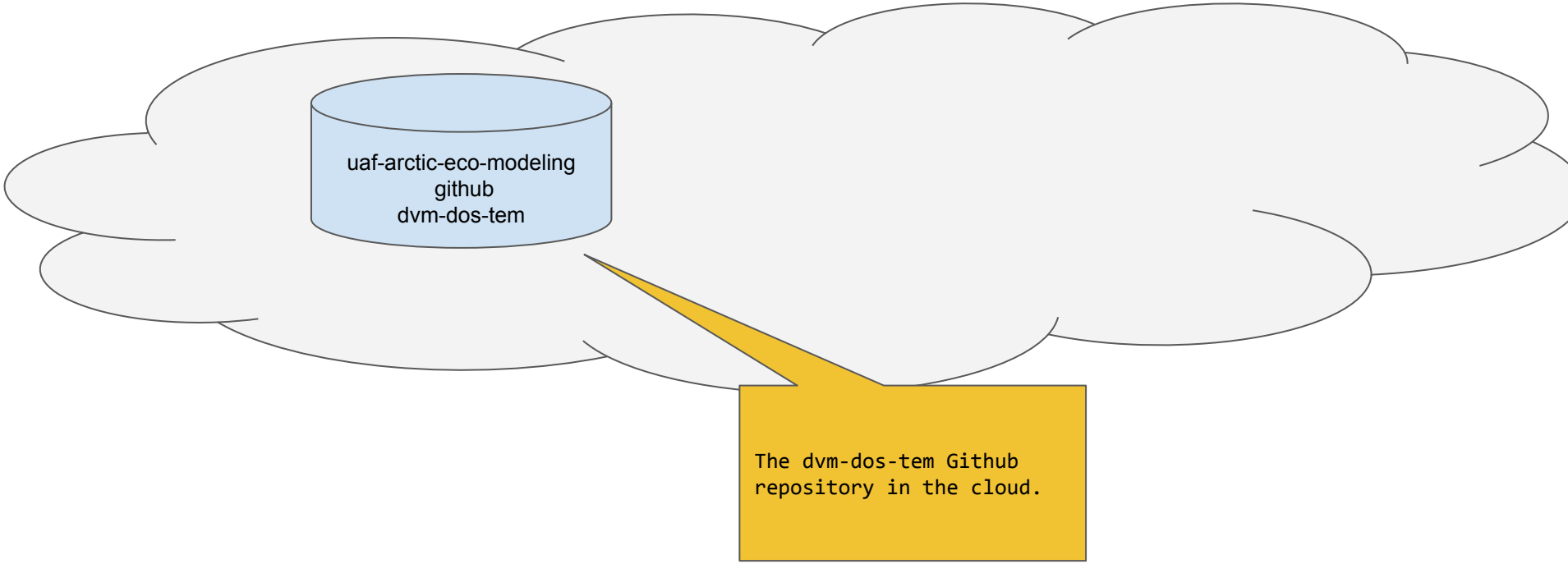
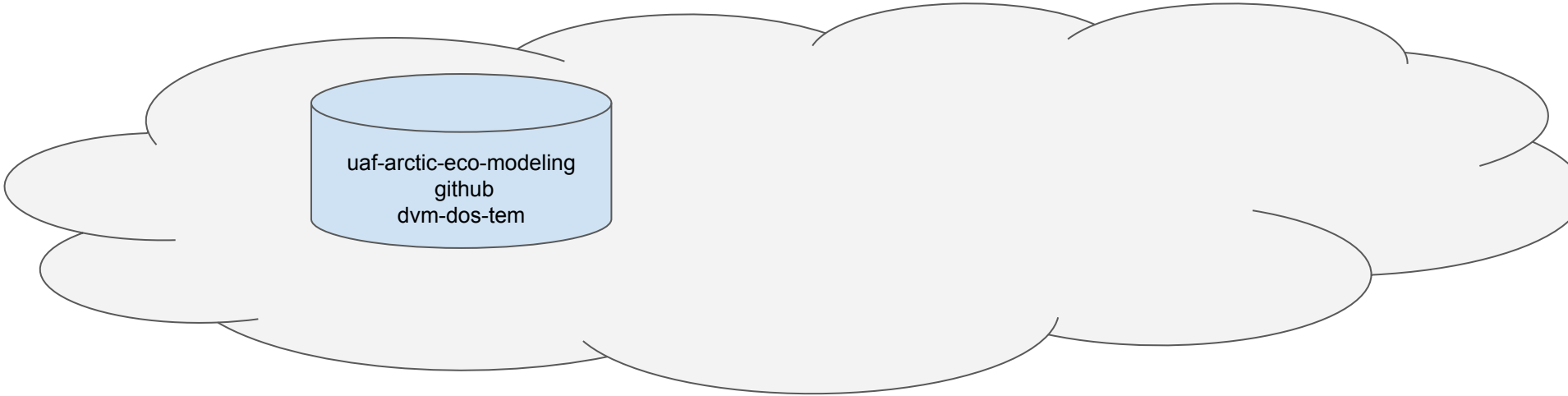
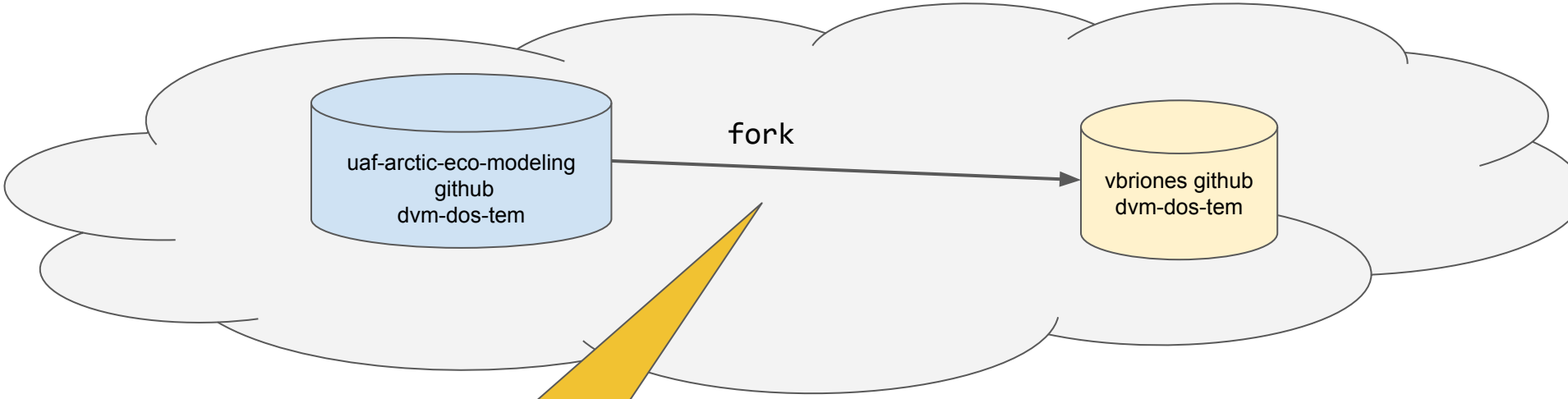

uaf-arctic-eco-modeling
github
dvm-dos-tem
The dvm-dos-tem Github repository in the cloud.

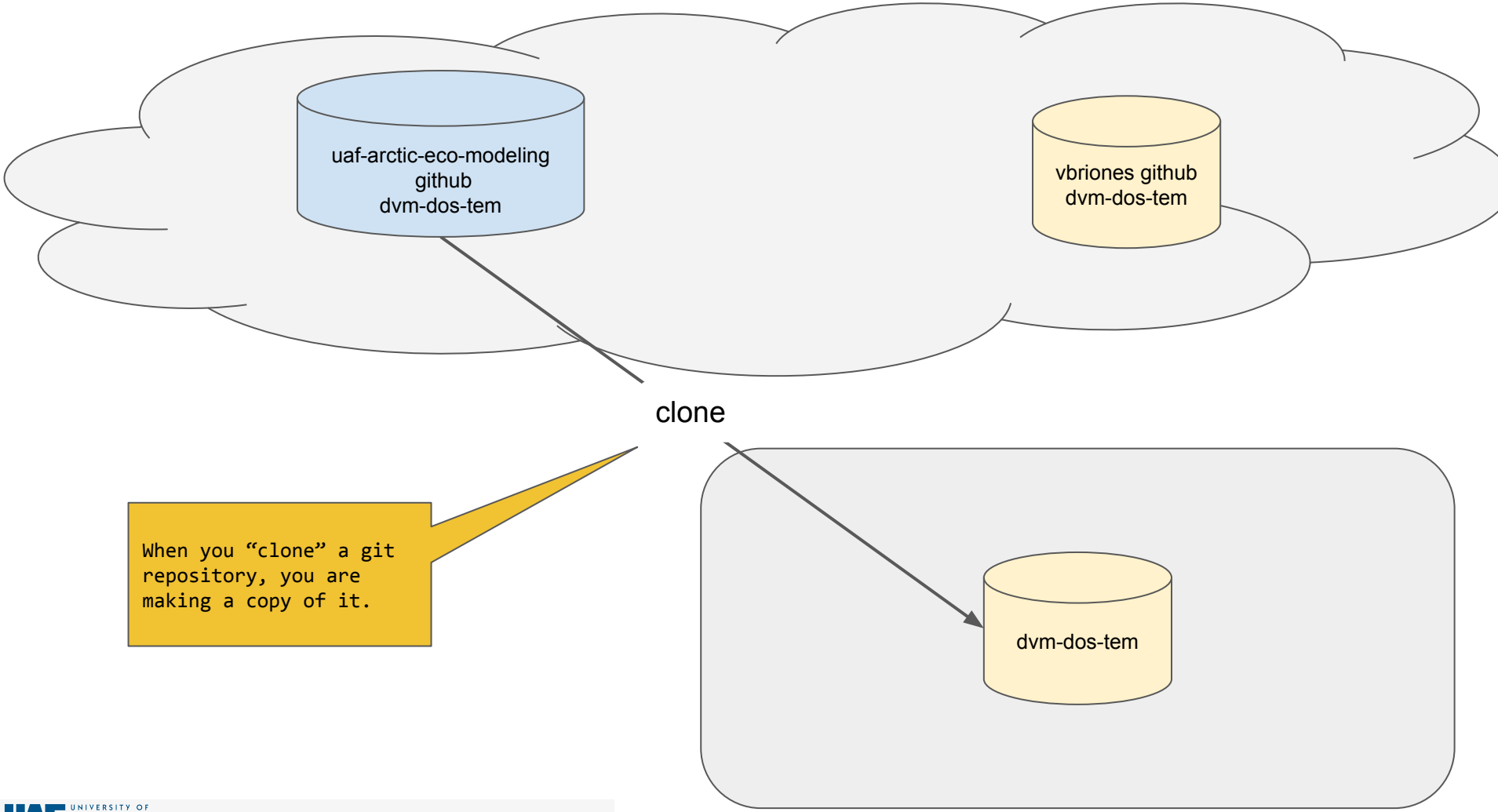


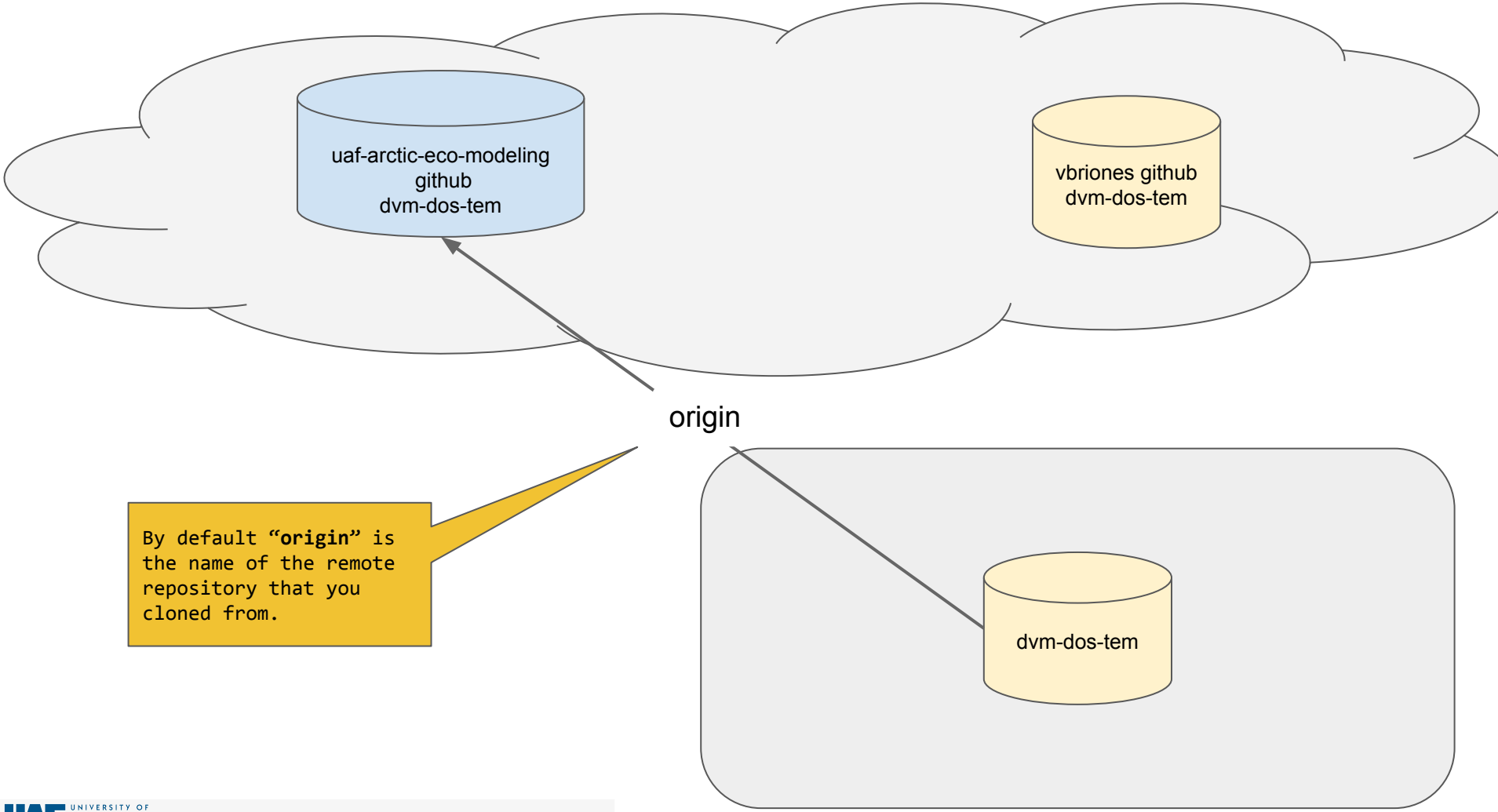
Your computer; could be:

