Weld

https://code.google.com/p/weld

A tool to manage git vendor branches

Background

- I'm Tibs / Tony Ibbs
- I used to work for Kynesim
- I now work for Velocix
- weld was originally written by Richard Watts, and it was the last project I worked on at Kynesim.

The problem

- Verson control of projects with a moderate to large number of packages.
- For instance: the sources needed to build a Linux system.

Two traditional ways to organise

1. One package per repository
2. One repository for the world

One package per repository

- easy to relate to upstream
- easy to track licensing
- harder to decide which packages to use
- impossible to track a change across multiple packages
- cumbersome to "name" a version of the project
- cloning many small packages can be slow

One repository for the world

- one repository per project
- easy to make a change across multiple packages
- "name" a version by the SHA1 id for the commit
- easy to branch the entire project
- harder to reason about individual packages
- difficult to relate to individual upstreams

- hard to share packages between projects

Or there's weld

- Attempts to make it reasonably simple to have something of both worlds.
- We only support git
- Meta-information in a `.weld` directory, next to the `.git` directory
- The normal user just sees a single repository
- Package managers set the weld up, using the `weld` command line tool

A little terminology: weld

- A **weld** is a git repository containing the source code for a project.
- `weld` is also the command line tool that is used to maintain welds.

A little terminology: seam

- A **seam** is a mapping from a directory in an external git repository to the corresponding directory in the weld.
- Colloquially it is also the directory in the weld that is so described.

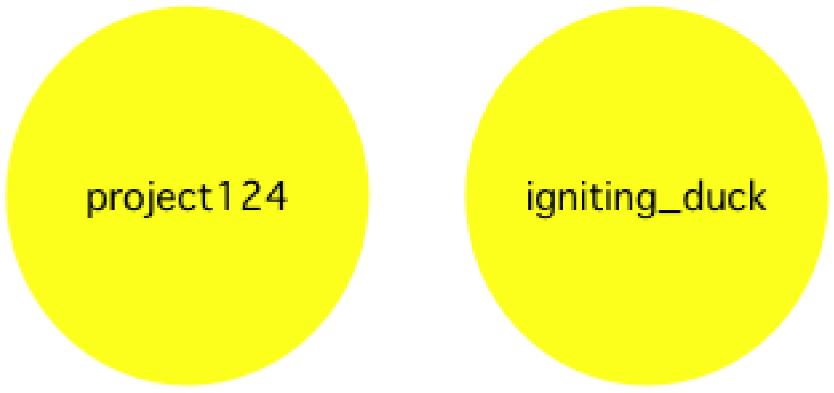
A little terminology: base

- A **base** is an external git repository (and implicitly its branch or other specifiers) from which seams are pulled (and to which they are pushed).
- The term may also be used to refer to the clone of that external directory in the `.weld/bases` directory.

Creating a weld

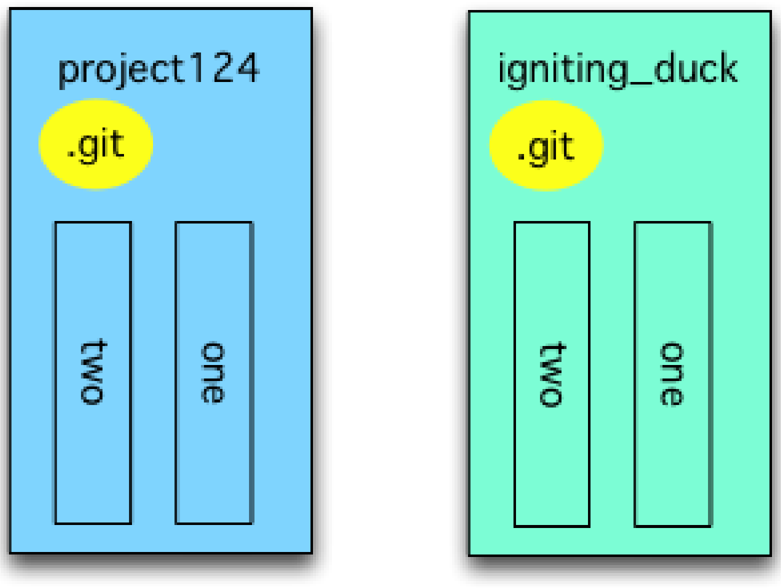
(Done by the weld manager)

We start with two remote repositories



We can clone them in the normal manner

```
$ git clone file://<repo_base>/project124
$ git clone file://<repo_base>/igniting_duck
```



