Git Introduction

Presented to
ATPESC 2017 Participants

Alicia Klinvex
Sandia National Laboratories

Q Center, St. Charles, IL (USA)
Date 08/09/2017

In this talk, you will learn…

- What is version control?
- What is the difference between a workspace, staging area, local repository, and upstream repository?
- How do I record changes to my files?

- We will *not* cover every git command, but you will have a foundation that allows you to pick up the rest easily.
- We will *not* debate whether git, svn, mercurial, etc is best.

What not to do: my grad school workflow

- I wrote an eigensolver code that I used on several different machines (lanczos, golub, endeavor)
- If someone else wanted a copy, I gave them a tarball
- Sometimes, I saved the tarball on dropbox, just in case
- **What could possibly go wrong?**

Why this workflow is suboptimal

- How do you make sure the code being used is the same on all three machines, since it's under active development?
- How do my colleagues get updates to the code?
- If I break something, how do I get back to an unbroken state?
- If my computer caught fire, how much of my work would disappear forever?

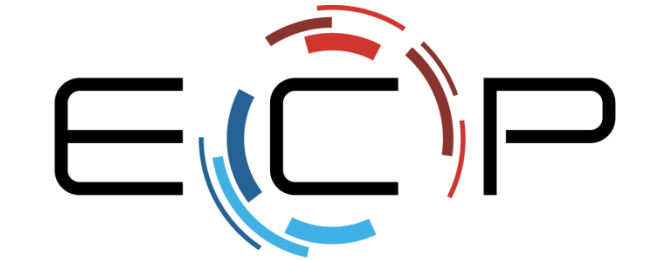
What could I have done differently?

- Use distributed version control (like git)!
- Version control: a category of software tools that help a software team manage changes to source code over time
- Keeps track of every modification to the code in a special database called a *repository*

How would that help?

- If I broke my code, I could go back to an earlier version, because it would be stored in the repository
- Colleagues could get my latest updates without even talking to me
- I could synchronize my work across the three different machines
- Because the distributed repository isn't stored on my machine, the risk of me losing everything is much lower.

And now, an analogy about food photography to help us understand git…



Step 1: Prepare the stuff you're photographing

- The place where you prepare the stuff is called your *workspace*

source: guff.com

Step 2: Put things in the staging area

- The staging area is the well lit spot with a backdrop that the camera's pointed at
- It's where you put things that are ready to be photographed
- You might not move everything from your workspace to the staging area

source: petapixel.com

Step 3: Take a picture and stick it in your album

- The pictures in a photo album are in a linear order
- The stuff in the workspace is irrelevant; you aren't photographing the workspace
- The entire staging area gets photographed though

source: beetsandbones.com

Now back to git!

1. Make changes to your code in the workspace
2. Move the desired changes to the staging area
3. Take a snapshot and put it in the repo
   - Things in the staging area are part of the snapshot
   - Things in the workspace are left out

Let’s make a repository!

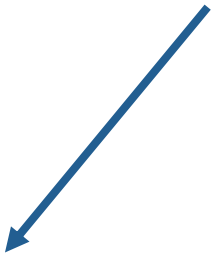
- I have a bunch of Matlab files demonstrating Krylov solver concepts, and some png images generated by those files
-
```
[amklinv@klogin2 krylov]$ ls
gmres.png      jacobi.m    krylov.m    mmread.m   mygmres.m  sd.png
gmres_test.m   jacobi.png  krylov.png  mmwrite.m  rotmat.m   steepest_descent.m
```

Let’s make a repository!

- `[amklinv@klogin2 krylov]$ git init`
  `Initialized empty Git repository in /home/amklinv/krylov/.git/`
- What did this do?
  - Created an invisible directory called .git
  - This directory is our local repository
  - What’s in the local repository?

Let’s check the state of our workspace/staging area!

```
[amklinv@klogin2 krylov]$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       gmres.png
#       gmres_test.m
#       jacobi.m
#       jacobi.png
#       krylov.m
#       krylov.png
#       mmread.m
#       mmwrite.m
#       mygmres.m
#       rotmat.m
#       sd.png
#       steepest_descent.m
nothing added to commit but untracked files present (use "git add" to track)
```



Note that git helpfully tells us what to do next

What do we want to add to the staging area?