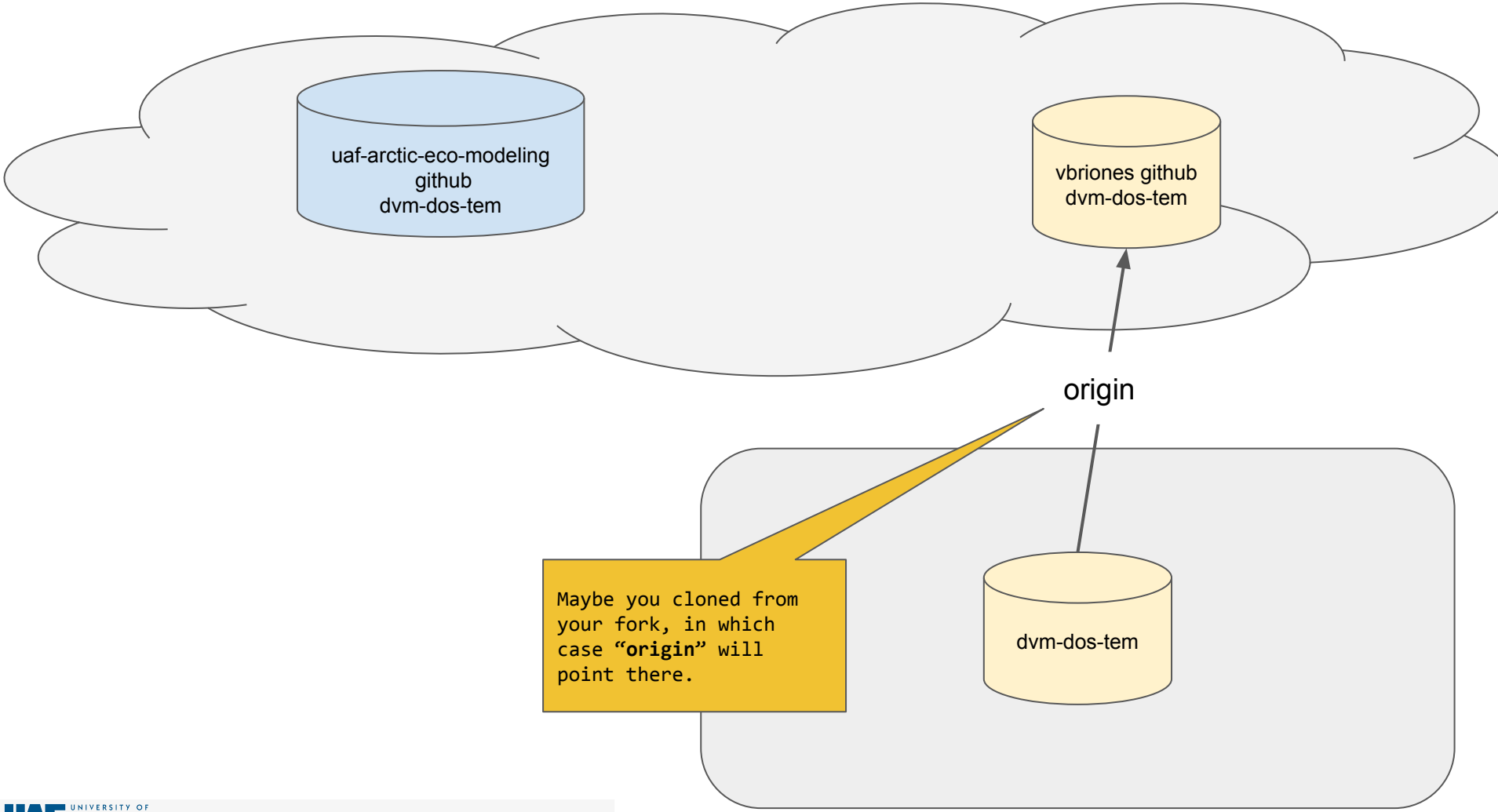
- A laptop.
- A server at your institution.
- A cloud instance (GCE, Amazon, Heroku, etc).
- An automated build server.
- etc.

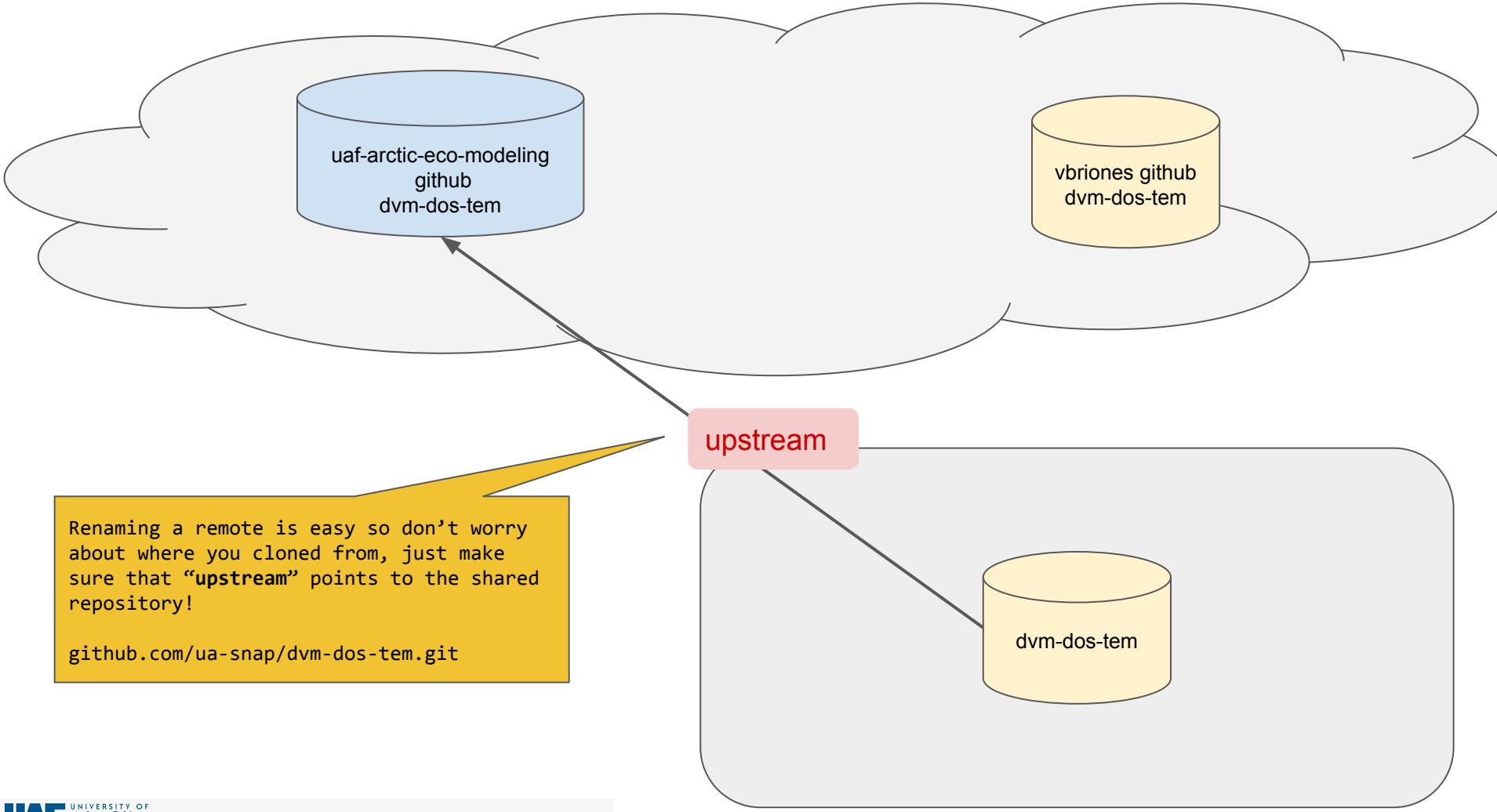


When you “fork” a project on Github, you make a copy of the repository under your Github account.


uaf-arctic-eco-modeling
github
dvm-dos-tem
vbriones github
dvm-dos-tem
clone
When you “clone” a git repository, you are making a copy of it.
dvm-dos-tem

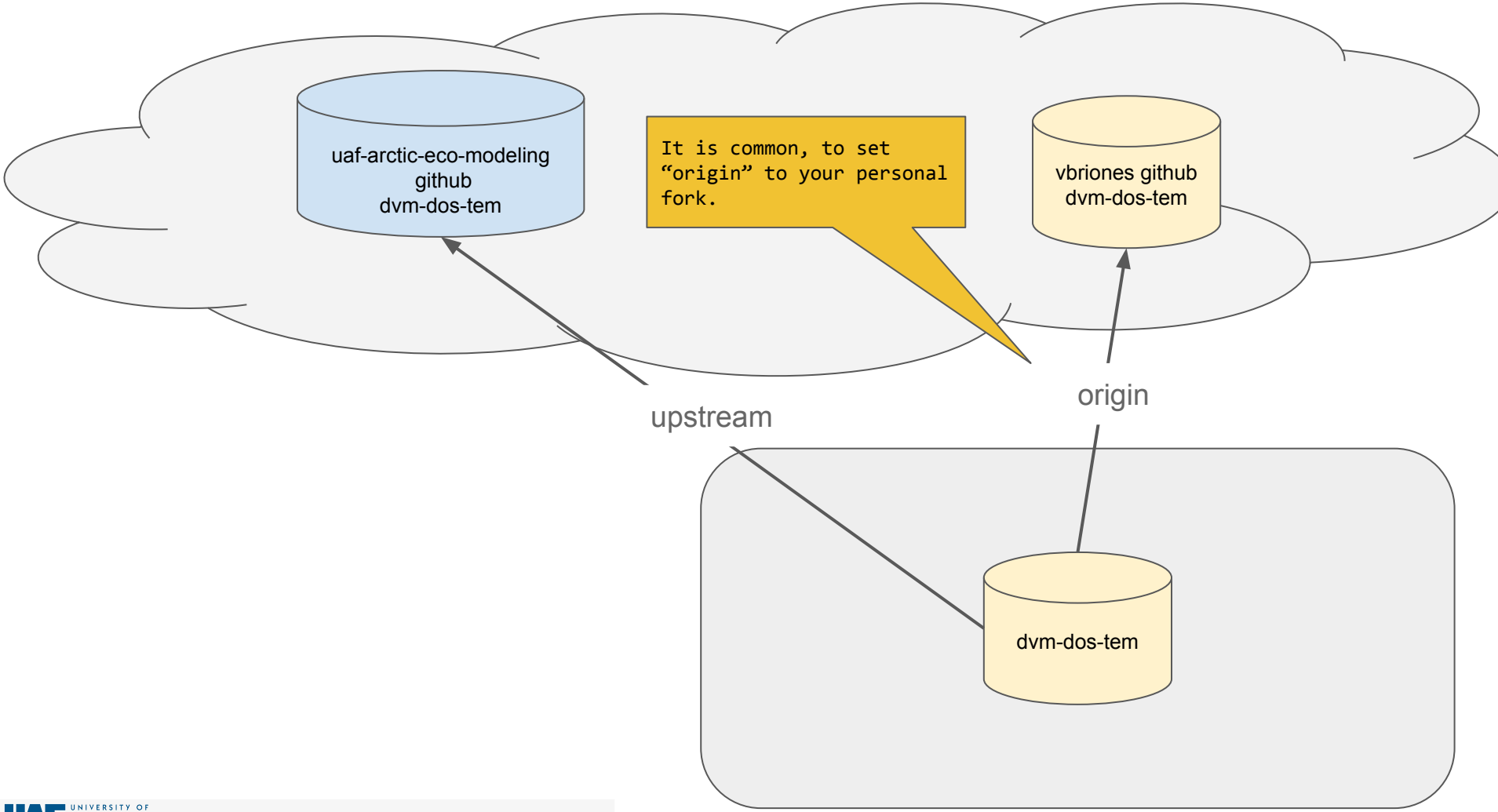

uaf-arctic-eco-modeling
github
dvm-dos-tem
vbriones github
dvm-dos-tem
origin
By default **“origin”** is the name of the remote repository that you cloned from.
dvm-dos-tem