...or we can use weld

OK, some XML

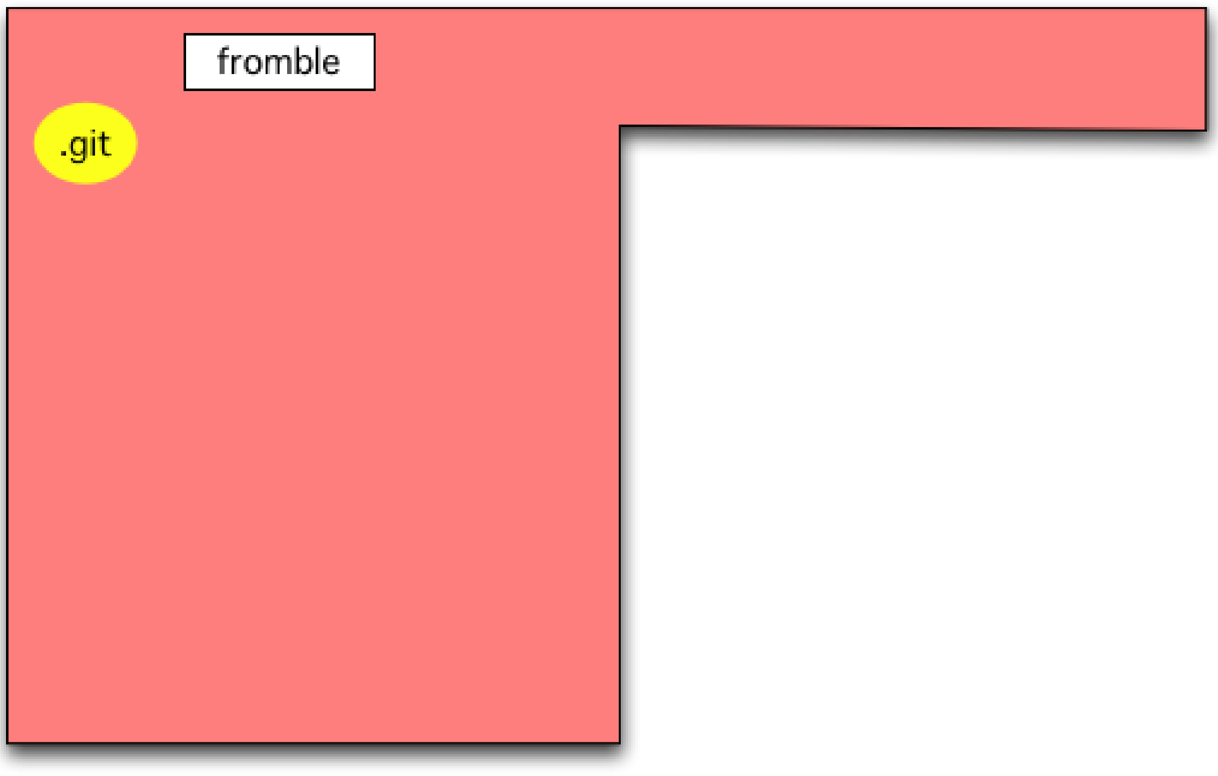
- A weld is described by an XML file:

```
<?xml version="1.0" ?>
<weld name="frank">
  <origin uri="file://<repo_base>/fromble" />
  <base name="project124" uri="file://<repo_base>/project124"/>
    <seam base="project124" dest="124" />
  <base name="igniting_duck" uri="file://<repo_base>/igniting_duck" />
    <seam base="igniting_duck" source="one" dest="one_duck" />
    <seam base="igniting_duck" source="two" dest="two_duck" />
</weld>
```

Given that, we can `weld init`

```
$ mkdir fromble
$ cd fromble
$ weld init ../frank.xml
> git init
> git add fromble/.weld/welded.xml .gitignore
> git remote rm origin
> git remote add origin file://<repo_base>/fromble
> git commit --allow-empty --file /tmp/weldcommitYp7JZ2
Weld initialised OK.
```

