

# Managing Your Code with Iceberg

Stéphane Ducasse and Guillermo Polito

March 24, 2019

Copyright 2017 by Stéphane Ducasse and Guillermo Polito.

The contents of this book are protected under the Creative Commons Attribution-ShareAlike 3.0 Unported license.

You are **free**:

- to **Share**: to copy, distribute and transmit the work,
- to **Remix**: to adapt the work,

Under the following conditions:

**Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page:  
<http://creativecommons.org/licenses/by-sa/3.0/>

Any of the above conditions can be waived if you get permission from the copyright holder. Nothing in this license impairs or restricts the author's moral rights.



Your fair dealing and other rights are in no way affected by the above. This is a human-readable summary of the Legal Code (the full license):  
<http://creativecommons.org/licenses/by-sa/3.0/legalcode>

# Contents

<b>Illustrations</b>	<b>ii</b>
----------------------	-----------

## **I Managing your code with Iceberg**

<b>1 Publishing your first project</b>	<b>3</b>
1.1 For the impatient . . . . .	3
1.2 Iceberg setup . . . . .	4
1.3 Create a new project on github . . . . .	5
1.4 Add a new project to Iceberg . . . . .	6
1.5 [Optional but strognly suggested] Adding a src directory . . . . .	7
1.6 Repair to the rescue . . . . .	7
1.7 Create project meta-data . . . . .	8
1.8 Adding and committing your package . . . . .	8
1.9 Defining a BaselineOf . . . . .	11
1.10 Loading from an existing repository . . . . .	12
1.11 [Optional] Add a nice .gitignore file . . . . .	12
<b>2 Empowering your projects</b>	<b>15</b>
2.1 Adding Travis integration . . . . .	15
2.2 On windows . . . . .	16
2.3 Adding badges . . . . .	16
<b>3 Understanding the architecture</b>	<b>19</b>
3.1 Glimpse at the architecture . . . . .	19
<b>4 Iceberg Glossary</b>	<b>21</b>
4.1 Git . . . . .	21
4.2 Iceberg . . . . .	23
<b>Bibliography</b>	<b>25</b>

# Illustrations

1-1	Use Custom SSH keys settings. . . . .	5
1-2	Create a new project. . . . .	5
1-3	Cloning new project. . . . .	6
1-4	Just after cloning an empty project. . . . .	7
1-5	Adding an src folder to . . . . .	7
1-6	Create project meta-data action and explanation. . . . .	8
1-7	Showing where the metadata will be saved and the format encodings. . . .	9
1-8	Details of meta data commit. . . . .	9
1-9	The package is clean, meta-data are saved, but it is not loaded. . . . .	10
1-10	Adding a package to your project. . . . .	10
1-11	Publishing your committed changes. . . . .	11
1-12	With a Baseline. . . . .	12
3-1	Create a new project. . . . .	20

## Part I

# Managing your code with Iceberg





# Publishing your first project

In this chapter we explain how you can publish your project on github using Iceberg. We do not explain concepts like commit, push/pull, merging, or cloning. We thank Peter Uhnak for his first blog on publishing Pharo code on Github.

As git is distributed versioning system, you need a local clone of the repository. This is to this local repository that your changes will be committed to before being pushed to remote repositories. In general you commit to your local clone, and from there you push to github or other places. Iceberg will do all the operations and more for you.

## 1.1 For the impatient

If you do not want to read anything, here is the executive summary.

- Create a project on github or any git-based platform.
- Configure Iceberg to use custom ssh keys
- Open Iceberg.
- Add a project (chose clone from ...).
- Optionally, in the cloned repository, create a directory named `src` on your file system using either the `FileList` or your command line.
- Open your project and add your packages (It is always good to add a baseline).
- Commit your project.
- Push it to your remote repository.

You are done. Now we can explain calmly.

## 1.2 Iceberg setup

To be able to commit to your git project, you will need to set a valid set of 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 add them for the smoother communication with the server.

In case they are not (and you will notice as soon as you try to clone a project or commit a change into one), you can add them following these steps (on Windows, if you want a nice command line environment, install [\\*http://mingw.org/wiki/msys](http://mingw.org/wiki/msys)):

### Generating a key pair

To do this execute the command:

```
[ ssh-keygen -t rsa
```

It will generate a private and a public key (on unix-based installation in the directory `.ssh`). You should copy your `id_rsa.pub` key to your github account. Keep the keys in a safe place.