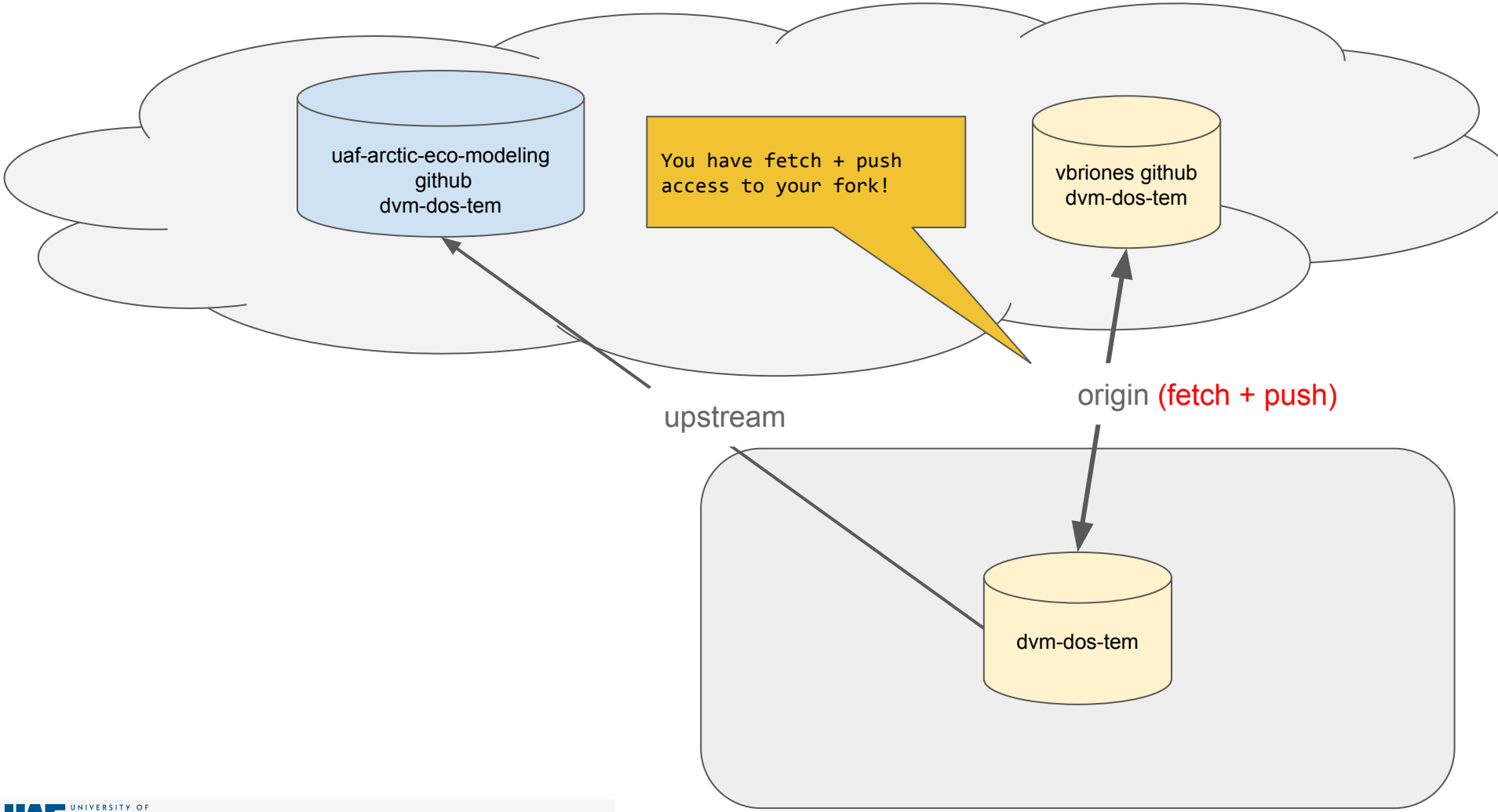

uaf-arctic-eco-modeling
github
dvm-dos-tem
vbriones github
dvm-dos-tem
origin
Maybe you cloned from your fork, in which case “**origin**” will point there.
dvm-dos-tem

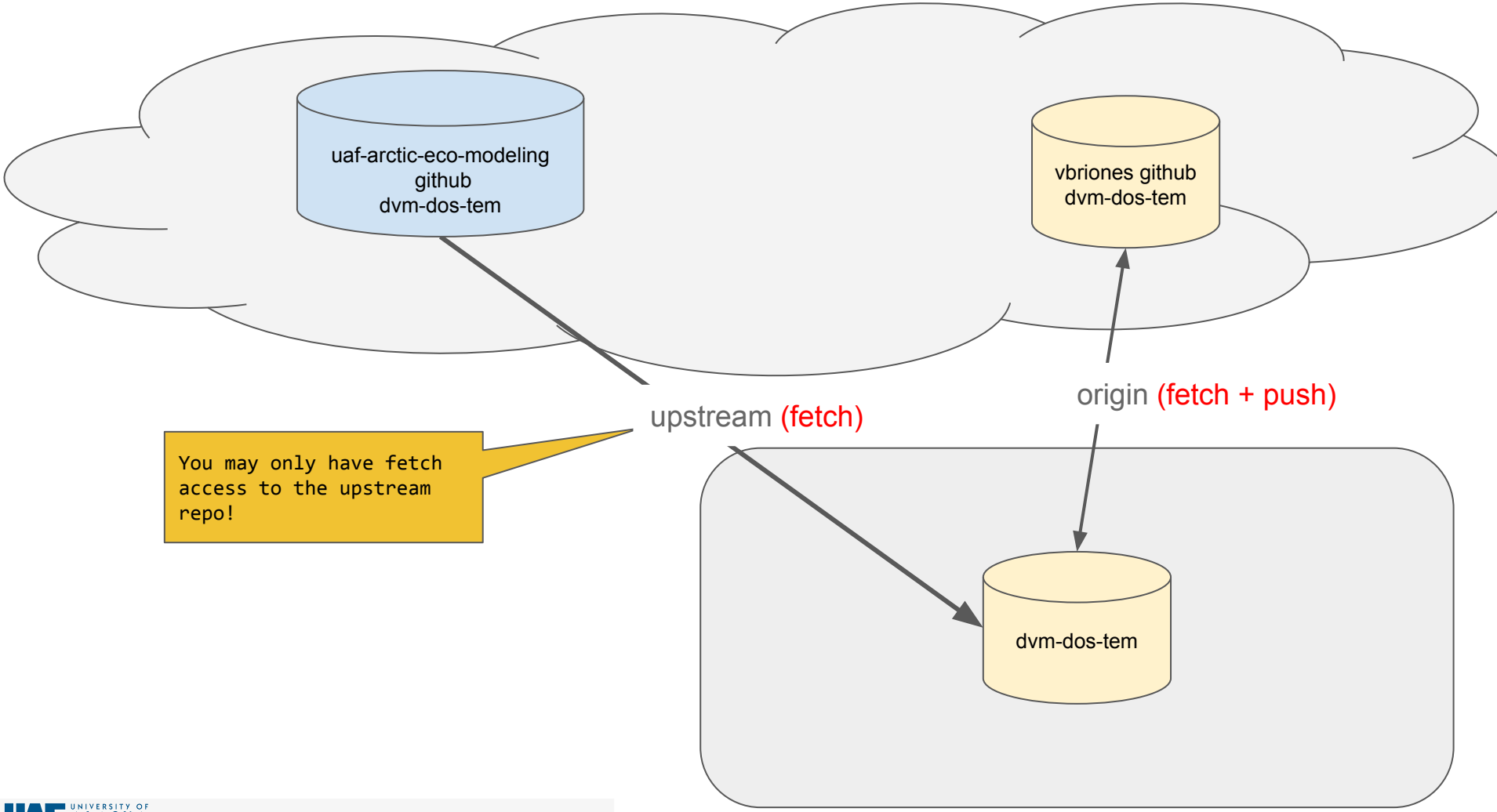


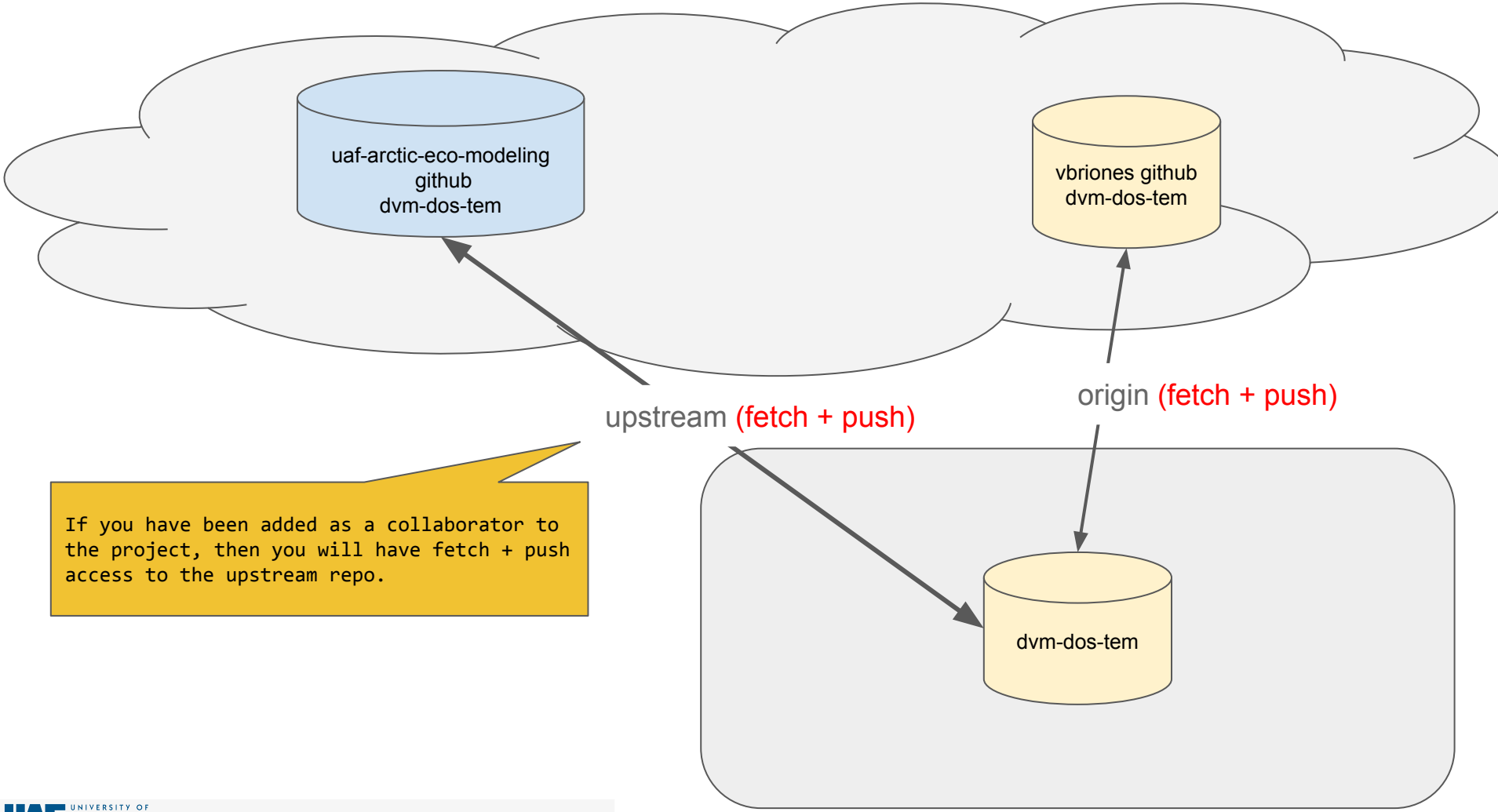
Renaming a remote is easy so don't worry about where you cloned from, just make sure that "**upstream**" points to the shared repository!

github.com/ua-snap/dvm-dos-tem.git

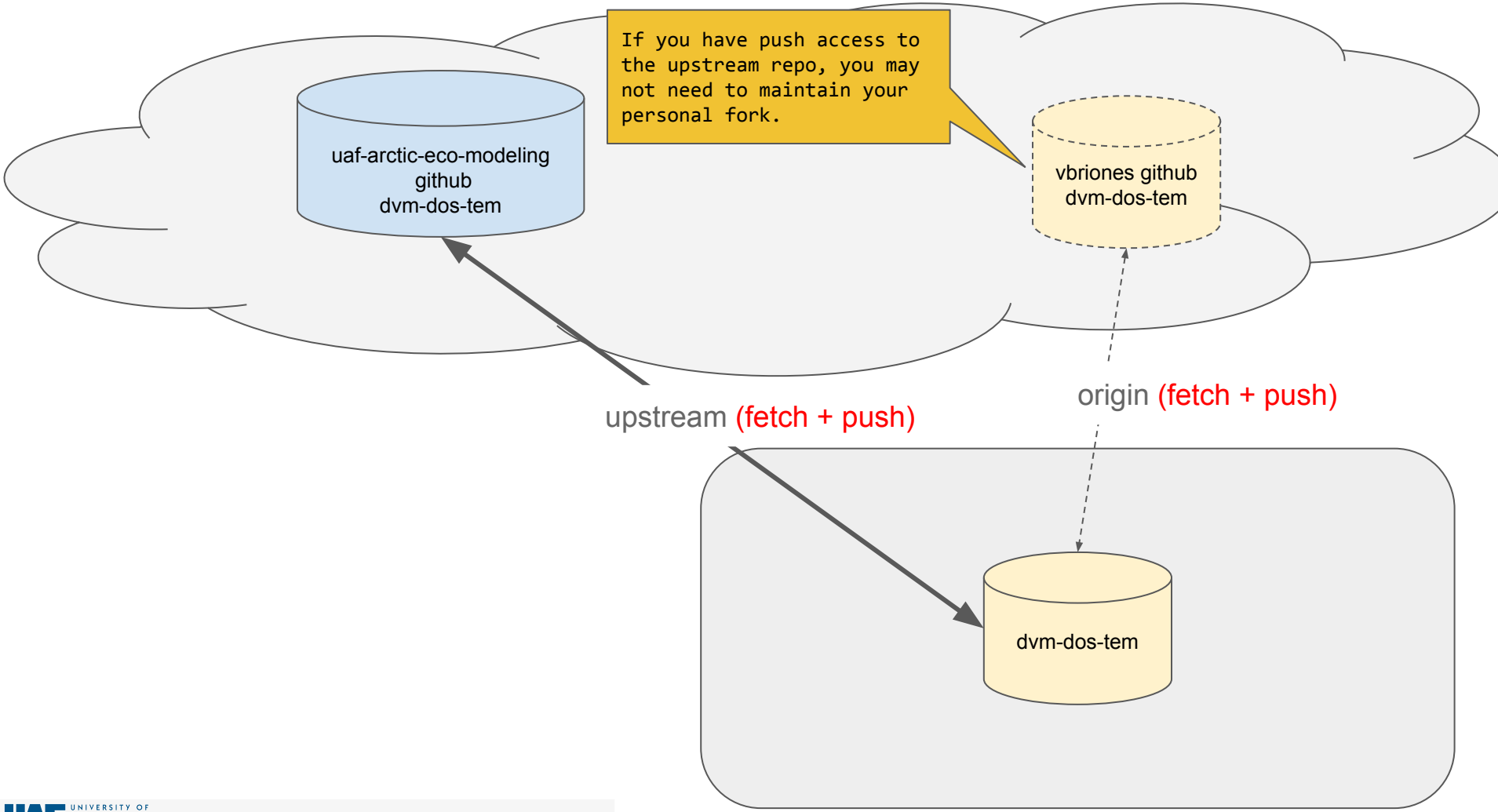

uaf-arctic-eco-modeling
github
dvm-dos-tem
It is common, to set "origin" to your personal fork.
vbriones github
dvm-dos-tem
upstream
origin
dvm-dos-tem