and we get an empty weld

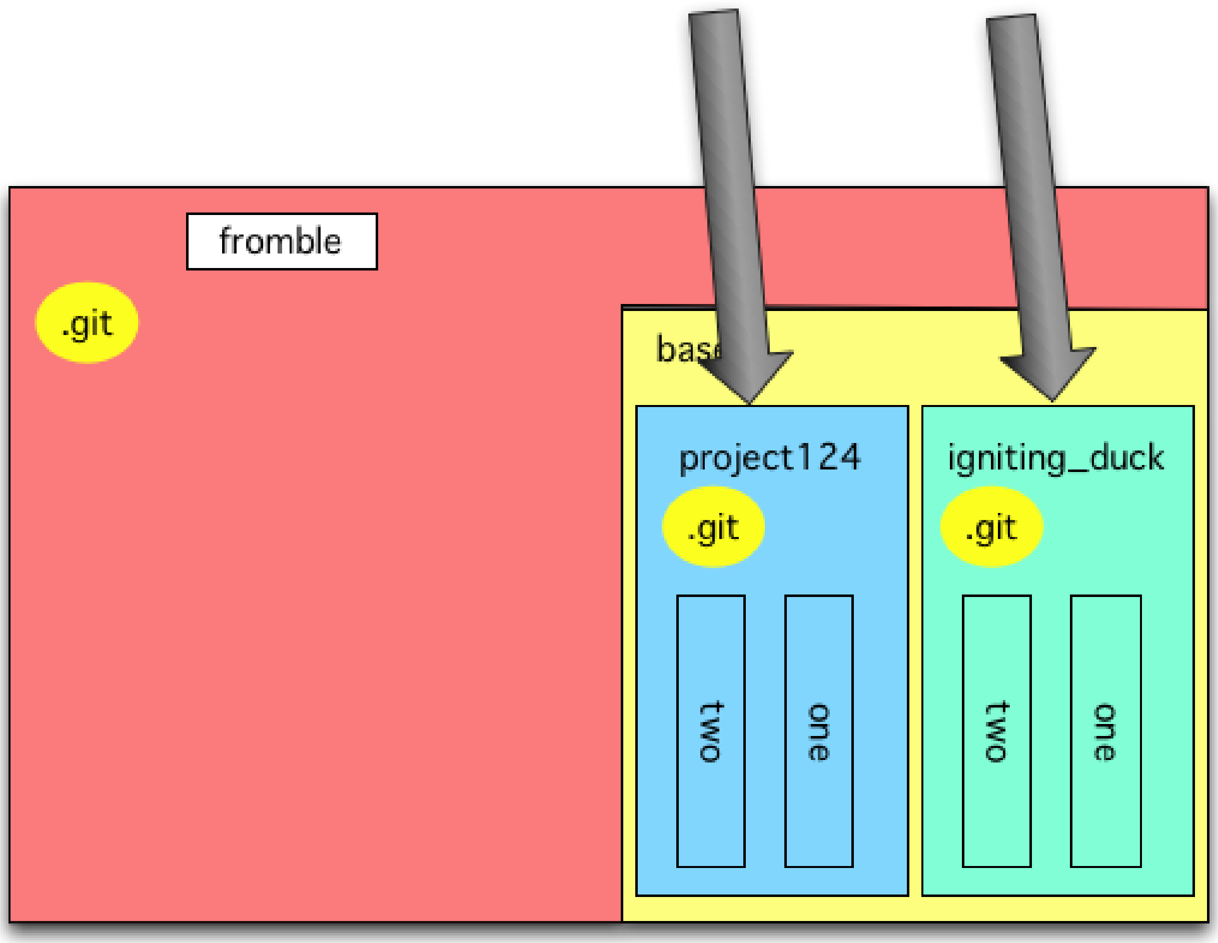


To populate our weld

```
$ weld pull _all
```

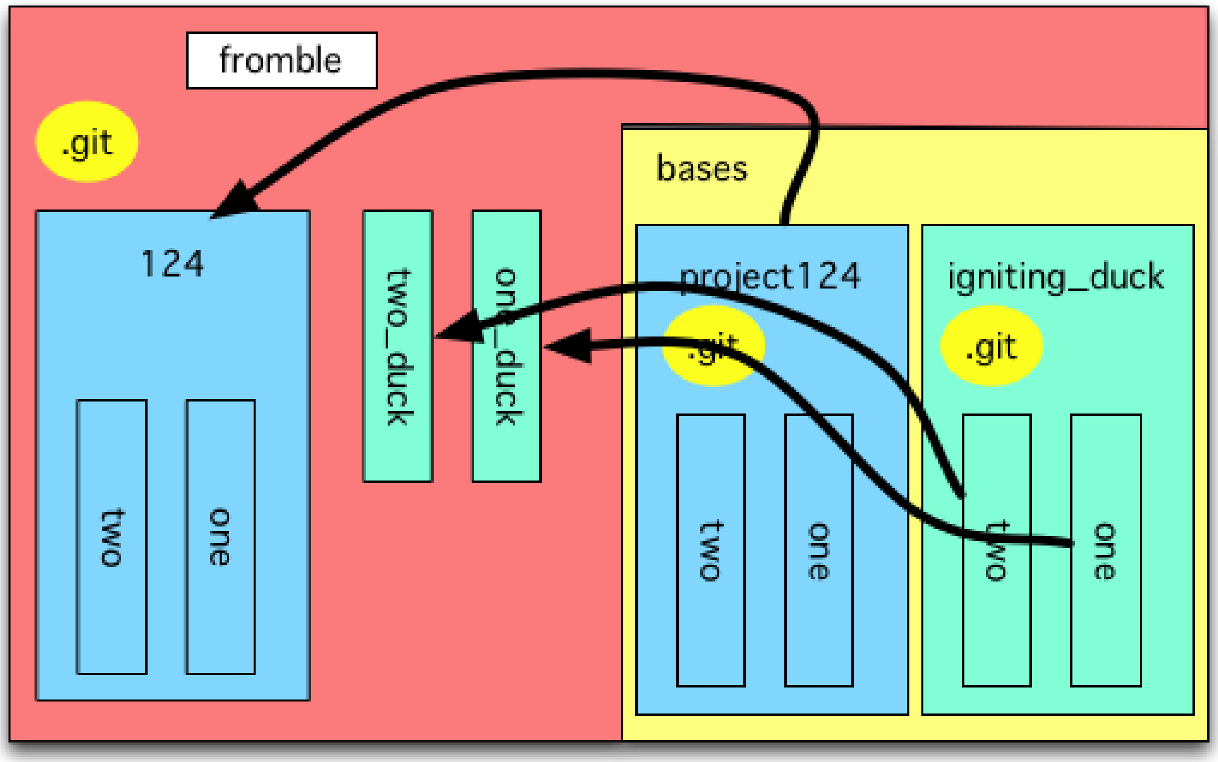
weld pull _all (1)

This clones the two remote repositories into the weld's `.weld/builds` directory

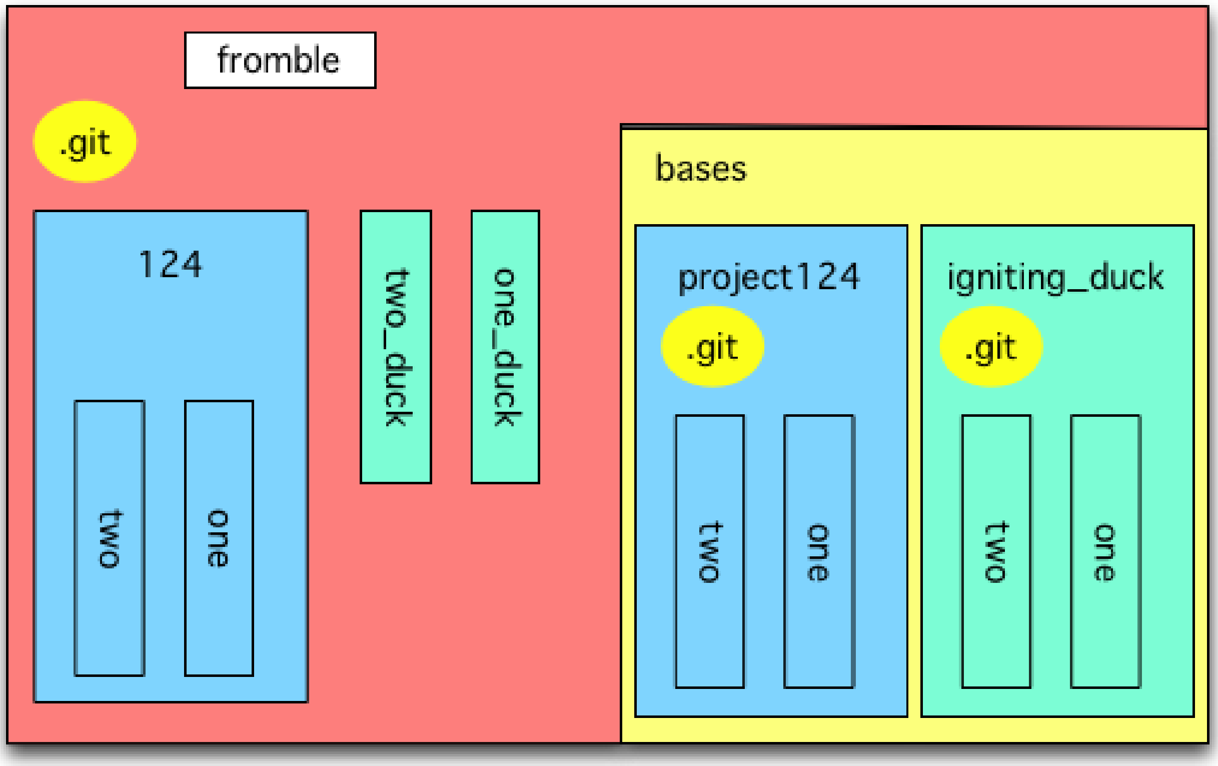


weld pull _all (1)

and then copies the content of those clones into the appropriate places in the weld, and commits the new weld contents.