- We have two types of files
  - Matlab (text)
  - Images (binary)
- The images were generated by the Matlab files and can be easily regenerated
- We will only add the Matlab files
  - We can tell git to *ignore* the png files by modifying .gitignore
- Git best practices
  - Don’t store derivative content
  - Try not to store large binary files

Let’s add the Matlab files to the staging area!

- [amklinv@klogin2 krylov]$ **git add** *.m
- [amklinv@klogin2 krylov]$ git status
  # On branch master
  #
  # Initial commit
  #
  # Changes to be committed:
  #   (use "git rm --cached <file>..." to unstage)
  #
  #   new file: .gitignore
  #   new file: gmres_test.m
  #   new file: jacobi.m
  #   new file: krylov.m
  #   new file: mmread.m
  #   new file: mmwrite.m
  #   new file: mygmres.m
  #   new file: rotmat.m
  #   new file: steepest_descent.m

Where are my changes?

- Before git add…
  a) workspace
  b) staging area
  c) local repository

Where are my changes?

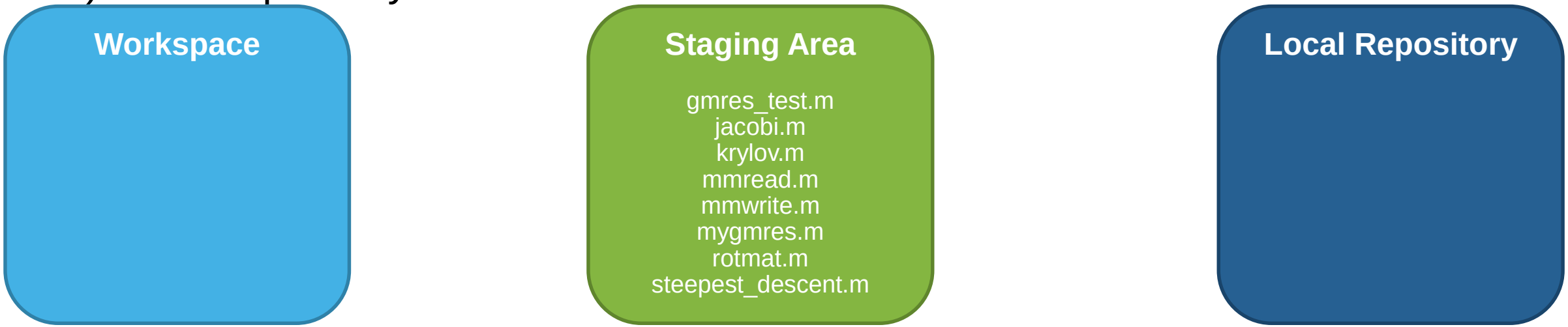
- Before git add…
  a) **workspace**
  b) staging area
  c) local repository

Where are my changes?

- After git add…
  a) workspace
  b) staging area
  c) local repository

Where are my changes?

- After git add…
  a) workspace
  b) staging area
  c) local repository



How do I get my files into the repository?

- `[amklinv@klogin2 krylov]$ git commit`

- # Please enter the commit message for your changes. Lines starting
  # with '#' will be ignored, and an empty message aborts the commit.
  # On branch master
  #
  # Initial commit
  #
  # Changes to be committed:
  #   (use "git rm --cached <file>..." to unstage)
  #
  #   new file: .gitignore
  #   new file: gmres_test.m
  #   new file: jacobi.m
  #   new file: krylov.m
  #   new file: mmread.m
  #   new file: mmwrite.m
  #   new file: mygmres.m
  #   new file: rotmat.m
  #   new file: steepest_descent.m
  #

Writing a good commit message

- Give a general 50 character overview of what you did
- Then give more details
- Which of these is more useful?

Tpetra: Improve build time for a test

@trilinos/tpetra The test was reinstantiating Tpetra classes unnecessarily. It looks like this was fall-out from the effort >= two years ago to let Tpetra turn off GlobalOrdinal = int support. The test got hacked to build, rather than actually being fixed. This commit doesn't fix the hack, but it at least gets rid of the reinstantiations that make the test slow to build, esp. on CUDA.

Piro: Adding back what was deleted

What's our state now?