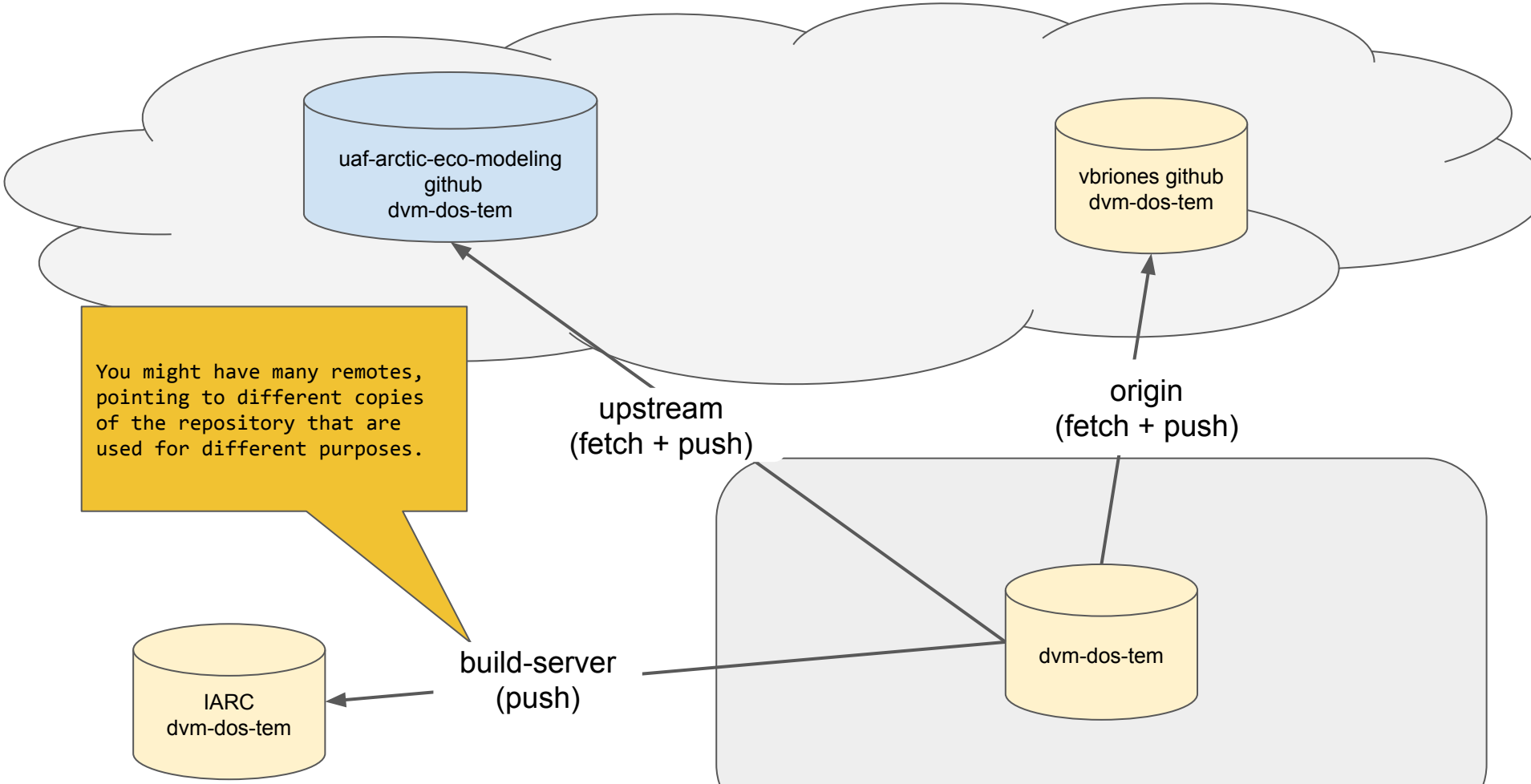

uaf-arctic-eco-modeling
github
dvm-dos-tem
You have fetch + push
access to your fork!
vbriones github
dvm-dos-tem
upstream
origin (fetch + push)
dvm-dos-tem


uaf-arctic-eco-modeling
github
dvm-dos-tem
vbriones github
dvm-dos-tem
upstream (fetch)
origin (fetch + push)
You may only have fetch access to the upstream repo!
dvm-dos-tem

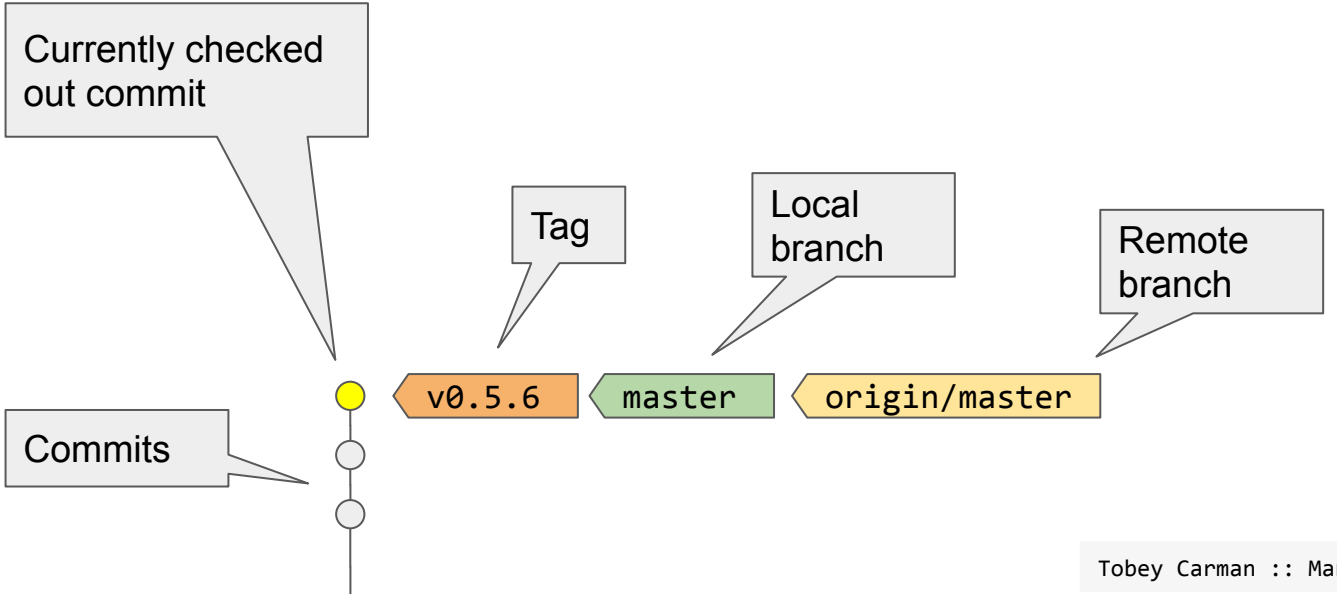


If you have been added as a collaborator to the project, then you will have fetch + push access to the upstream repo.

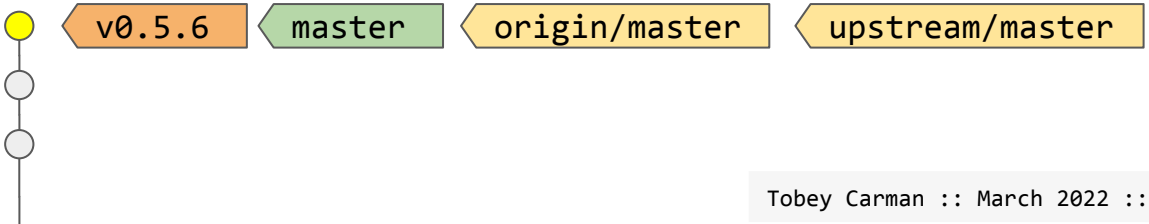

If you have push access to the upstream repo, you may not need to maintain your personal fork.
uaf-arctic-eco-modeling
github
dvm-dos-tem
vbriones github
dvm-dos-tem
upstream (fetch + push)
origin (fetch + push)
dvm-dos-tem


uaf-arctic-eco-modeling
github
dvm-dos-tem
vbriones github
dvm-dos-tem
You might have many remotes, pointing to different copies of the repository that are used for different purposes.
upstream
(fetch + push)
origin
(fetch + push)
dvm-dos-tem
IARC
dvm-dos-tem
build-server
(push)

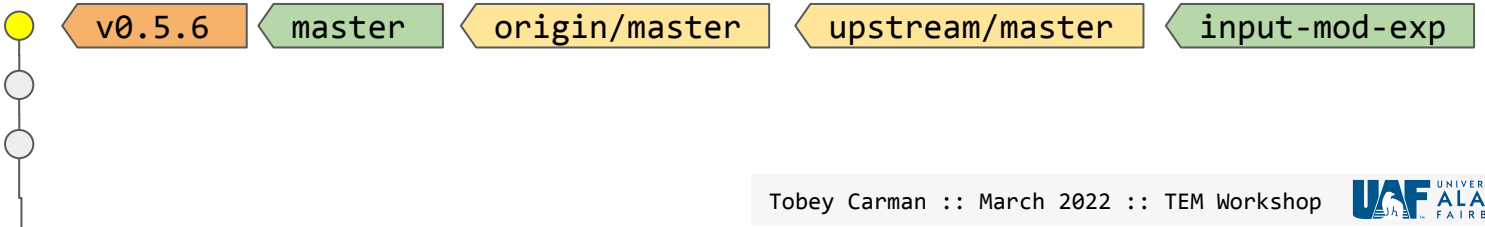
Single developer, using topic branch(es) and pull requests to contribute code to upstream repository.



```
$ git checkout master
```

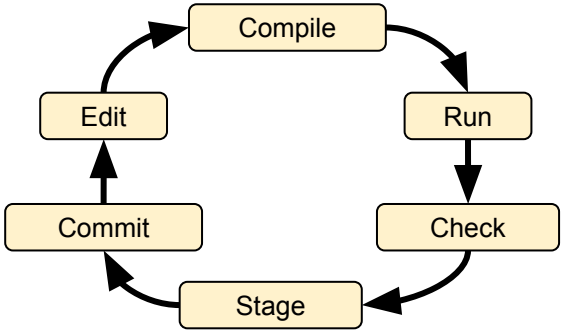


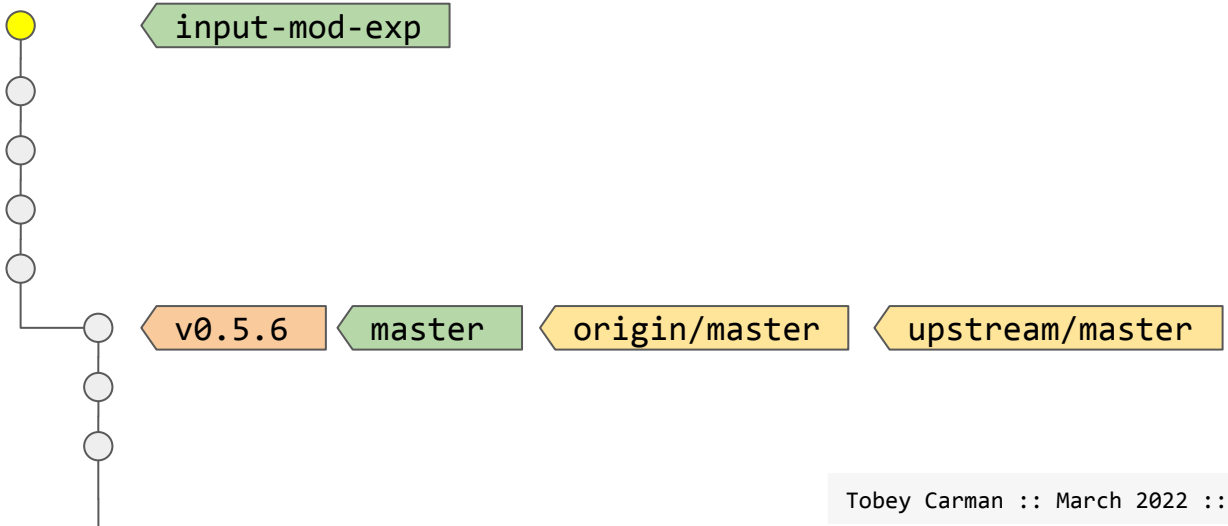
```
$ git checkout -b input-mod-exp
```



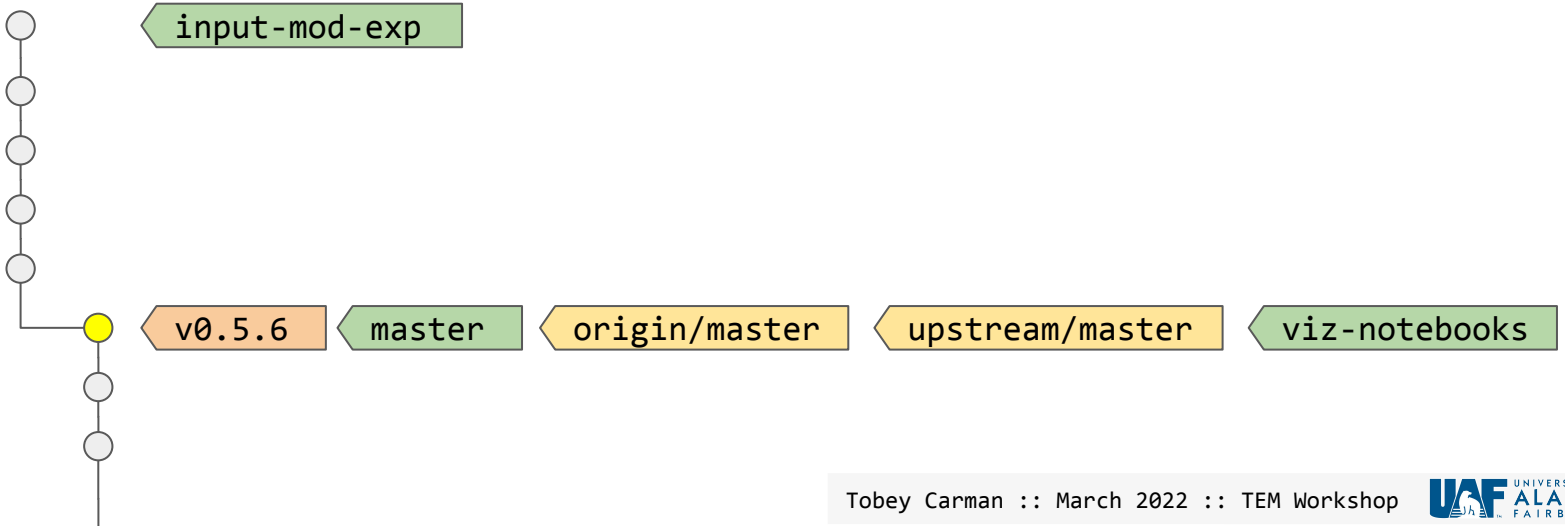
Core Developer Process Loop
with
version control

i.e. programming!!