On Windows, you can follow instructions on how to generate your keys as explained at <http://guides.beanstalkapp.com/version-control/git-on-windows.html#installing-ssh-keys>.

### Adding the key to your ssh

In linux, execute in your shell:

```
[ ssh-add ~/.ssh/id_rsa
```

In OSX, execute in your shell:

```
[ ssh-add -K ~/.ssh/id_rsa
```

Both for OSX and linux you can add such lines in your `.bash_profile` (or the one corresponding to your shell installation such as `.zshrc`) so they are automatically executed on each new shell session.

### Tell Pharo to use your keys

You need to go to settings browser, search for "Use custom SSH keys" and complete your data there as shown in Figure 1-1).

Alternatively, you can execute in your image playground or add to your Pharo system preference file the following expressions:



### 1.3 Create a new project on github

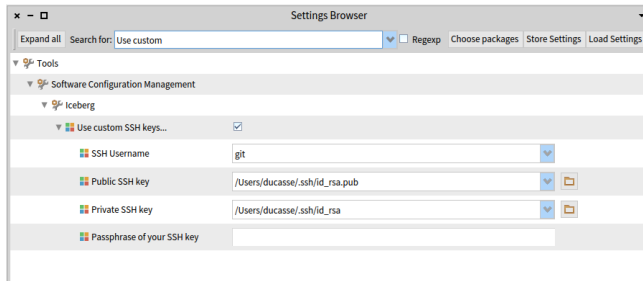


Figure 1-1 Use Custom SSH keys settings.