- [amklinv@klogin2 krylov]$ git status
  # On branch master
  nothing to commit, working directory clean
- [amklinv@klogin2 krylov]$ **git log**
  commit 4a3ccd53172d099443f212f6ce3377b92caf8112
  Author: Alicia Klinvex amklinv@sandia.gov
  Date: Wed Aug 2 09:53:59 2017 -0700

      Added Matlab scripts demonstrating Krylov methods

      These scripts were used to generate images for an intern seminar
      at SNL. They teach concepts like "what is the effect of the choice
      of restart for GMRES" and "how do GMRES, MINRES, and CG perform on
      the same ill-conditioned linear system".

Where are my changes?

- After git commit…
  a) workspace
  b) staging area
  c) local repository

Where are my changes?

- After git commit…
  a) workspace
  b) staging area
  c) local repository

Oops, I accidentally deleted a file! Fix it!

- [amklinv@klogin2 krylov]$ git status
  # On branch master
  # Changes not staged for commit:
  #   (use "git add/rm <file>..." to update what will be committed)
  #   (use "git checkout -- <file>..." to discard changes in working directory)
  #
  #   deleted: krylov.m
  #
  no changes added to commit (use "git add" and/or "git commit -a")

git is helping us again!

- [amklinv@klogin2 krylov]$ **git checkout** krylov.m
- [amklinv@klogin2 krylov]$ git status
  # On branch master
  nothing to commit, working directory clean

Oops, I accidentally broke everything! Fix it!

- [amklinv@klogin2 krylov]$ git status
  # On branch master
  # Changes not staged for commit:
  #   (use "git add <file>..." to update what will be committed)
  #   (use "git checkout -- <file>..." to discard changes in working directory)
  #
  #   modified: jacobi.m
  #
  no changes added to commit (use "git add" and/or "git commit -a")
- [amklinv@klogin2 krylov]$ git checkout jacobi.m
- [amklinv@klogin2 krylov]$ git status
  # On branch master
  nothing to commit, working directory clean

Local Git Review

Local git review

- Which command makes a new repository?
  a) git add
  b) git init

Local git review

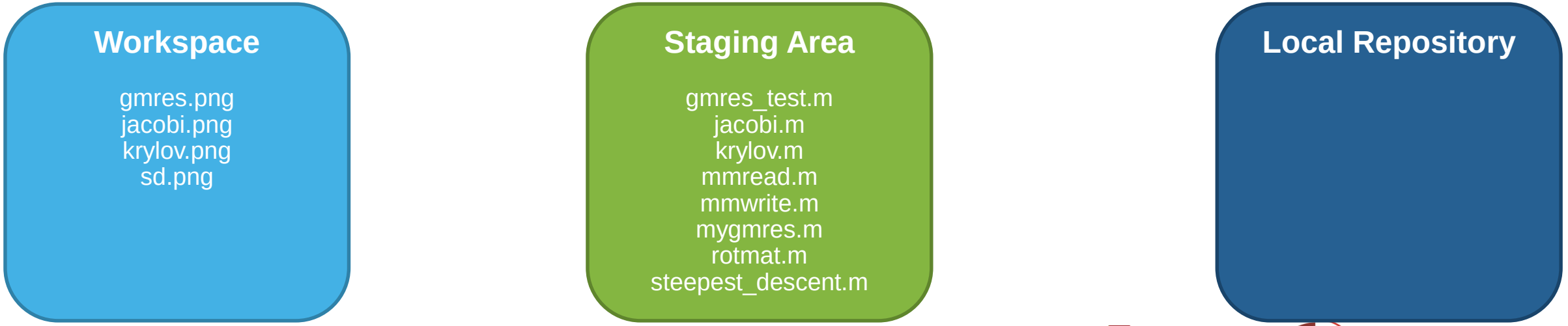
- Which command makes a new repository?
  a) git add
  b) git init