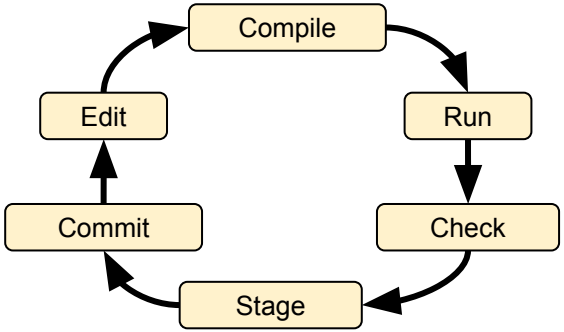


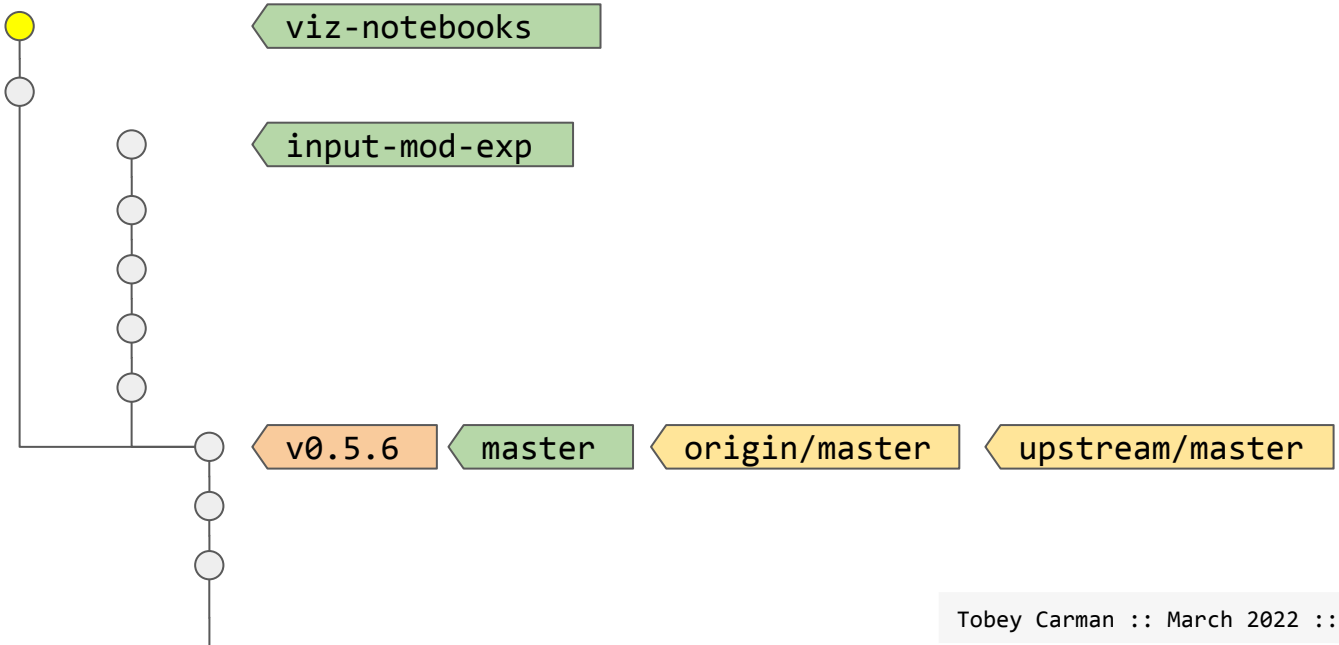
```
$ git checkout master
$ git checkout -b viz-notebooks
```



Core Developer Process Loop
with
version control!

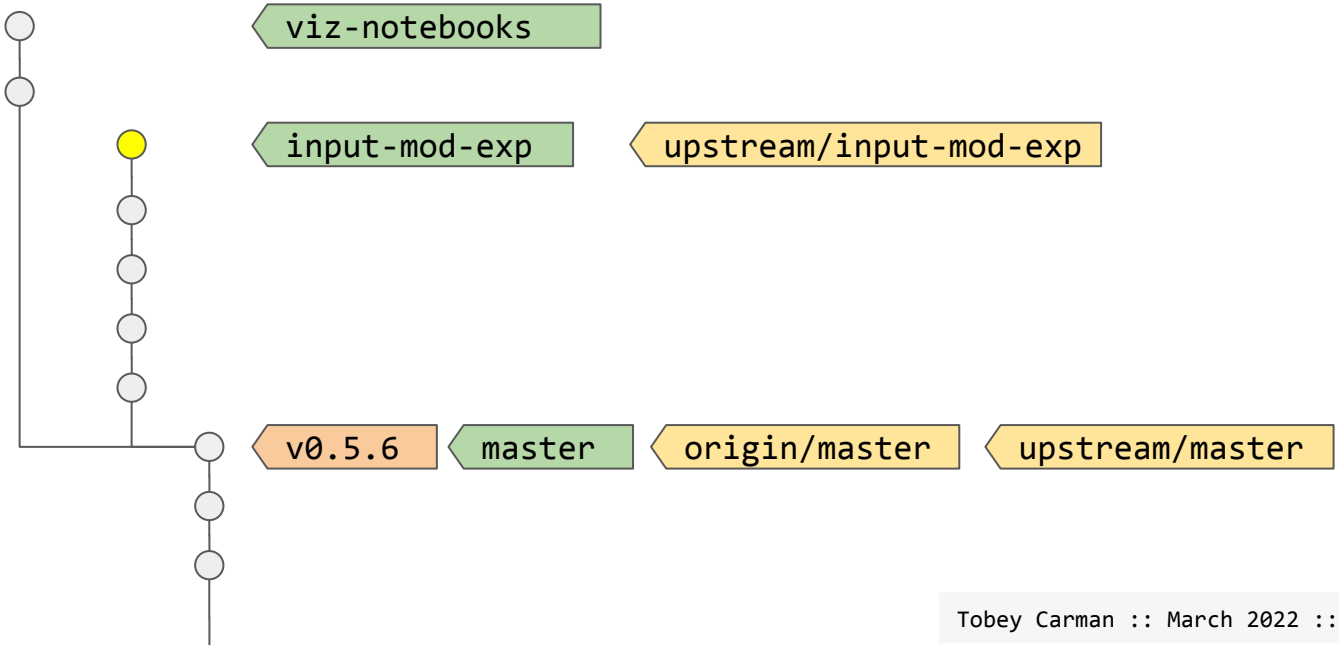
i.e. programming!!





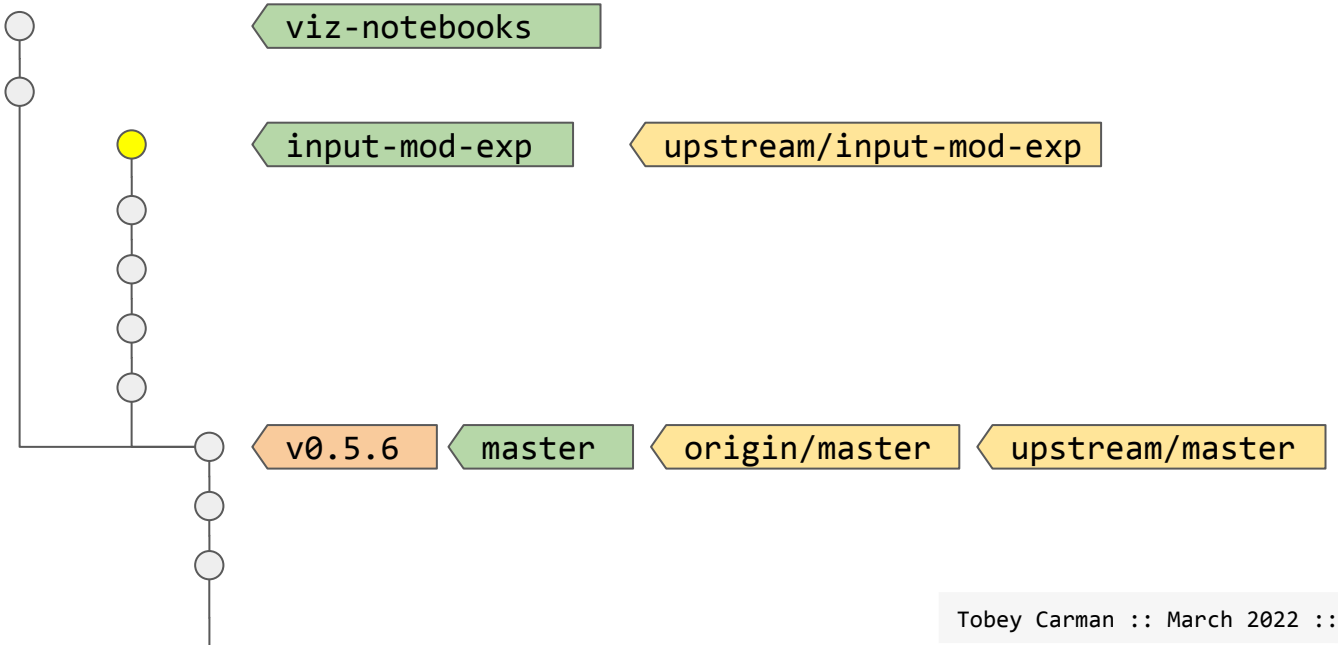
Share work with other people (and backup to cloud)

```
$ git checkout input-mod-exp
$ git push upstream input-mod-exp
```



Github website:

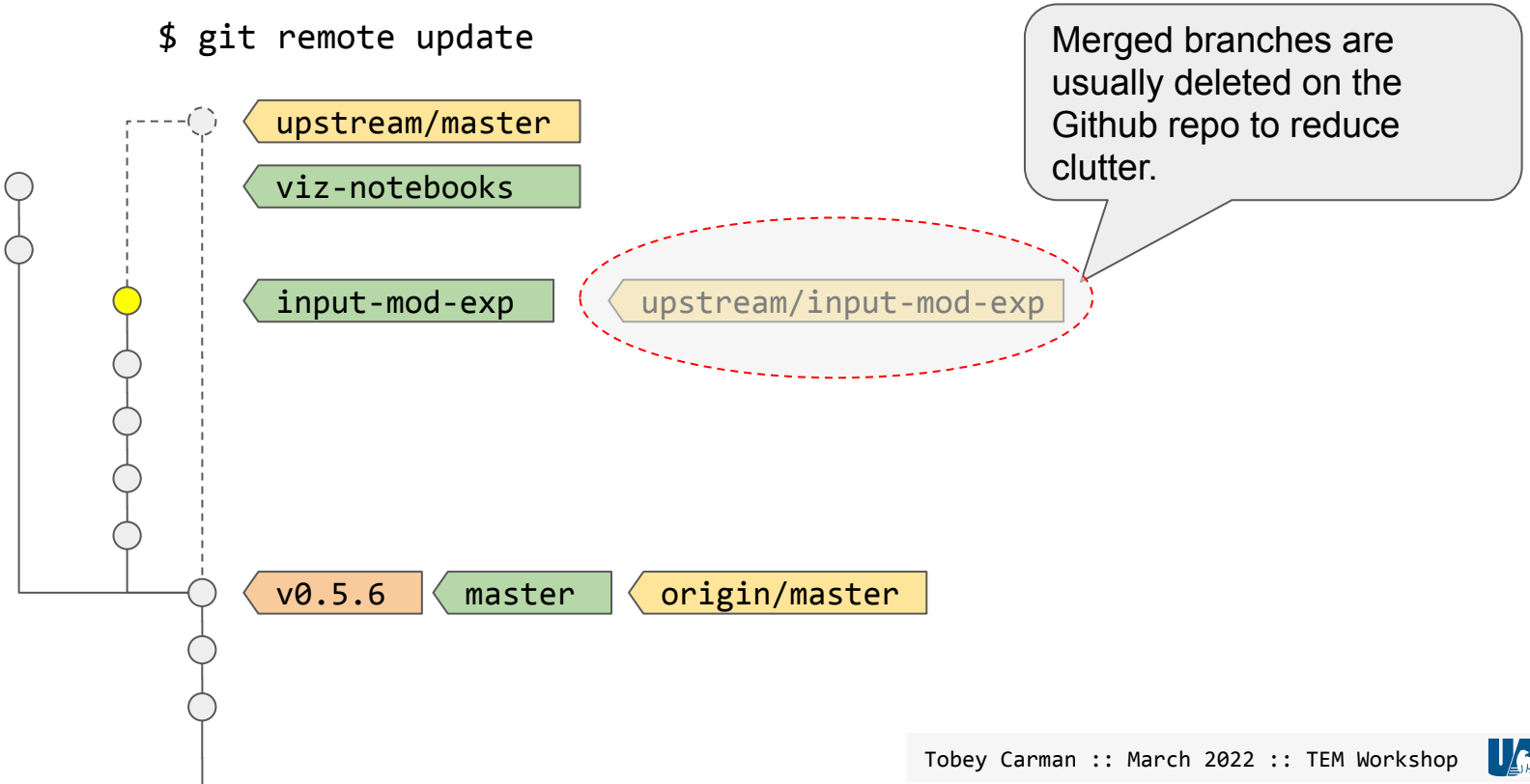
- Create Pull Request from input-mod-exp branch → master
- Group review, test, discussion, etc