And now we have a useful weld



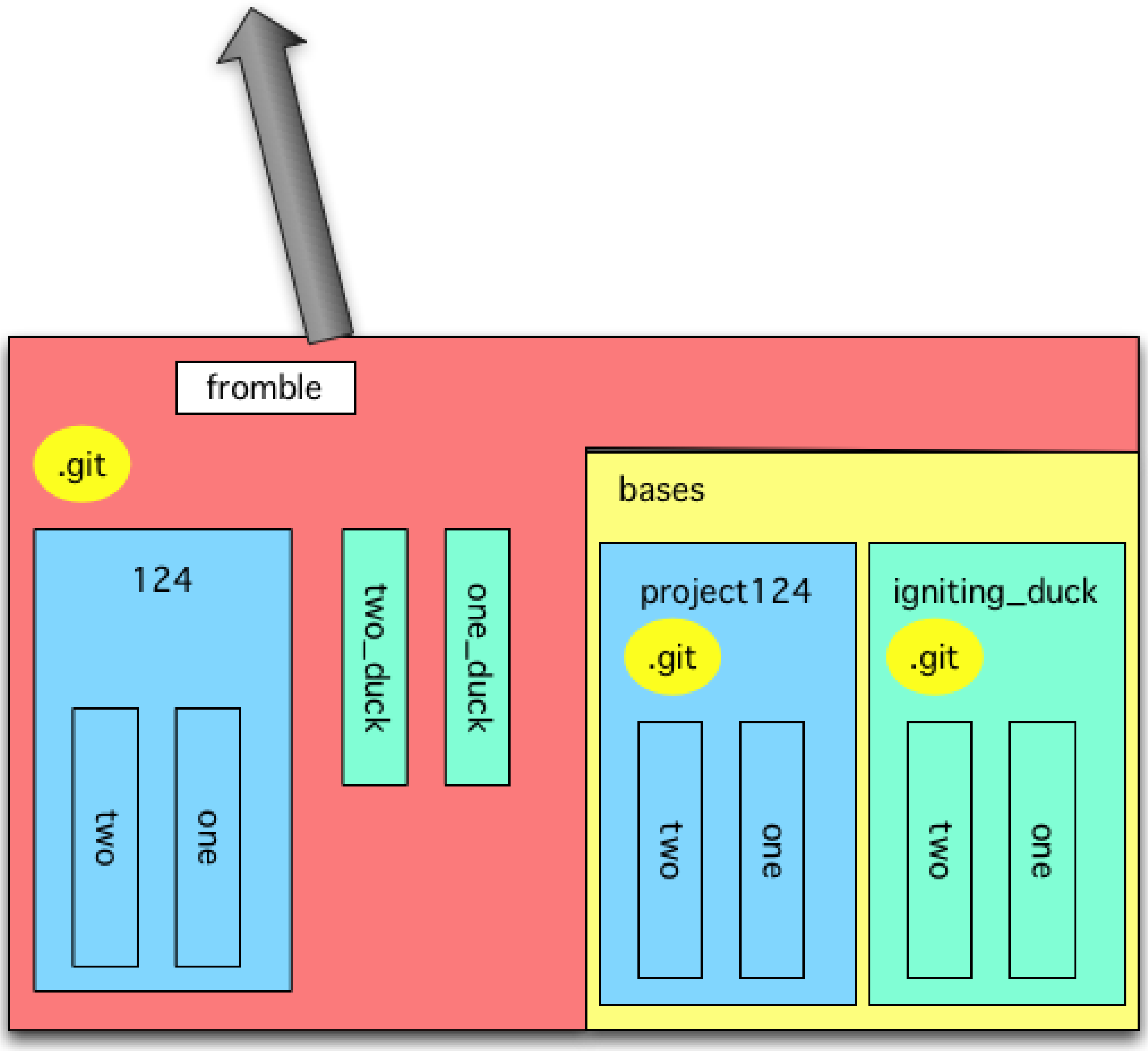
We can create a bare repository

- in the normal manner - in this case:

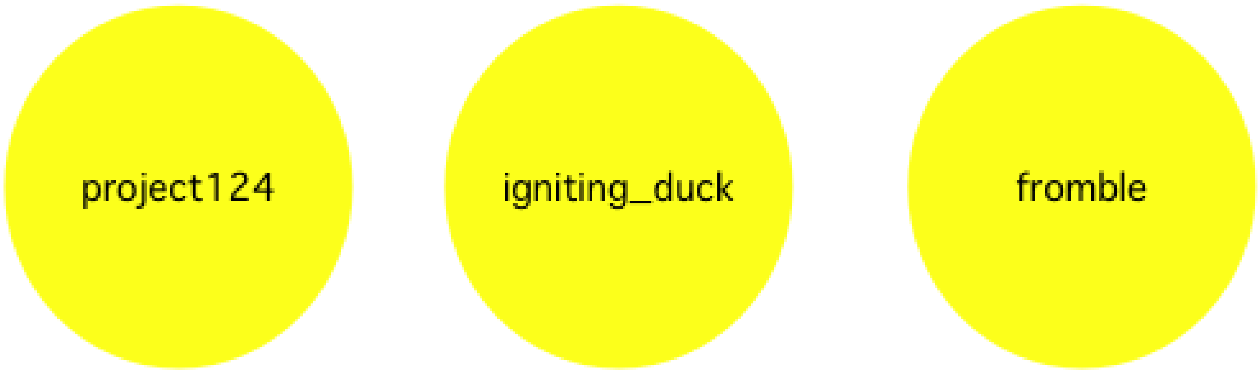
```
$ pushd <repo-base>
$ mkdir fromble
$ cd fromble
$ git init --bare
$ popd
```