Local git review

- Which command makes a new repository?
  a) git add
  b) git init

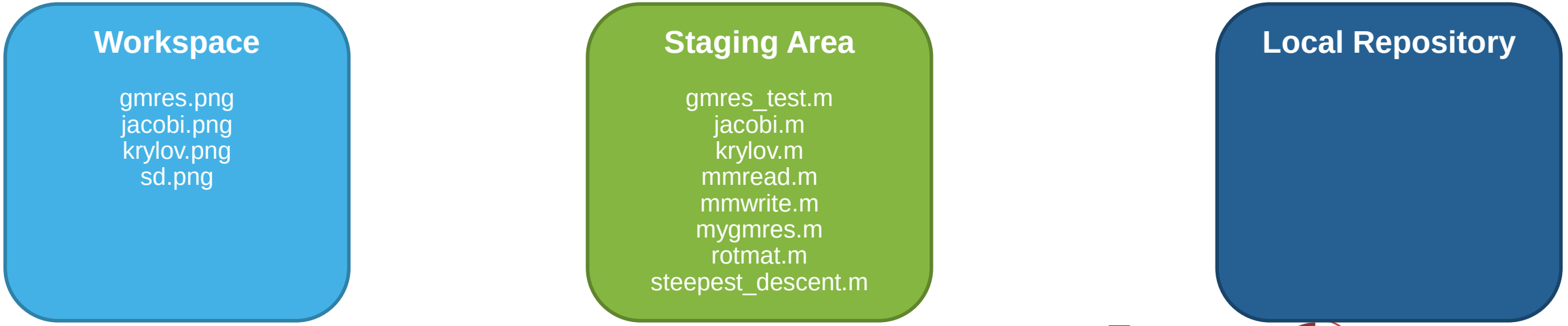
Local git review

- Which command tells us about the state of the workspace and staging area?
  a) git log
  b) git status



Local git review

- Which command tells us about the state of the workspace and staging area?
  a) git log
  b) git status



Local git review

- Which command moves things from the workspace to the staging area?
  a) git add
  b) git commit

Local git review

- Which command moves things from the workspace to the staging area?
  a) git add
  b) git commit

Local git review

- Which command moves things from the workspace to the staging area?
  - **a) git add**
  - b) git commit

Local git review

- Which command moves things from the staging area to the repo?
  a) git add
  b) git commit

Local git review

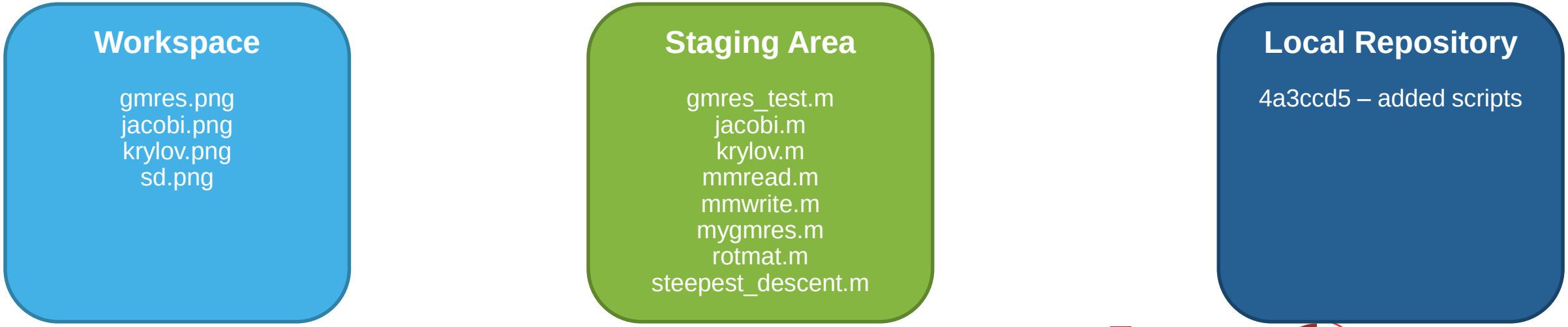
- Which command moves things from the staging area to the repo?
  a) git add
  b) git commit

Local git review

- Which command moves things from the staging area to the repo?
  a) git add
  b) git commit