Github website: tcarman2 or rarutter merges PR 🎉

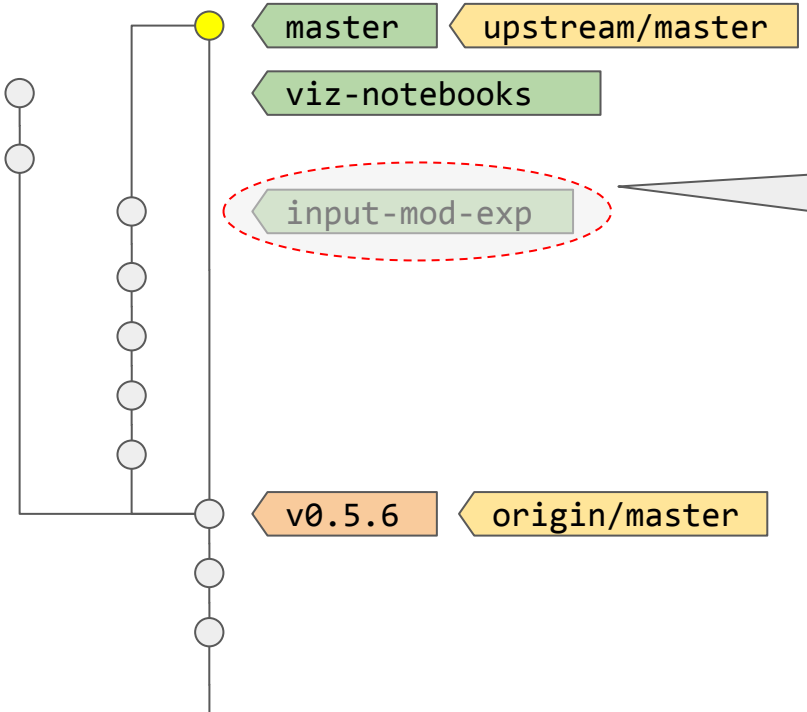
See what happened upstream w/o changing anything locally:

```
$ git remote update
```



Update from upstream after new code merged to upstream/master

```
$ git checkout master
$ git pull upstream master
```

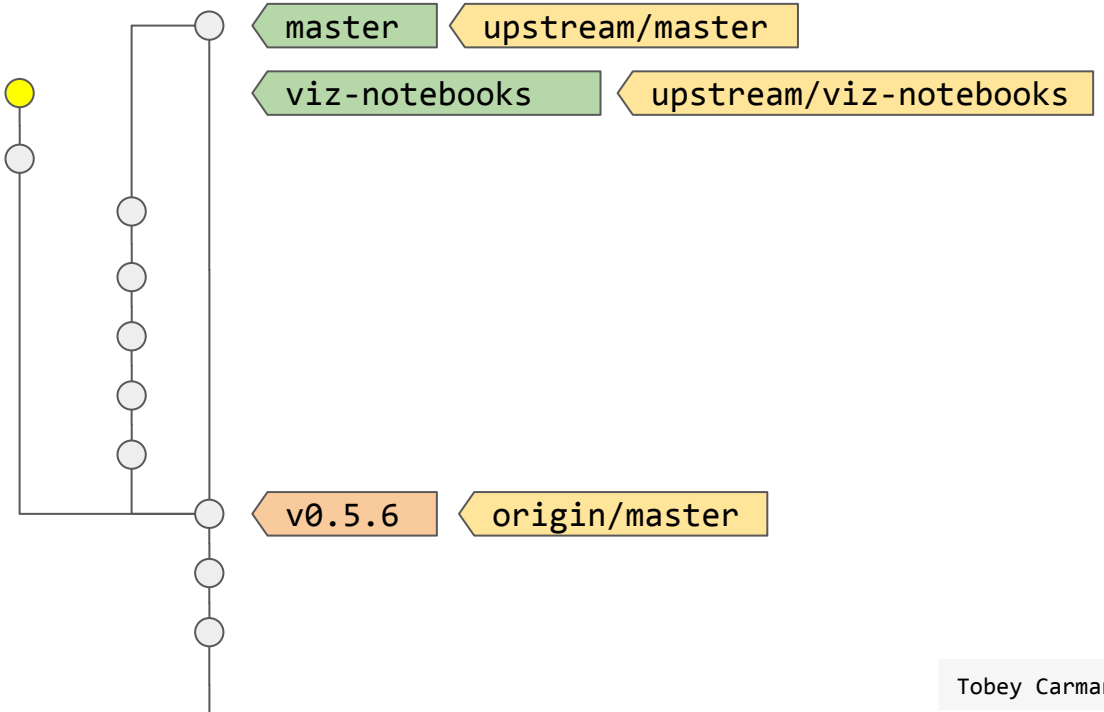


You may optionally delete this branch if you want to reduce clutter locally.

```
$ git branch -d input-mod-exp
```

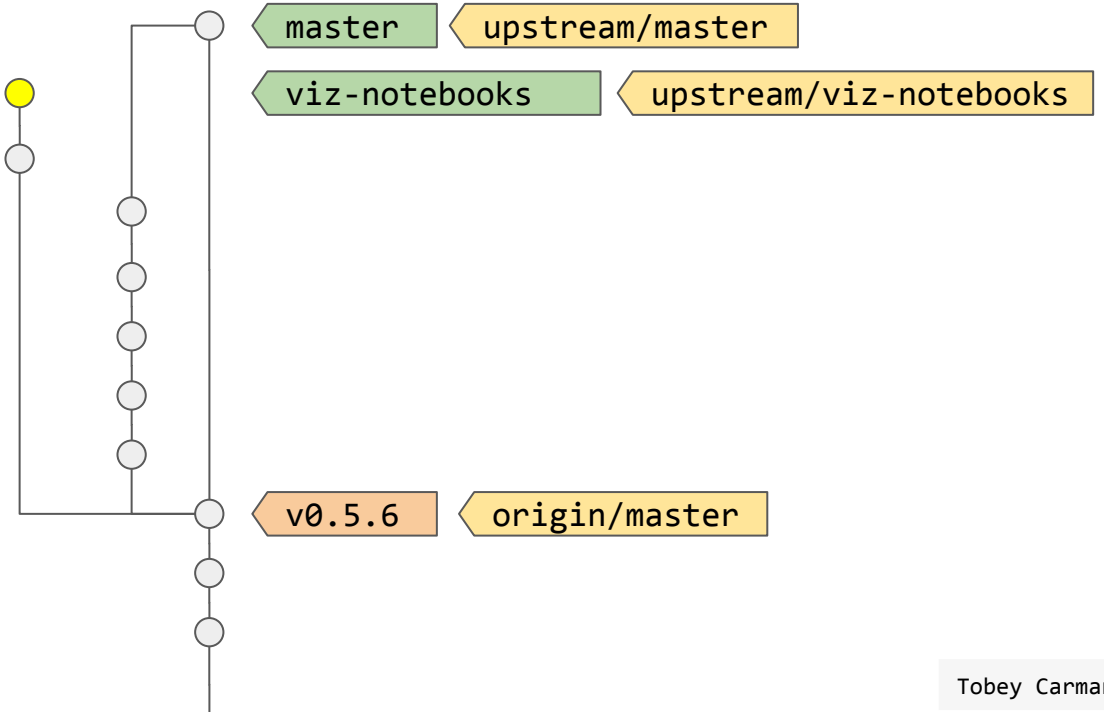
Share work on viz-notebooks branch (and backup to cloud)

```
$ git checkout viz-notebooks
$ git push upstream viz-notebooks
```



Github website:

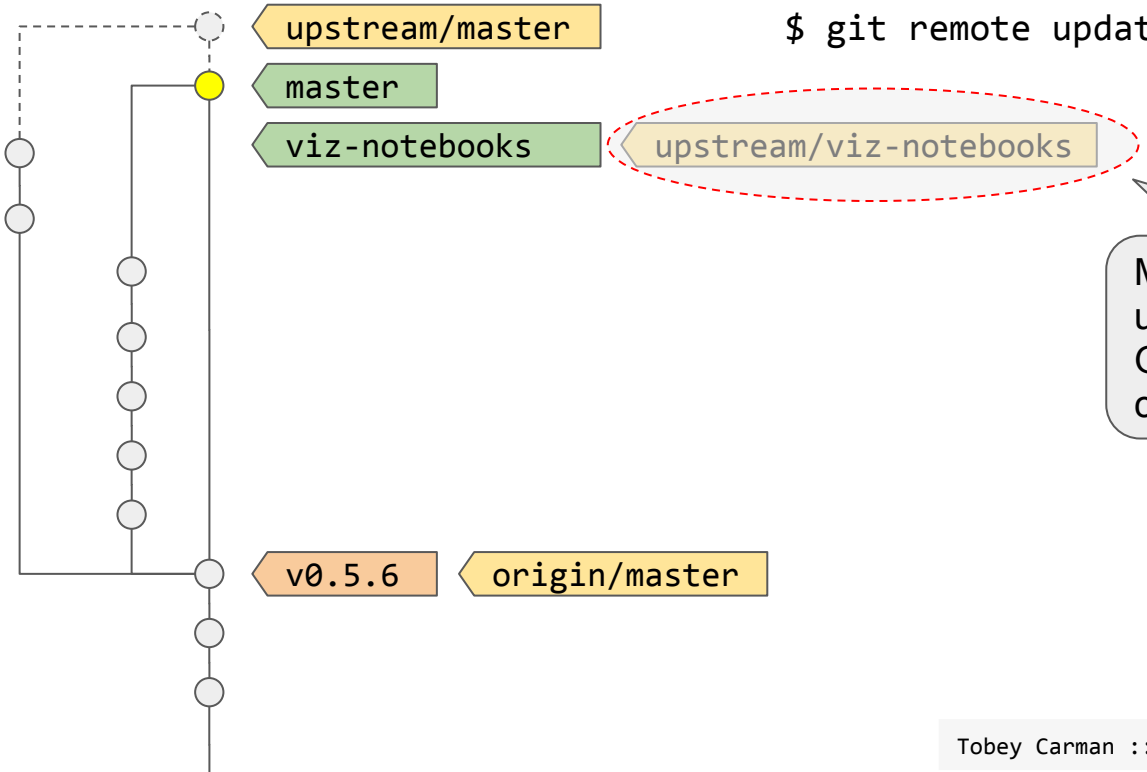
- Create Pull Request from viz-notebooks → master
- Group review, test, discussion, etc



Github website: tcarman2
or rarutter merges PR 🎉

See what happened upstream w/o changing anything locally:

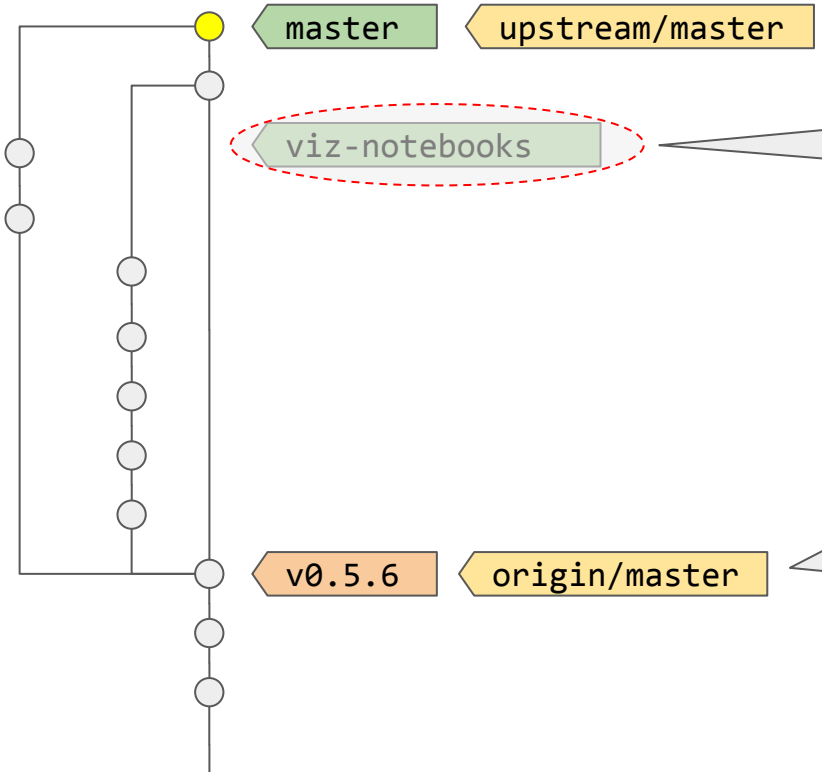
```
$ git remote update
```



Merged branches are usually deleted on the Github repo to reduce clutter.

Update from upstream after new code merged to upstream/master

```
$ git checkout master
$ git pull upstream master
```



You may optionally delete this branch if you want to reduce clutter locally.

```
$ git branch -d viz-notebooks
```

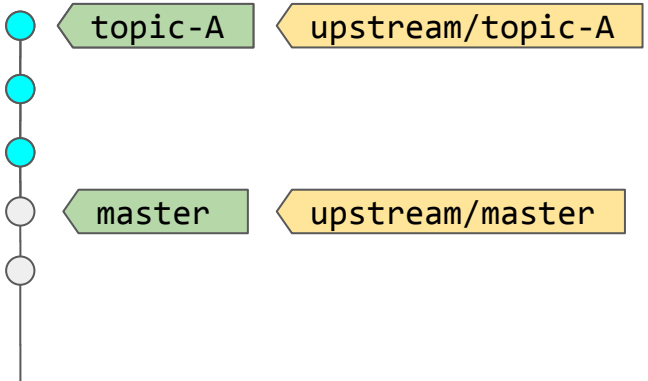
Note that we have left the `origin/master` branch behind. If you are maintaining your fork, then you would push from your master back to `origin/master`. In many cases, you don't need to maintain your own fork on Github.