- and push to it:

$ git push master origin



So we now have three remote repositories

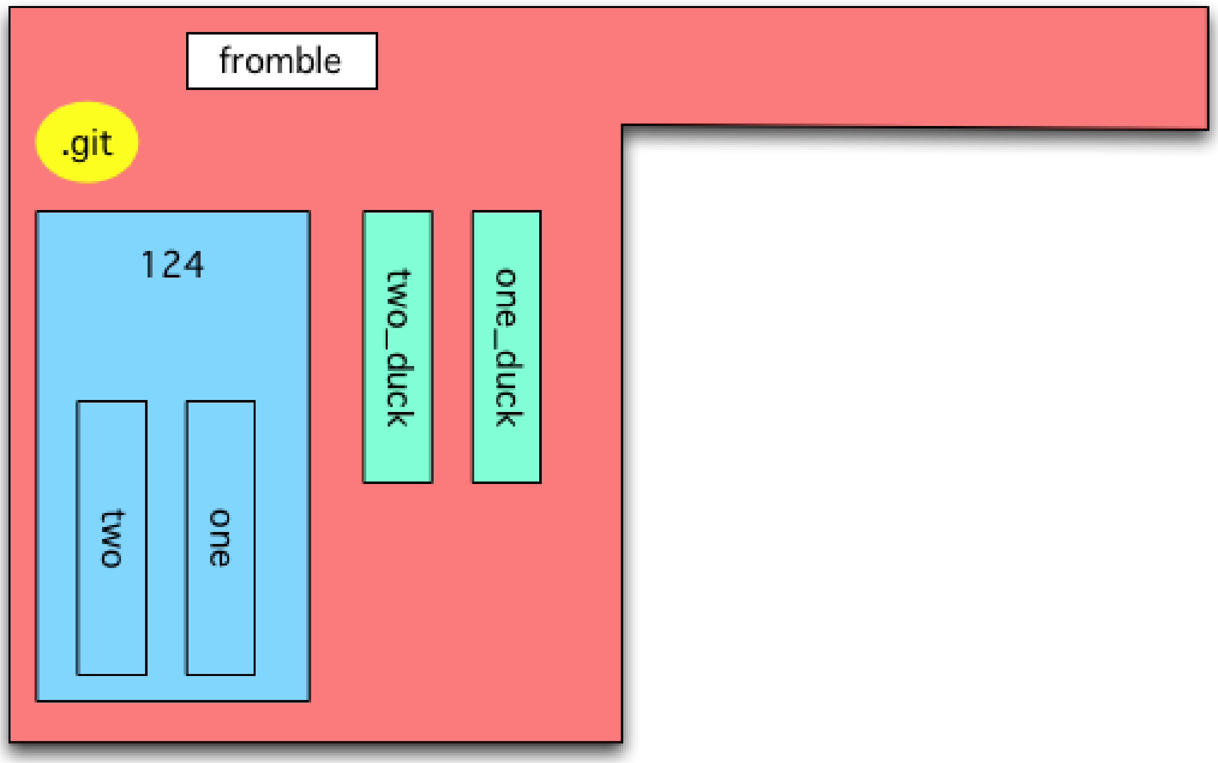


Using the weld

- Another user can now clone the weld directly:

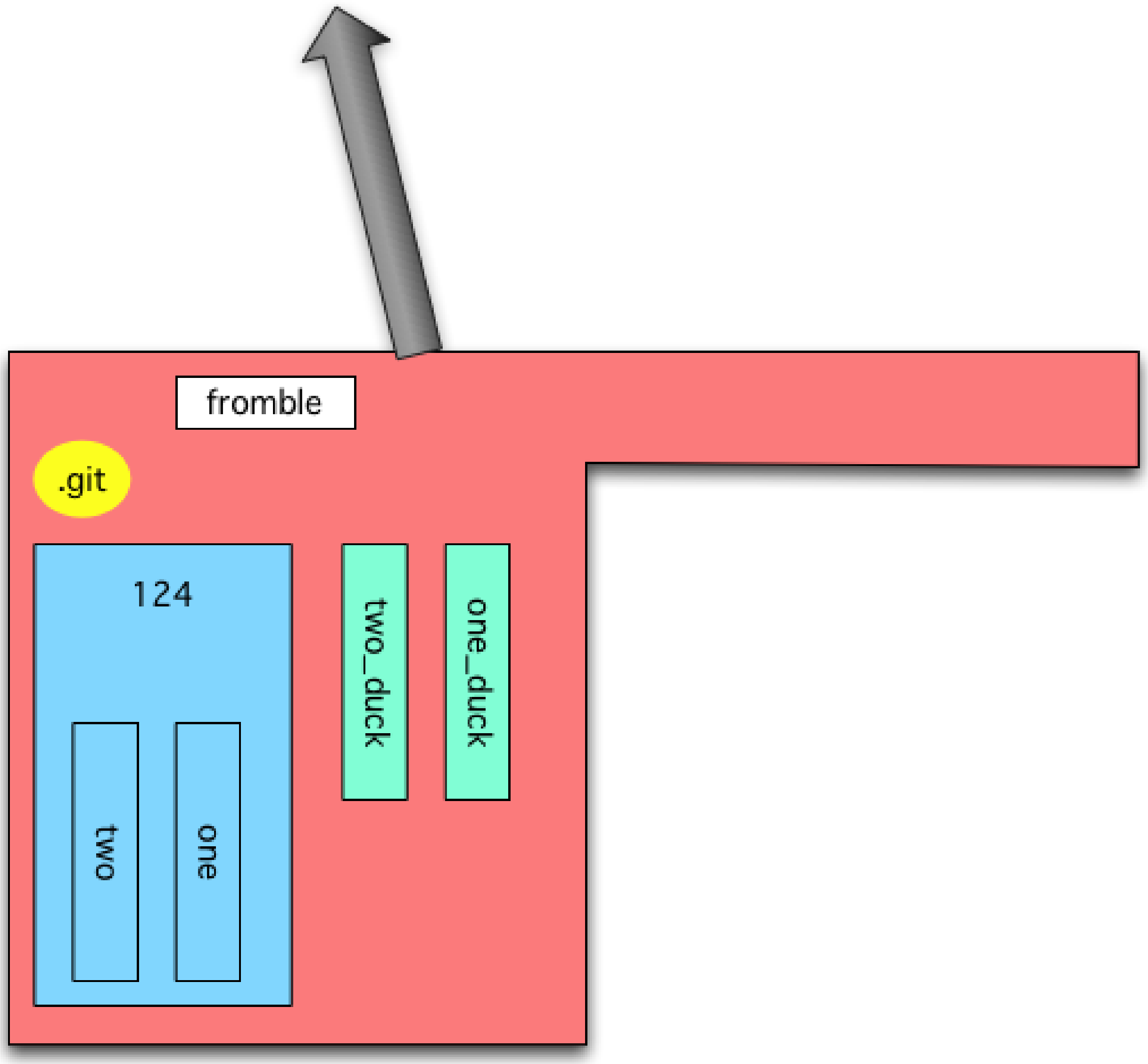
```
$ cd ~/work
$ git clone file://<repo_base>/fromble
```