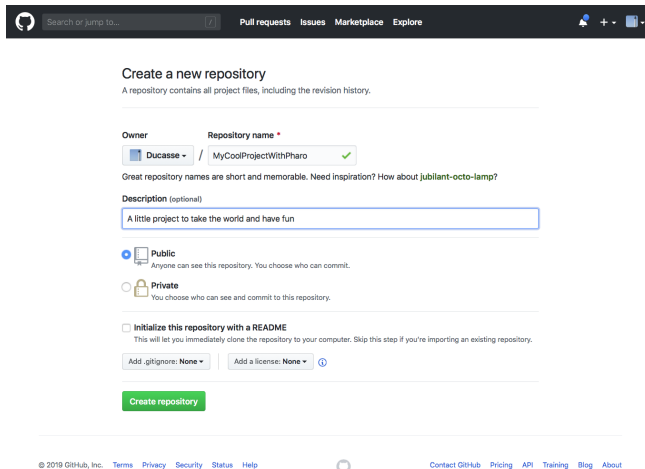


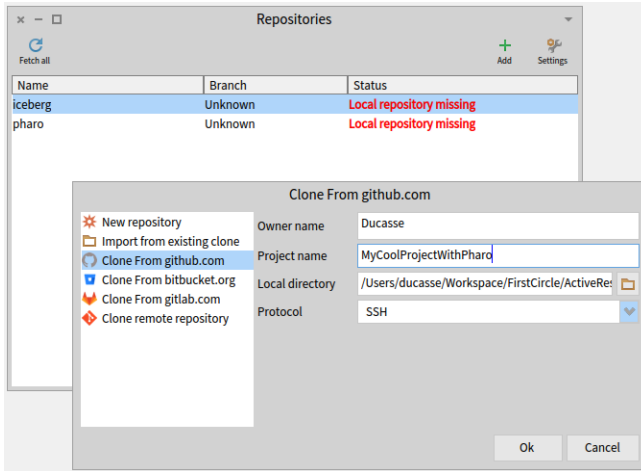
Figure 1-2 Create a new project.

```
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.

### 1.3 Create a new project on github

Figure 1-2 shows the creation of a project on Github. You should be able to do it without more information.



**Figure 1-3** Cloning new project.

## 1.4 Add a new project to Iceberg

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

- Press the '+' button available on the right of the Iceberg main window.
- Then select from which source you want to get your the project. Since you did not clone your project yet, choose the github choice.

Figure 1-3 shows that we are cloning the repository we just created. We specify the owner, the project, and the physical location where the local clone and git working copy will be on your disc.

Once done, Iceberg has added your project to its list of managed projects and cloned an empty repository to your disc as shown by Figure 1-4 shows the status of the project

We explain it now:

- MyCoolProjectWithPharo has a star and is green because the system thinks that some changes were not committed. It may happens because of packages in your image in strange state. You should not really bother 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.

### 1.5 [Optional but strognly suggested] Adding a src directory

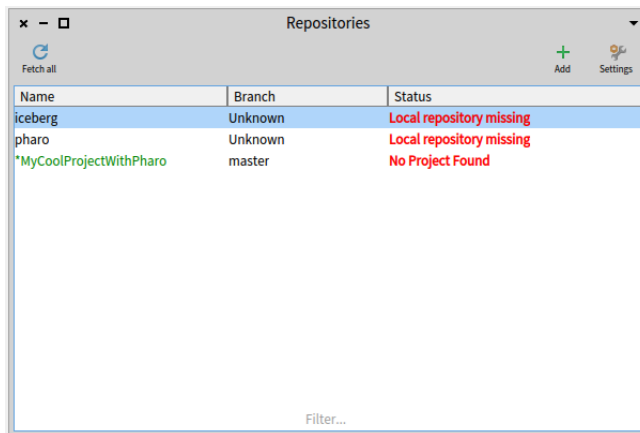


Figure 1-4 Just after cloning an empty project.

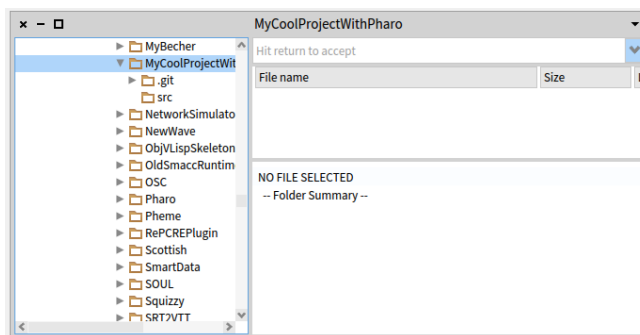


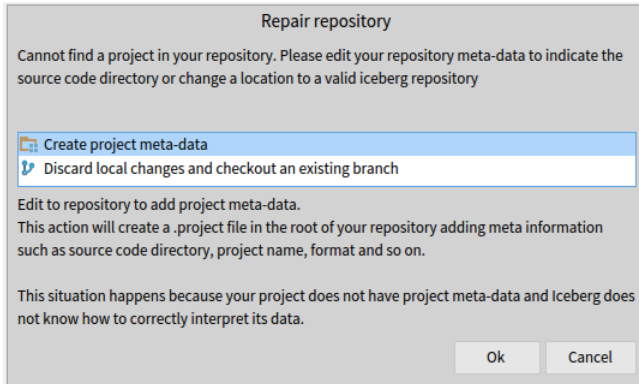
Figure 1-5 Adding a src folder to .

### 1.5 [Optional but strognly suggested] Adding a src directory

Some developers like to group all their code in a directory `src` and this is a nice practice. We follow strongly suggest to follow it. You can go to your filesystem and create a repository. And you can also use the Pharo FileList Browser to do it as shown in Figure 1-5.

### 1.6 Repair to the rescue

Iceberg is a smart tool that tries to help you fixing the problems you may encounter while working with Git. As a general principal each time you get a status with a red status (such as No Project Found, or Detached Working



**Figure 1-6** Create project meta-data action and explanation.

Copy), you should ask Iceberg to fix it using the **Repair** item.

Iceberg cannot solve all the situations automatically but it will propose and explain you possible repair actions. The actions are ranked from the most probable to the least one. Each action will explain you the situation and the consequence of the action. It is always clever to read them.

Figure 1-6 shows the action Create project meta-data and its explanation.

## 1.7 Create project meta-data

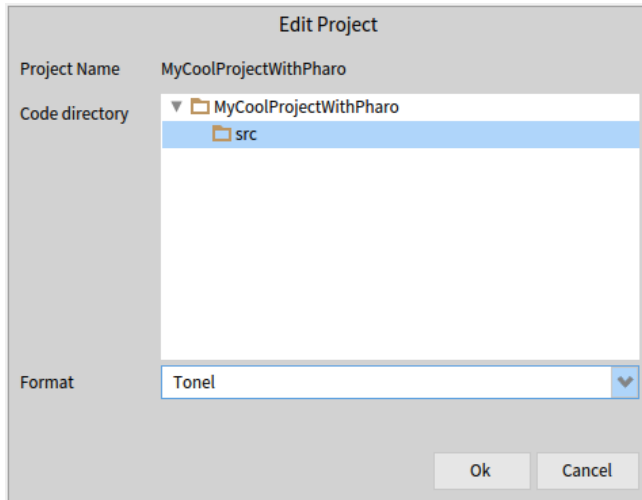