Two developers working on a single topic branch and staying in sync using

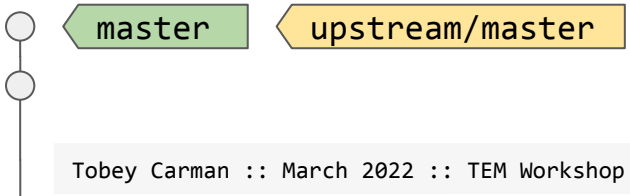
```
$ git pull --rebase
```

Person Y

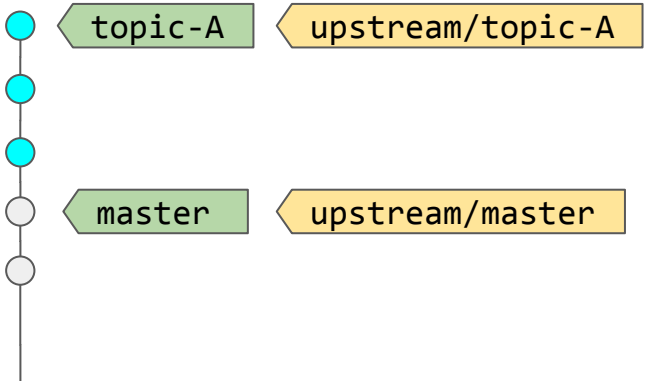
```
$ git checkout master
$ git checkout -b topicA

# edit, compile, run, check commit, 3x

$ git push upstream topicA
```



Person Z

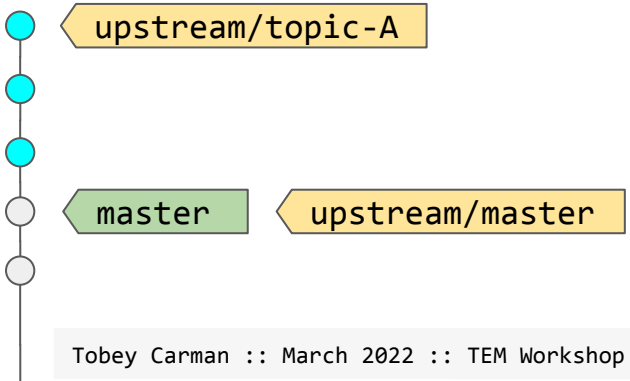


Person Y



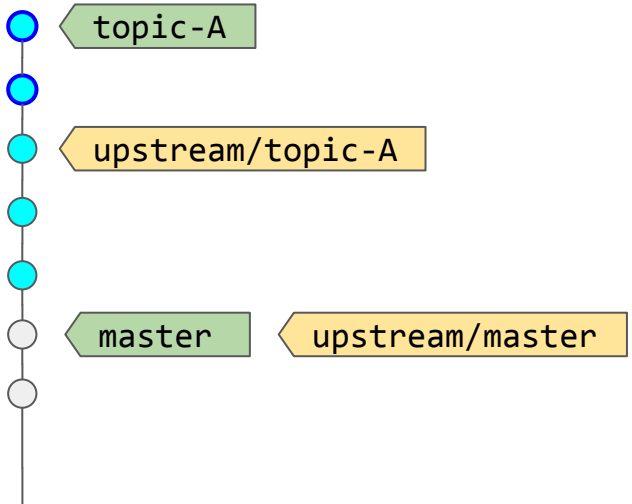
Person Z

```
$ git remote update
```



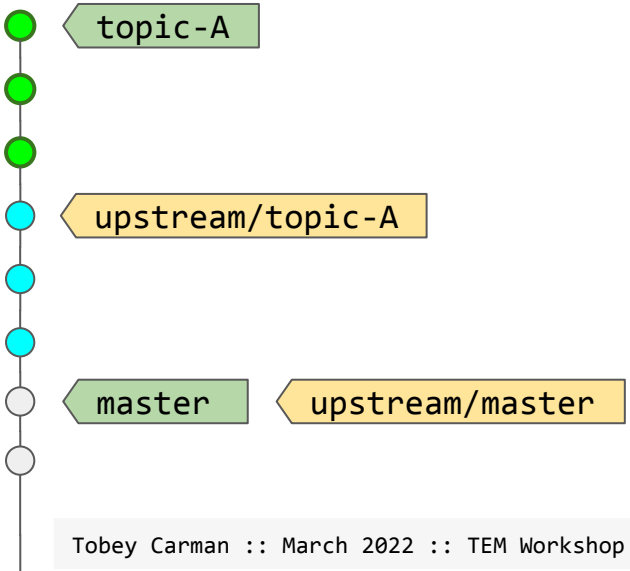
Person Y

```
# edit, compile, run, check commit, 2x
```



Person Z

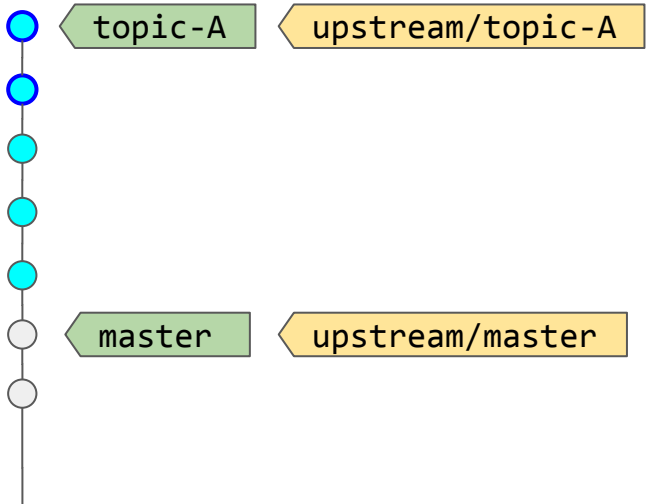
```
$ git checkout topicA

# edit, compile, run, check commit, 3x
```

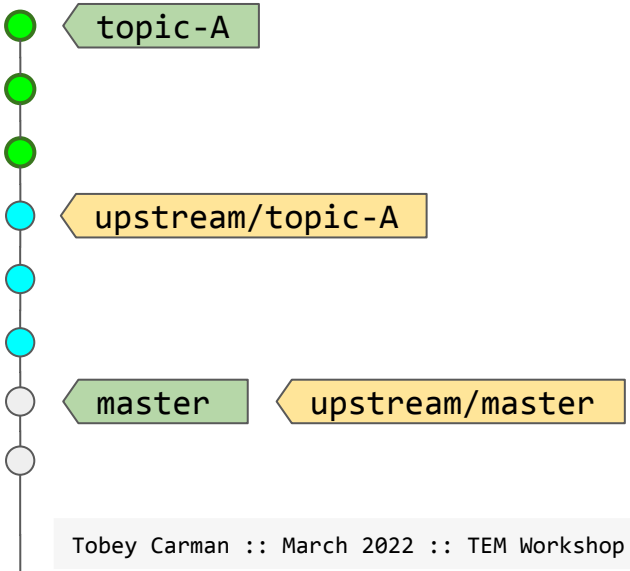


Person Y

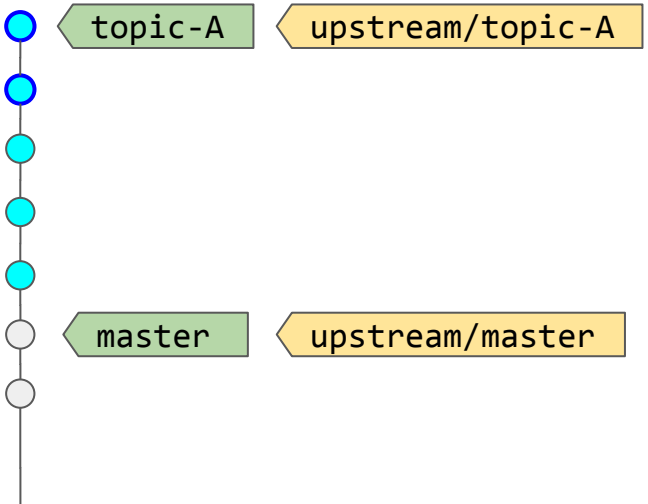
Person Z

```
# Check that no one else pushed first!!
$ git remote update

# Then push...
$ git push upstream topicA
```