which gives them the weld with its seams

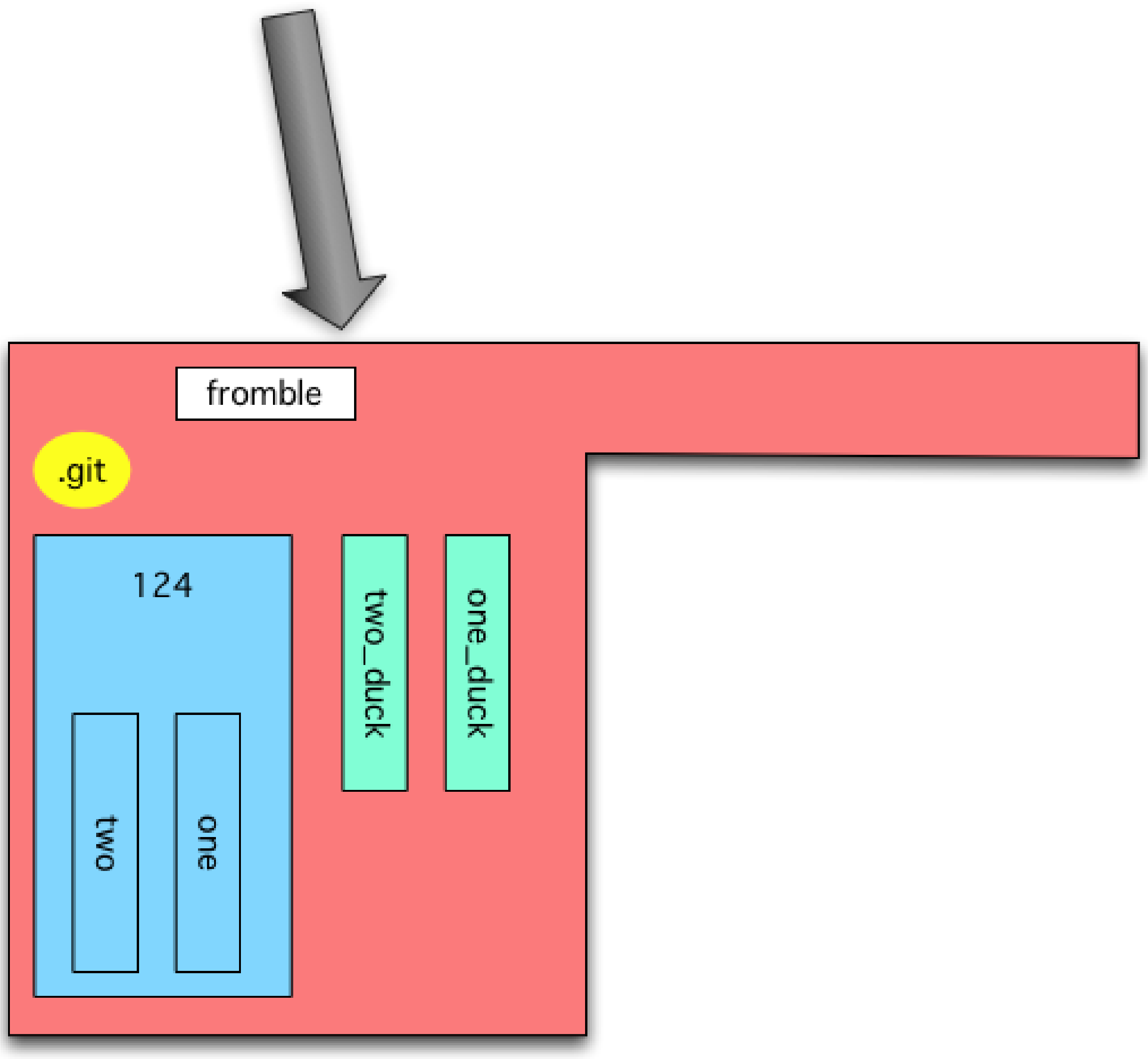


- Just work with the weld as any other git repository.

Push



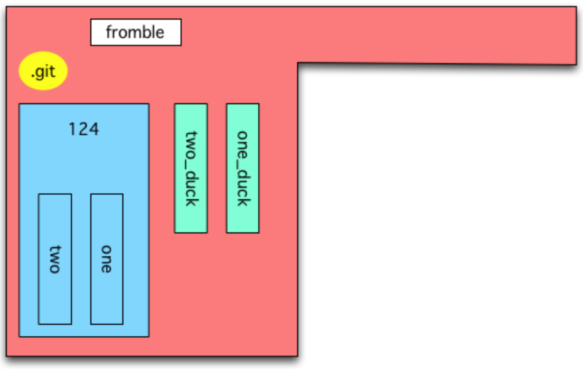
Pull



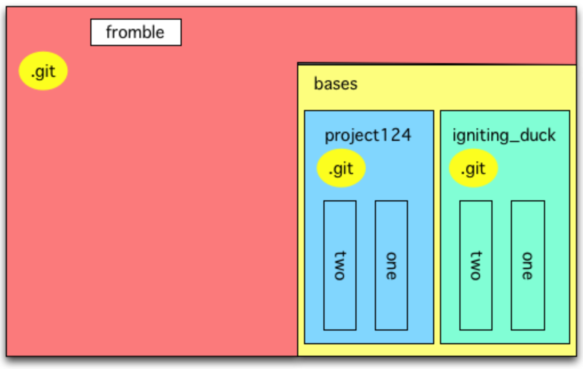
What about pushing back upstream?