When you chose to create the project meta-data, Iceberg show you the file of your project as well as the repository encoding as shown in Figure 1-7. Tonel is the favorite encoding for Pharo project. Tonel has been designed to be Windows and file system friendly. So change it only if you know why.

When continuing, Iceberg shows you the files that you are actually commit as shown in Figure 1-8

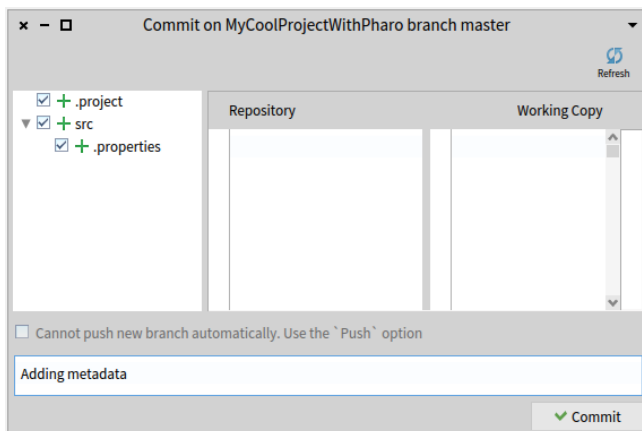
Once you have committed the meta-data, Iceberg shows you that your project is repaired but not loaded as shown in Figure 1-9. This is normal since up until now we did not add package to our project. But your local repository is ready.

## 1.8 Adding and committing your package

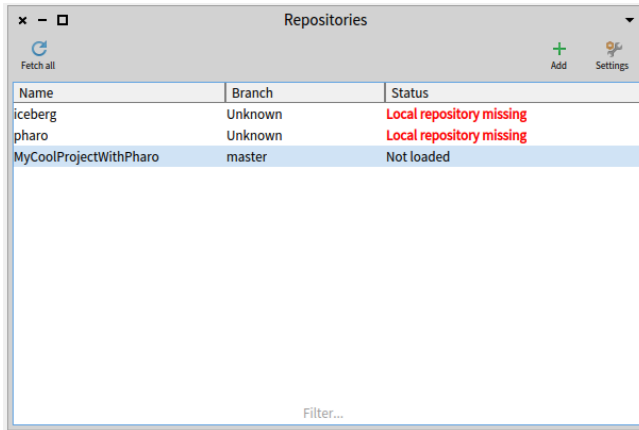
Once you project contains Iceberg meta-data, Iceberg will be able to manage it easily. Double click on your project and add your package by pressing the + (Add Package) button as shown by Figure 1-10.



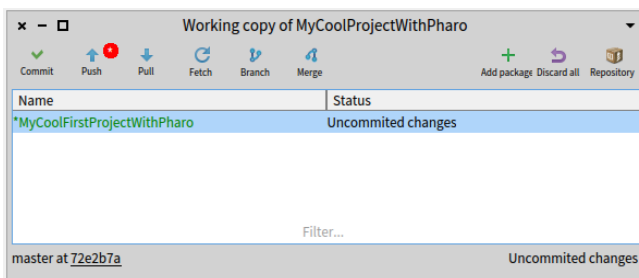
**Figure 1-7** Showing where the metadata will be saved and the format encodings.



**Figure 1-8** Details of meta data commit.



**Figure 1-9** The package is clean, meta-data are saved, but it is not loaded.



**Figure 1-10** Adding a package to your project.

Again Iceberg shows that your package contains changes that are not committed using the green color and the \* in front of the package name.

Now you are left with two actions:

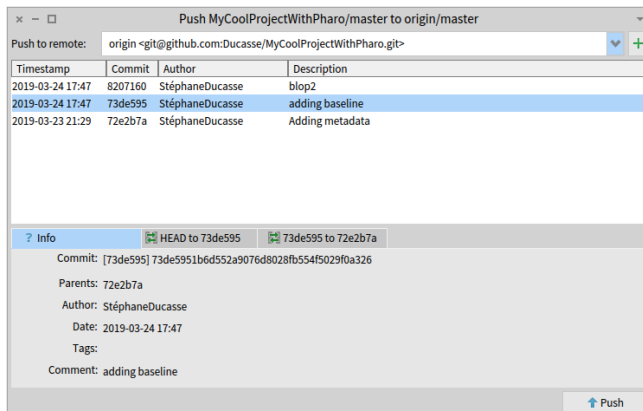
- Commit the changes to your local repository using the Commit button. Iceberg will reflect this change by removing the \* and the changing the color.

You can commit several times.

- Publishing your changes from your local directory to your remote repository using the Push button.

When you push your changes, Iceberg will show you all the commits that you already did and it will push all the changes that are present to your remote repository as shown in Figure 1-11. The figure shows the commits we did to add a baseline and this is what we will do now. Indeed what is left to do

## 1.9 Defining a BaselineOf



**Figure 1-11** Publishing your committed changes.

is to define a baseline to make sure that you can easily load your project in another image.

## 1.9 Defining a BaselineOf

A BaselineOf is a description of the project architecture. You will express the dependencies between packages and other projects so that all the dependent projects are loaded without the user to have to understand and bother about them. A baseline is expressed as a subclass of BaselineOf and packaged in a package named 'BaselineOfXXX'.

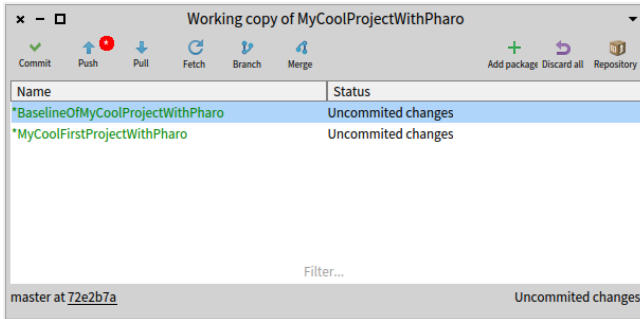
So if you have no dependencies, you can have something like this.

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

BaselineOfMyCoolProjectWithPharo >> baseline: spec
<baseline>
spec
  for: #common
  do: [ spec package: 'BMyCoolProjectWithPharo'.
      spec group: 'default' with: #('FileDialog') ]
```