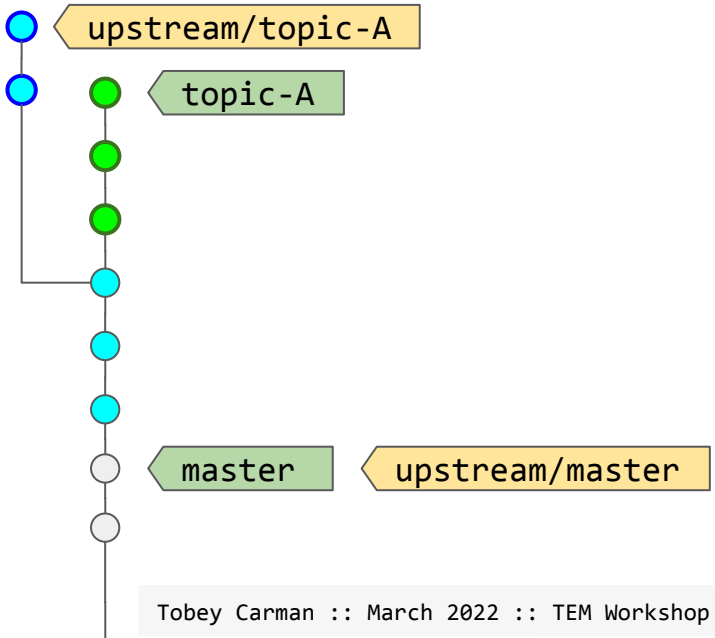


Person Y

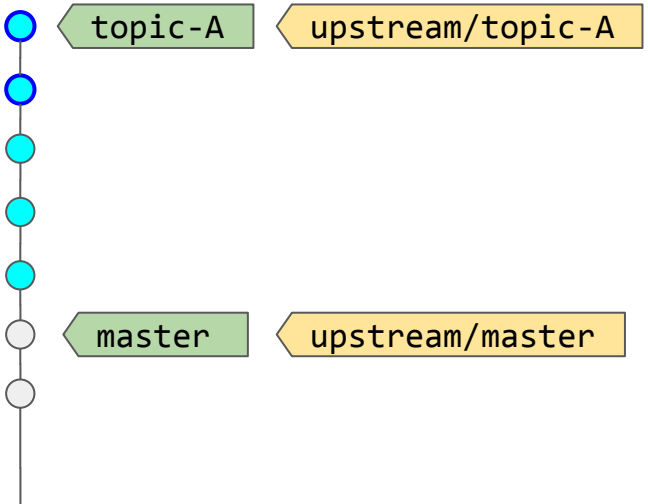


Person Z

```
$ git remote update
```

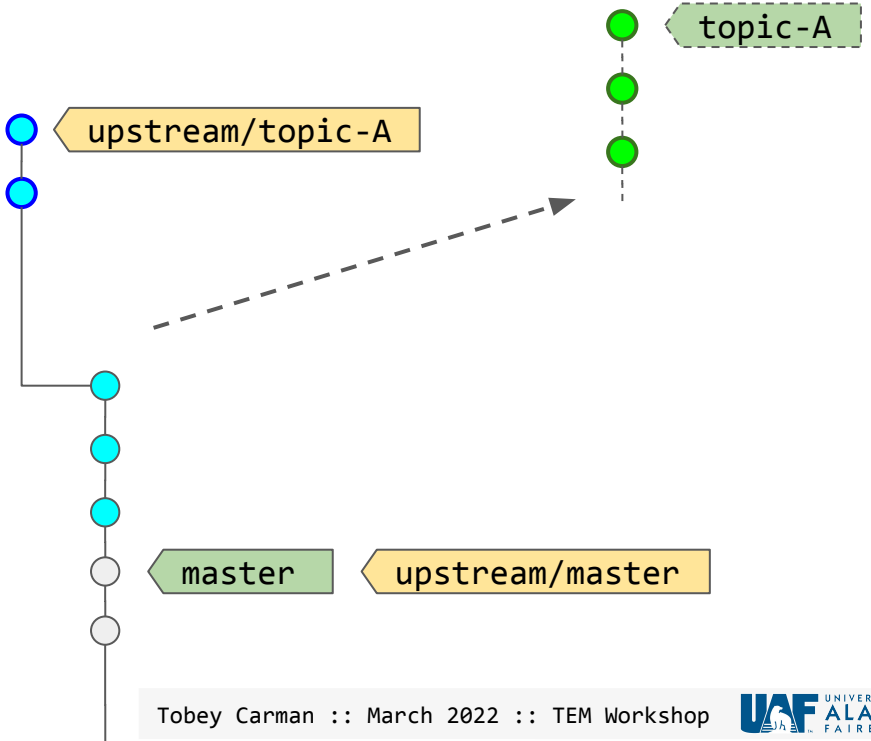


Person Y

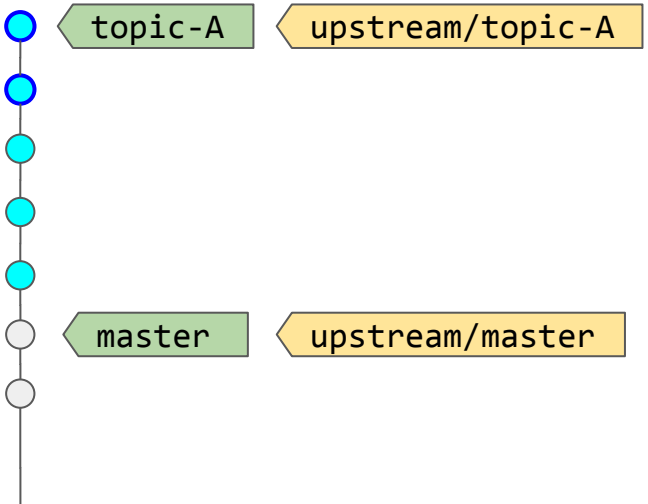


Person Z

```
$ git pull --rebase upstream topicA
```

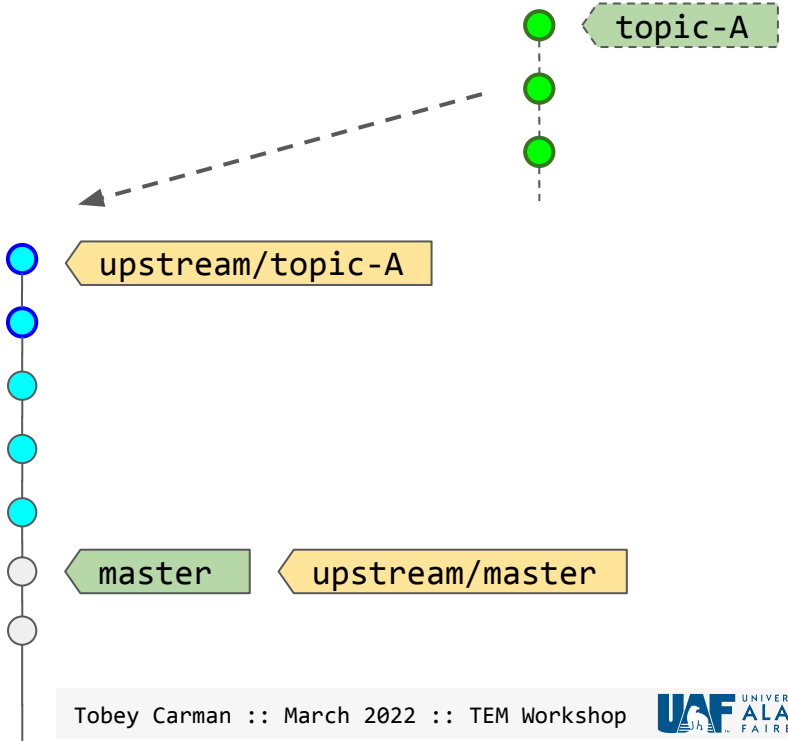


Person Y

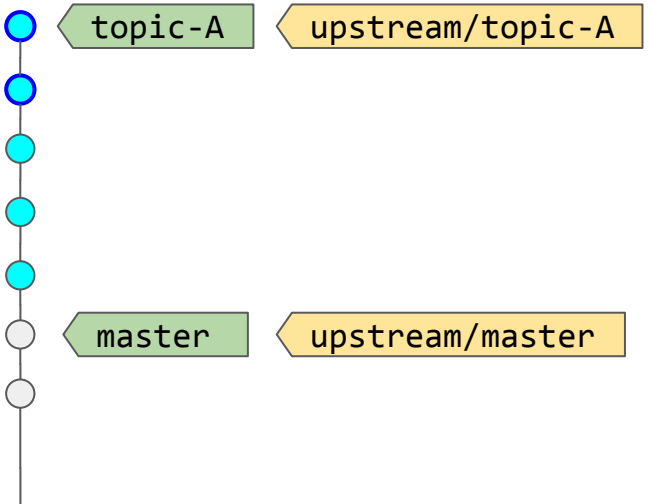


Person Z

```
$ git pull --rebase upstream topicA
```

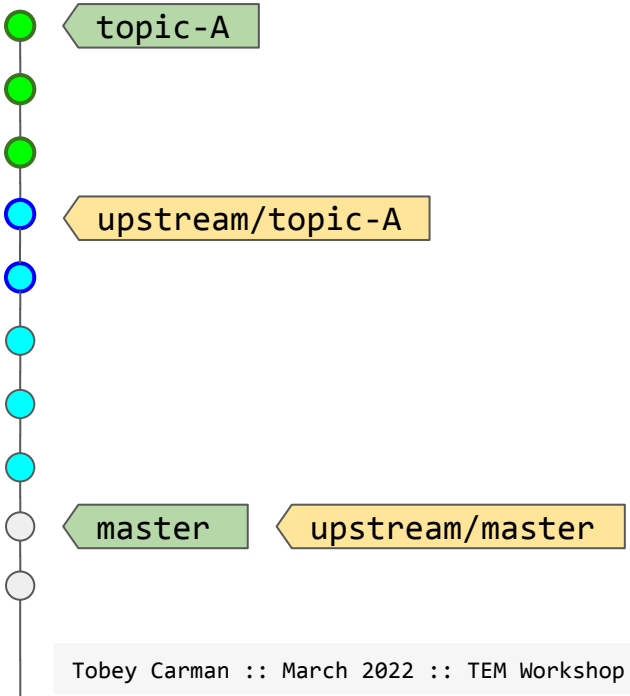


Person Y

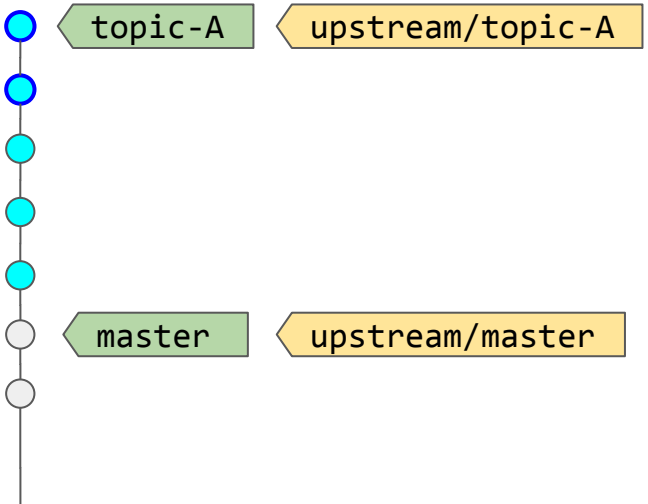


Person Z

```
$ git pull --rebase upstream topicA

COMPLETE!
```

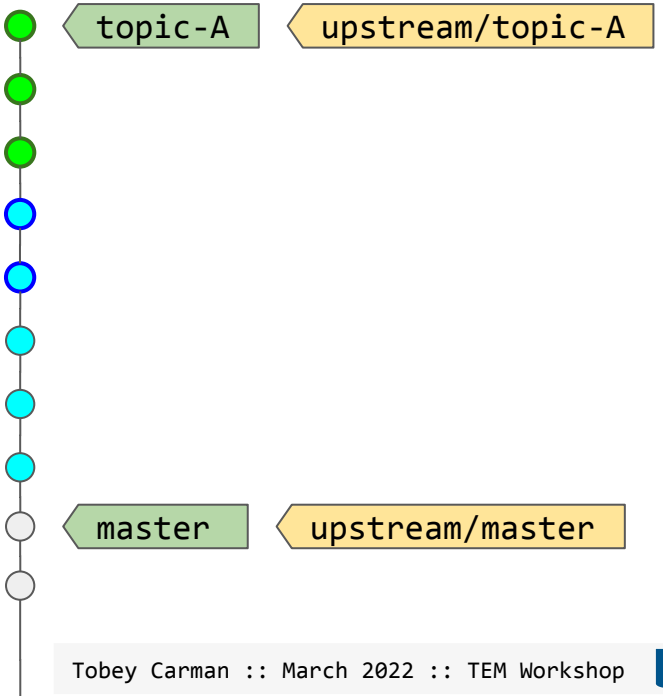


Person Y



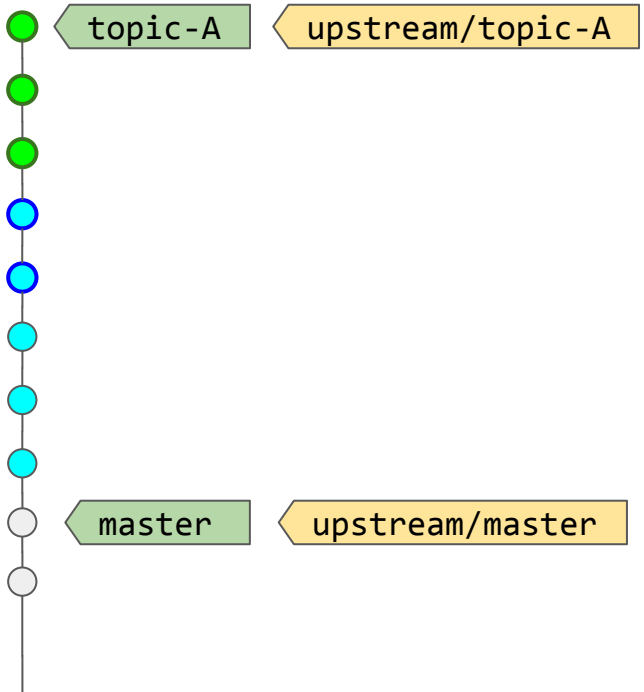
Person Z

```
$ git push upstream topicA
```

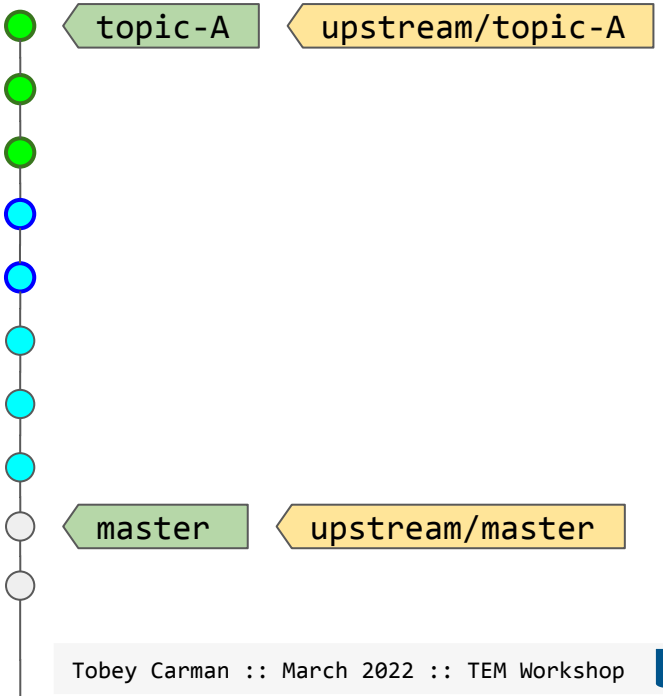


Person Y

```
$ git pull --rebase upstream topicA
```

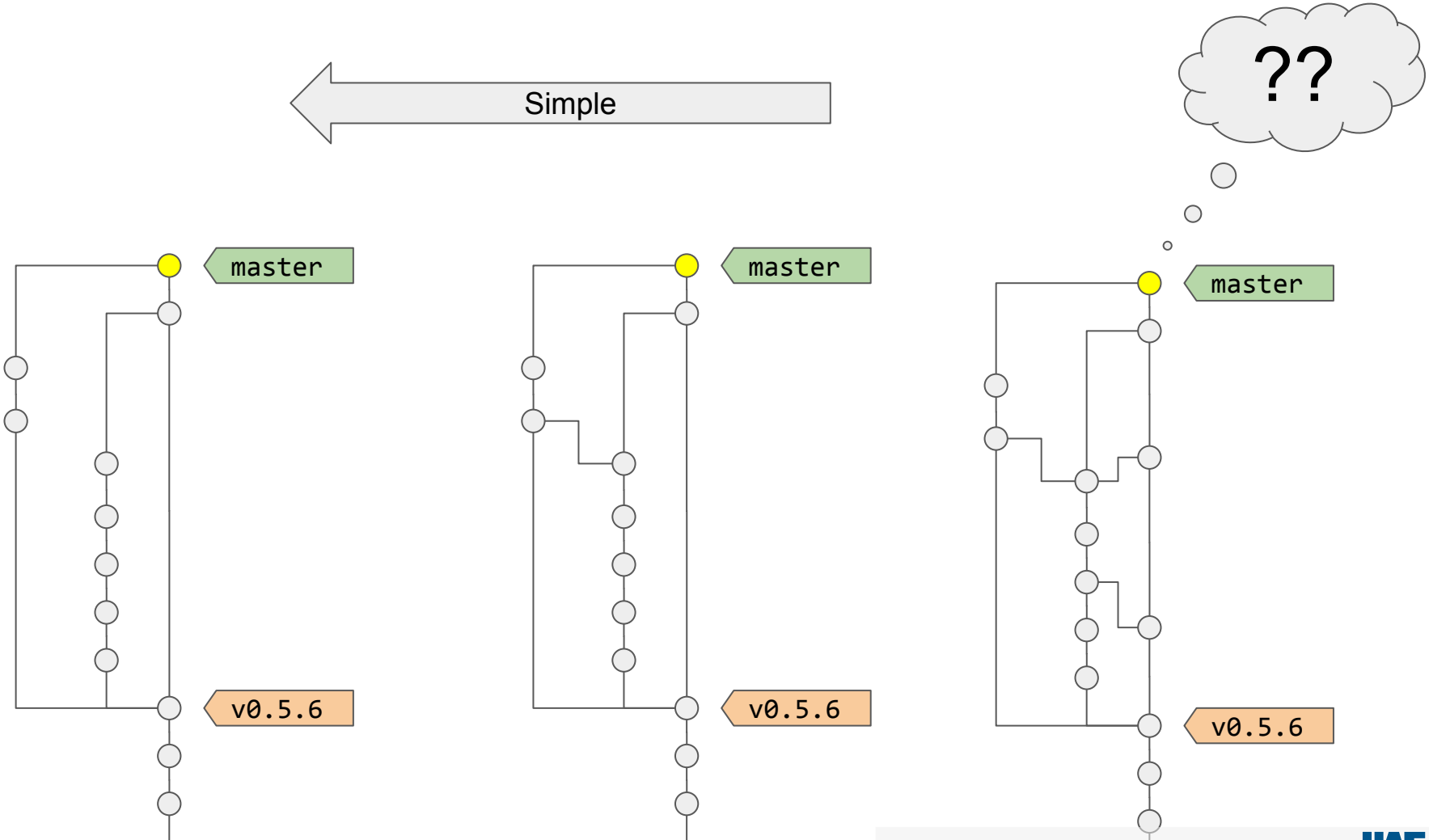


Person Z



When using git, keep in mind what should and should not be tracked. In light of the previous workflows, what would happen if:

- You tracked model outputs?
- You tracked model configurations for runs?
- Every possible collaborator tracked model configurations for their runs?
- Person Y indents with 2 spaces and Person Z indents with 4 spaces?
- Person Y and Z track their personal setup files (.vscode, .vimrc, etc)


Simple
??
master
v0.5.6
master
v0.5.6
master
v0.5.6