- That's what `weld push` is for.
- Specifically, `weld push <basename>` pushes the appropriate commits to the named base, allowing user interaction if necessary.

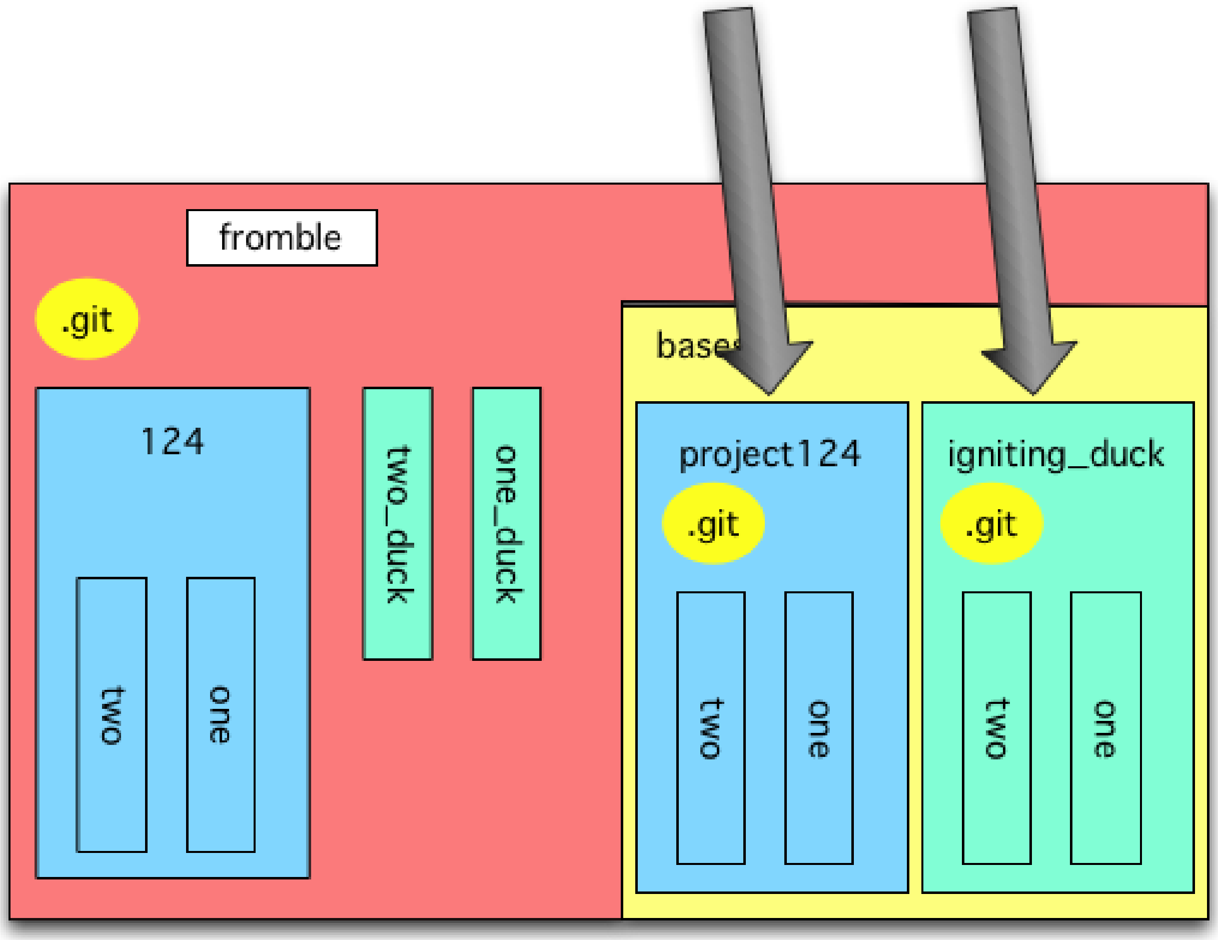
weld push (0)