Once you have defined the baseline, you should add it to your project as shown in Figure 1-12. And you should commit it and push your changes to your remote repository.

The online documentation available at: <https://github.com/pharo-open-documentation/pharo-wiki/blob/master/General/Baselines.md>.



**Figure 1-12** With a Baseline.

## 1.10 Loading from an existing repository

If you already have repository, and you just want to load it into Pharo, you can do it in two ways. The first one is as we did above. You can select a package and manually load it.

The second one makes use of Metacello. This will however only work if you have already created a BaselineOf. With baseline, you can just do

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

Do not forget the /src at the end if you have created a subdirectory for the code!

## 1.11 [Optional] Add a nice .gitignore file

Iceberg manages automatically such files.

```
# For Pharo 70 and up
# http://www.pharo.org
# Since Pharo 70 all the community is moving to git.

# image, changes and sources
*.changes
*.sources
*.image

# Pharo Debug log file and launcher metadata
PharoDebug.log
pharo.version
meta-inf.ston
```



### 1.11 [Optional] Add a nice .gitignore file

```
:
# Since Pharo 70, all local cache files for Monticello package
#   cache, playground, epicea... are under the pharo-local
#   /pharo-local

# Metacello-github cache
/github-cache
github-*.zip
```



## Empowering your projects

Now that you can save like a breeze your code on github you can take advantage of services to automate actions using Travis for example.

### 2.1 Adding Travis integration

By adding two simple files, you can get the tests of your project being run after each commit with travis. You need to enable the travis of your github repository. Check on your travis account.

And you should add the two following files: `.travis.yml` and `.smalltalk.ston` on the top level of your repository.

`.travis.yml`

```
[ language: smalltalk
  sudo: false
  os:
    - linux
  smalltalk:
    - Pharo-7.0
```

`.smalltalk.ston`

```
[ SmalltalkCISpec {
  #loading : [
    SCIMetacelloLoadSpec {
      #baseline : 'MyCoolProjectWithPharo',
      #directory : 'src',
      #platforms : [ #pharo ]
    }
  ]
}
```

```
}
[ }
```

If you've done everything right Travis will pick up the changes and will start testing and building it... and you're done, congratulations!

