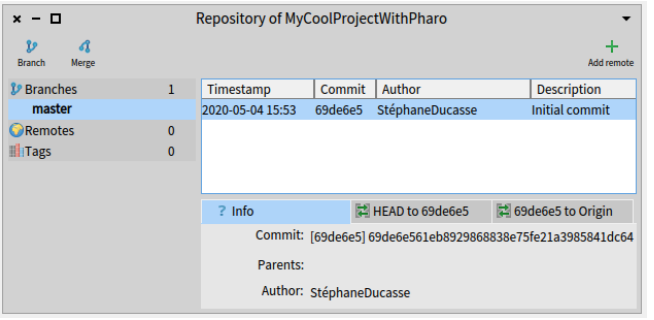

Figure 1-20 Opening the repository browser let you add and browse branches as well as remote repositories.

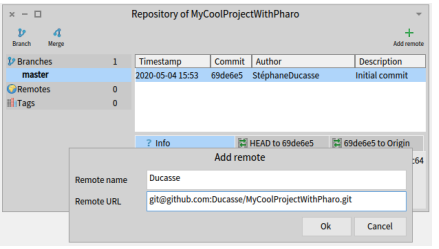

Figure 1-21 Adding a remote using the *Repository* browser of your project (SSH version).

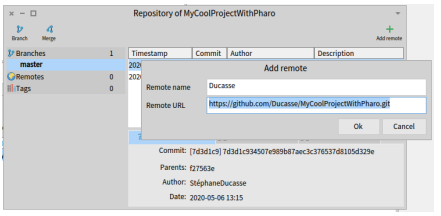

Figure 1-22 Adding a remote using the *Repository* browser of your project (HTTP version).

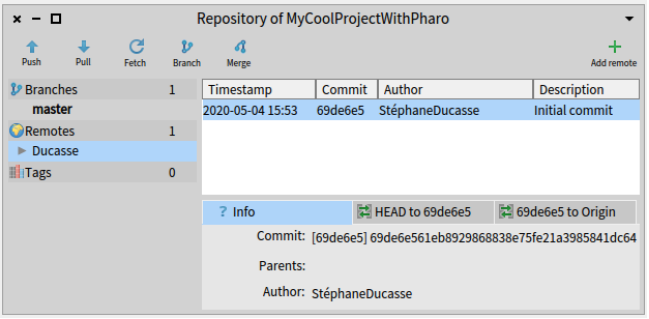

Figure 1-23 Once you pushed you changes to the remote repository.

1.7 Configure your project nicely

Versioning code is just the first part of making sure that you and other developers can reload your code. We will describe how to define a *baseline*: a project map that you will use to define dependencies within your project and dependencies to other projects. In the next chapter we will show how to configure your project to get more out of the services offered within the Github ecosystem such as Github actions to execute automatically your tests.

We start by showing you how you can commit your code if you did not create your remote repository first.

1.8 Defining a `BaselineOf`

A *baseline* is a description of a project architecture. You express the dependencies between your packages and other projects so that all the dependent projects are loaded without the user having to understand them or the links between them.

A baseline is expressed as a subclass of `BaselineOf` and packaged in a package named `'BaselineOfXXX'` (where 'XXX' is the name of your project). So if you have no dependencies, you can have something like this.

```
BaselineOf subclass: #BaselineOfMyCoolProjectWithPharo
  ...
  package: 'BaselineOfMyCoolProjectWithPharo'
```

```
BaselineOfMyCoolProjectWithPharo >> baseline: spec
  <baseline>
  spec
    for: #common
    do: [ spec package: 'MyCoolProjectWithPharo' ]
```

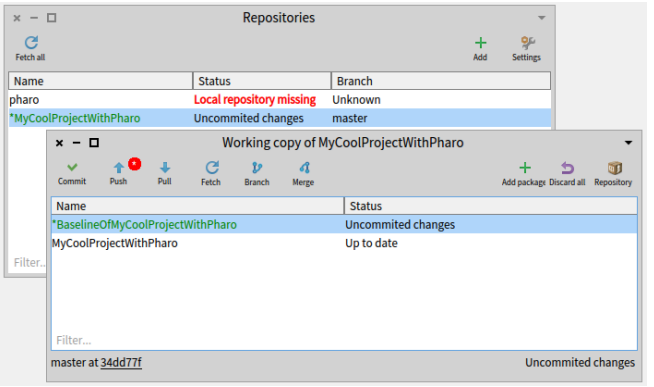

Figure 1-24 Added the baseline package to your project using the *Working copy* browser.

Once you have defined your baseline, you should add its package to your project using the working copy browser as explained in the previous chapter. You should obtain the following situation shown in Figure 1-24. Now, commit it and push your changes to your remote repository.

A more elaborated web resources about baseline possibility is available at: https://github.com/pharo-open-documentation/pharo-wiki/.

1.9 Loading from an existing repository

There are several ways to load your versioned code into a new Pharo image.

Loading baseline using Iceberg

To load a project interactively, you can use the metacello menu item of Iceberg. It lets you load a baseline and execute it to load the project packages. This way you are sure that all the subprojects are loaded.

Manual load

Sometimes you may need to load a given package or you project may not have defined a baseline yet. You can use Iceberg to load a specific package as follows:

- Add the project using Iceberg as we previously explained.
- Open the ”Working copy” browser by double clicking on the project line in the repositories browser.
- Select a package and manually load it.

Scripting the load

The second way is to make use of Metacello scripts.

```
Metacello new
  baseline: 'MyCoolProjectWithPharo';
  repository: 'github://Ducasse/MyCoolProjectWithPharo/src';
  load
```

For projects with metadata, like the one we just created, that's it. Notice that we not only mention the Github path but also added the code folder (here `src`).

1.10 Stepping back...

As `git` is a distributed versioning system, you need a local clone of your repository. In general you edit your working copy located on your hard-drive and you commit to your local clone, and from there you push to remote repositories like Github. We explain here the specificity of managing Pharo with git.

When coding in Pharo, you should understand that you are not directly editing your local working copy, you are modifying objects that represent classes and methods that are living in the Pharo environment. Therefore it is like you have a double working copy: Pharo itself and the `git` working copy.

When you use `git` command lines, you have to understand that there is the code in the image and the code in the working copy (and your local clone).To update your image, you *first* have to update your `git` working copy and *then* load code from the working copy to the image. To save your code you have to save the code to files, add them to your git working copy and commit them to your clone.

Now the interesting part is that Iceberg manages all this for you transparently. All the synchronization between these two working copies is done behind the scene.

The architecture of the system is as follows:

- You have your code in the Pharo image.
- Pharo is acting as a working copy (it contains the contents of the local `git` repository).
- Iceberg manages the publication of your code to the `git` working copy and the `git` local repository.
- Iceberg manages the publication of your code to remote repositories.
- Iceberg manages the re-synchronization of your image with the `git` local repository, `git` remote repositories and the `git` working copy.

1.11 Conclusion

This chapter presented the most important aspects of how to publish and package your code correctly. It will help you to reload it. Now you are ready to take advantage of git-based services, such as the continuous integration of Github Actions.

Bibliography