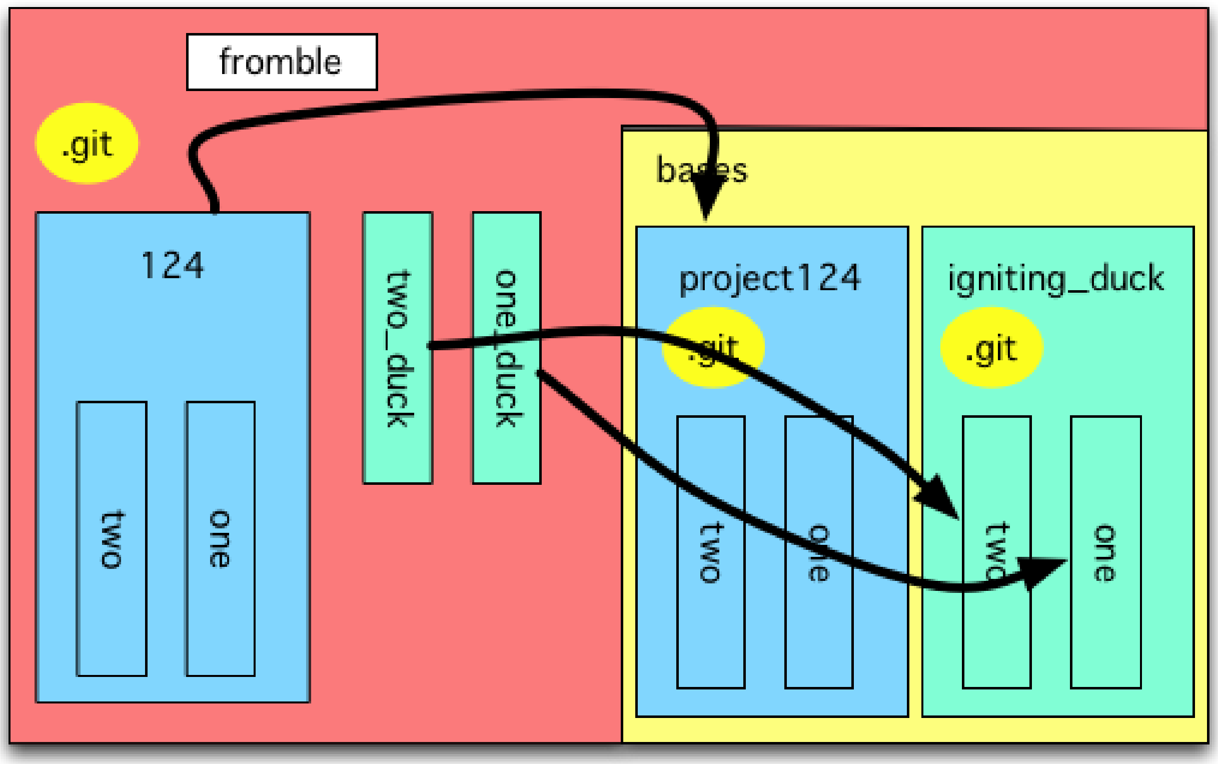
or



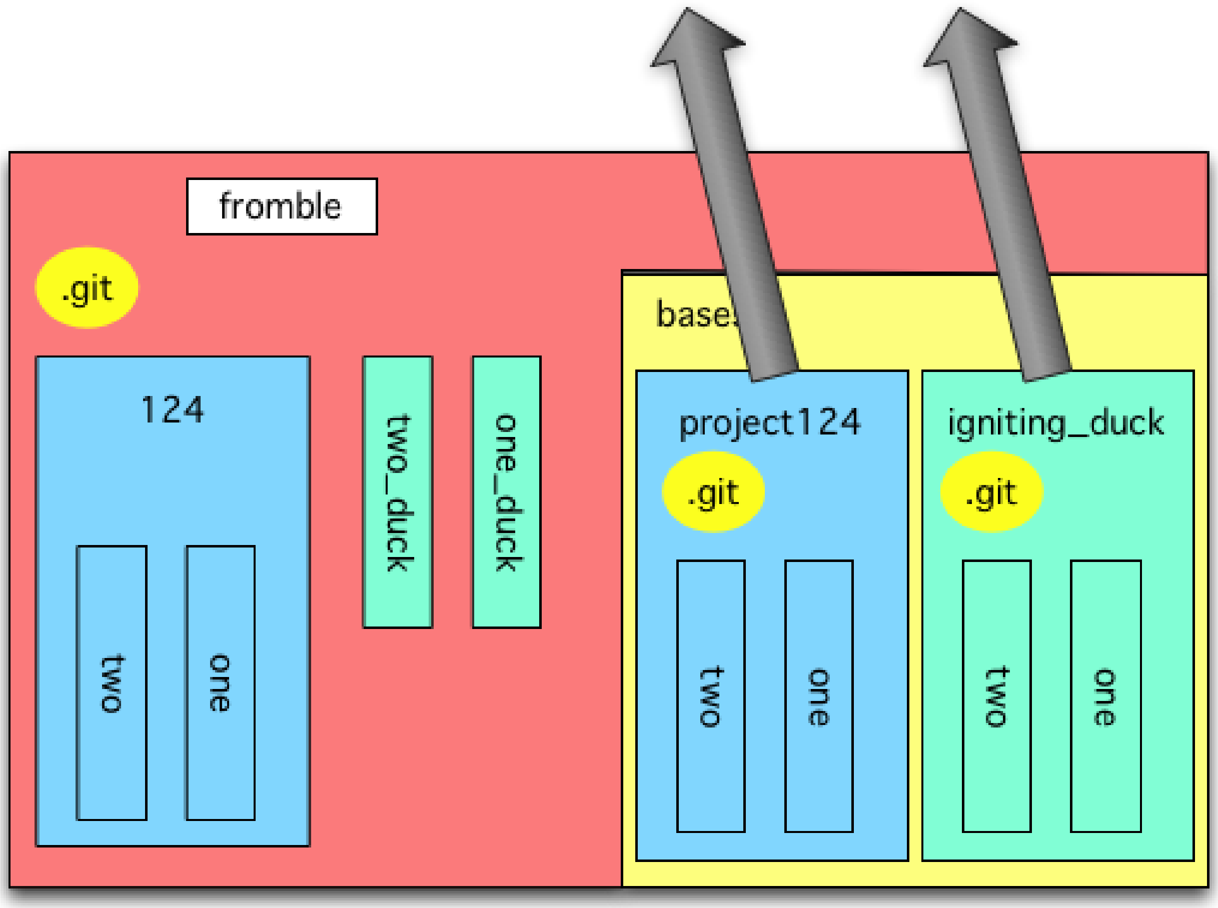
weld push (1)



weld push (2)


fromble
.git
124
two
one
two_duck
one_duck
bases
project124
.git
two
one
igniting_duck
.git
two
one

weld push (3)



Things not to do in a world of welds

- Don't use git submodules in bases
- Don't use commit messages that start "X-WeldState:"
- Don't use branches that start "weld-"
- Don't change the name of the `origin` remote of a weld (`weld` assumes that `origin` is the origin remote it should use)

Who is using it?

It is integrated with *muddle*, Kynesim's embedded system build and intergration tool, and so is used by Kynesim and their customers.

But it is entirely independent of muddle, so you can use it, too.

fin: weld

- https://code.google.com/p/weld
- Mozilla Public License 1.1
- Documentation on ReadTheDocs

(slides prepared using https://github.com/marianoguerra/rst2html5)