## 2.2 On windows

If you want to make sure that your code runs on windows you should use the Appveyor service and add the appveyor.yml file.

```
environment:
  CYG_ROOT: C:\cygwin
  CYG_BASH: C:\cygwin\bin\bash
  CYG_CACHE: C:\cygwin\var\cache\setup
  CYG_EXE: C:\cygwin\setup-x86.exe
  CYG_MIRROR: http://cygwin.mirror.constant.com
  SCI_RUN: /cygdrive/c/smalltalkCI-master/run.sh
matrix:
  - SMALLTALK: Pharo-6.1
  - SMALLTALK: Pharo-7.0

platform:
  - x86

install:
  - '%CYG_EXE% -dgnqNO -R "%CYG_ROOT%" -s "%CYG_MIRROR%" -l
    "%CYG_CACHE%" -P unzip'
  - ps: Start-FileDownload
    "https://github.com/hpi-swa/smalltalkCI/archive/master.zip"
    "C:\smalltalkCI.zip"
  - 7z x C:\smalltalkCI.zip -oC:\ -y > NULL

build: false

test_script:
  - '%CYG_BASH% -lc "cd $APPVEYOR_BUILD_FOLDER; exec 0</dev/null;
    $SCI_RUN"'
```

## 2.3 Adding badges

With CI happily running, you can add a badge to your readme that will show the current status of your project. Here is the one of the Containers-Stack project where we enabled the coveralls.io service to get the coverage of test computed and displayed too.

## 2.3 Adding badges

```
# Containers-Stack
A dead stupid stack implementation, but one fully working :)

[![Build
  Status](https://travis-ci.com/Ducasse/Containers-Stack.svg?branch=master)]
(https://travis-ci.com/Ducasse/Containers-Stack)
[![Coverage
  Status](https://coveralls.io/repos/github//Ducasse/Containers-Stack/badge.svg?
(https://coveralls.io/github//Ducasse/Containers-Stack?branch=master)
[![License](https://img.shields.io/badge/license-MIT-blue.svg)]()
[![Pharo
  version](https://img.shields.io/badge/Pharo-7.0-%23aac9ff.svg)]
(https://pharo.org/download)
[![Pharo
  version](https://img.shields.io/badge/Pharo-8.0-%23aac9ff.svg)]
(https://pharo.org/download)

## Installation
The following script installs Containers-Stack in Pharo.

```smalltalk
Metacello new
  baseline: 'ContainersStack';
  repository: 'github://Ducasse/Containers-Stack/src';
  load.
```
```

To obtain the necessary link, click on the badge in your Travis project overview and select one of the options. You can insert the markdown code directly to your README.md.



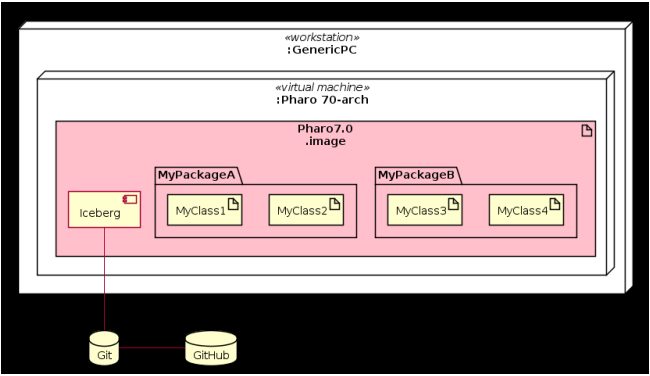
# Understanding the architecture

## 3.1 Glimpse at the architecture

As git is distributed versioning, you will need a local clone of the repository. You commit to your local clone, and from there you push to github or other places.

Figure 3-1 shows the architecture of the system.

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



**Figure 3-1** Create a new project.





# Iceberg Glossary

Git is complicated. Git with (Pharo) images is even more complicated. This page introduces the vocabulary used by Iceberg. Part of this vocabulary is Git vocabulary, part of it is Github's vocabulary, part of it is introduced by Iceberg.

## 4.1 Git

### Disk Working Copy (Git)

It is important not to confuse the code in your disk with the one of the repository itself. The repository (a kind of database) has a lot more information, such as known branches, history of commits, remote repositories, the git index and much more. Normally this information is kept in a directory named `.git`. The files that you see on your disk and that you edit are just a working copy of the contents in the repository.

### The git index (Git)

The index is an intermediate structure which is used to select the contents that are going to be committed.

So, to commit changes to your local repository, two actions are needed:

1. `git add someFileOrDirectory` will add `someFileOrDirectory` to the index.
2. `git commit` will create a new commit out of the contents of the index, which will be added to your local repository and to the current branch.

When using iceberg, you normally do not need to think about the index, Iceberg will handle it for you. Still, you might need to be aware that the index is part of the git repository, so if you have other tools working with the same repository there might be conflicts between them.

## Local and remote repositories (Git)

To work with Git you always need a local repository (which is different from the code you see on your disk, that is not the repository, that is just your working copy). Remember that the local repository is a kind of database (for code).

Most frequently your local repository will be related with one remote repository which is called origin and will be the default target for pull and push.

## Upstream (Git)

The upstream of a branch is a remote branch which is the default source when you pull and the default target when you push. Most probably is a branch with the same name in your origin remote repository.

## Commit-ish (Git)

A commit-ish is a reference that specifies a commit. Git command line tools usually accept several ways of specifying a commit, such as a branch or tag name, a SHA1 commit id, and several fatality-like combinations of symbols such as HEAD^, @{u} or master~2.

The following table contains examples for each commit-ish expression. A complete description of the ways to specify a commit (and other git objects) can be found at [https://mirrors.edge.kernel.org/pub/software/scm/git/docs/gitrevisions.html#\\_specifying\\_revisions](https://mirrors.edge.kernel.org/pub/software/scm/git/docs/gitrevisions.html#_specifying_revisions).

| Format                  | Examples                                 |
|-------------------------|--|
| 1. <sha1>               | dae86e1950b1277e545cee180551750029cfe735 |
| 2. <describeOutput>     | v1.7.4.2-679-g3bee7fb                    |
| 3. <refname>            | master, heads/master, refs/heads/master  |
| 4. <refname>@{<date>}   | master@{yesterday}, HEAD@{5 minutes ago} |
| 5. <refname>@{<n>}      | master@{1}                               |
| 6. @{<n>}               | @{1}                                     |
| 7. @{-<n>}              | @{-1}                                    |
| 8. <refname>@{upstream} | master@{upstream}, @{u}                  |
| 9. <rev>^               | HEAD^, v1.5.1^0                          |
| 10. <rev>~<n>           | master~3                                 |
| 11. <rev>^<type>        | v0.99.8^{commit}                         |

|                      |                        |
|----------------------|------------------------|
| 12. <rev>^{}}        | v0.99.8^{}}            |
| 13. <rev>^{}/<text>} | HEAD^{}/fix nasty bug} |
| 14. :/<text>         | :/fix nasty bug        |

---

## 4.2 Iceberg

### Iceberg Working Copy (Iceberg)

Iceberg also includes an object called the working copy that is not quite the same as Git's working copy. Iceberg's working copy represents the code loaded in the Pharo image, with the loaded commit and the packages.

### Local Repository Missing (Iceberg)

The Local Repository Missing status is shown by iceberg when a project in the image does not found its repository on disk. This happens most probably because you've downloaded an image that somebody else created, or you deleted/moved a git repository in your disk. Most of the times this status is not shown because iceberg automatically manages disk repositories.

To recover from this status, you need to update your repository by cloning a new git repository or by configuring an existing repository on disk.

### Fetch required. Unknown ... (Iceberg)

The Fetch required status is shown by Iceberg when a project in the image was loaded from a commit that cannot be found in its local repository. This happens most probably because you've downloaded an image that somebody else created, and/or your repository on disk is not up to date.

To recover from this status, you need to fetch from remotes to try to find the missing commit. It may happen that the missing commit is not in one of your configured remotes (even that nobody ever pushed it). In that case, the easiest solution is to discard your image changes and checkout an existing branch/commit.

### Detached Working Copy (Iceberg)

The Detached working Copy status is shown by Iceberg when a project in the image was loaded from a commit does not correspond with the current commit on disk. This happens most probably because you've modified your repository from the command line.

To recover from this status, you need to align your repository with your working copy. Either you can

1. discard your image changes and load the repository commit,

2. checkout a new branch pointing to your working copy commit or
3. merge what is in the image into the current branch.

**Detached HEAD (Git)** The Detached HEAD status means that the current repository on disk is not working on a branch but on a commit. From a git stand-point you can commit and continue working but your changes may get lost as the commit is not pointed by any branch. From an Iceberg stand-point, we forbid commit in this state to avoid strange situations. To recover from this status, you need to checkout a (new or existing) branch.

# Bibliography