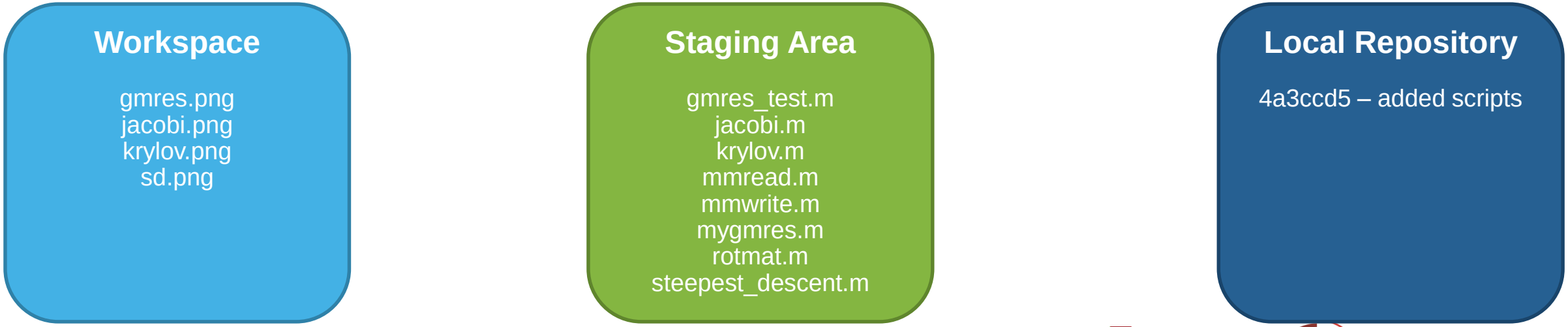
Local git review

- Which command tells us about the state of the local repository?
  a) git log
  b) git status



Local git review

- Which command tells us about the state of the local repository?
  - **a) git log**
  - b) git status



Local git review

- Which command undoes changes to the workspace?
  a) git checkout
  b) git rm

Local git review

- Which command undoes changes to the workspace?
  - **a) git checkout**
  - b) git rm

Local git review

- Workspace – where you do your actual work
- Staging area – where you prepare commits
- Local repository – where the commits are stored

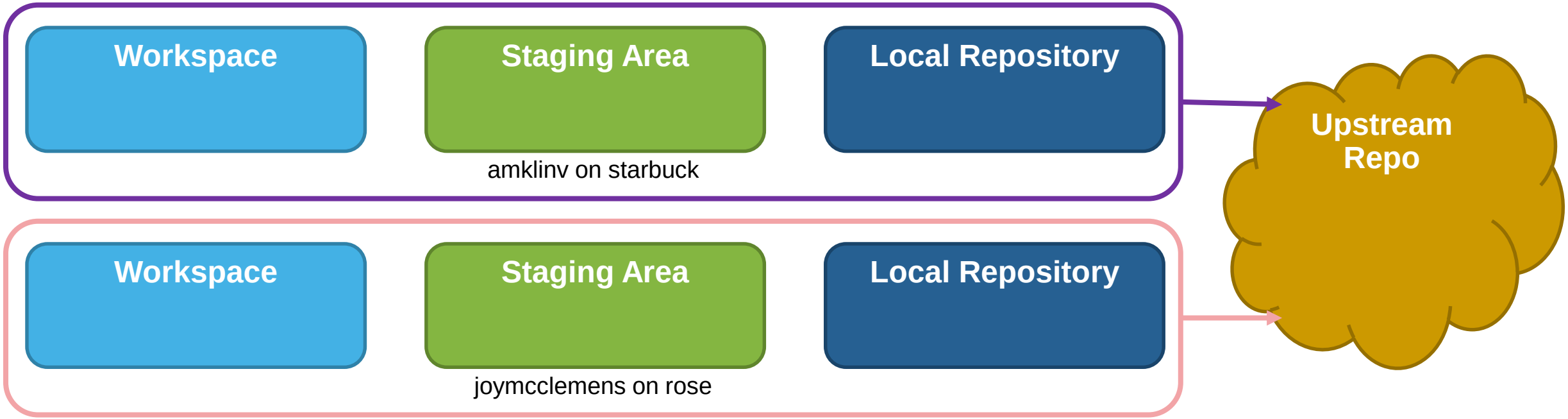
Local git review

- init – creates a new local repository
- status – tells you about any staged/unstaged changes
- add/rm – moves changes from the workspace to the staging area
- commit – moves changes from the staging area to the local repo
- log – tells you about the commits to the local repository
- checkout – undoes changes to the workspace

Which problems have we solved so far?

- Can we undo changes that broke things?
  - YES!
- Can I easily share my updates with collaborators?
  - Not yet…
- Can I easily synchronize my work across multiple machines?
  - Not yet…
- Is my code protected from my computer spontaneously combusting?
  - Not yet…
- Let's talk about distributed git!

Remember, now there are multiple people touching the same distributed repository!



- They may have different workspaces, staging areas, and local repos
- The local repos are not necessarily identical to the upstream one

How do I link my local repository to the one on github?

- Github actually walks you through this
- `git remote add origin https://github.com/amklinv/krylov.git`

How do I link my local repository to the one on github?

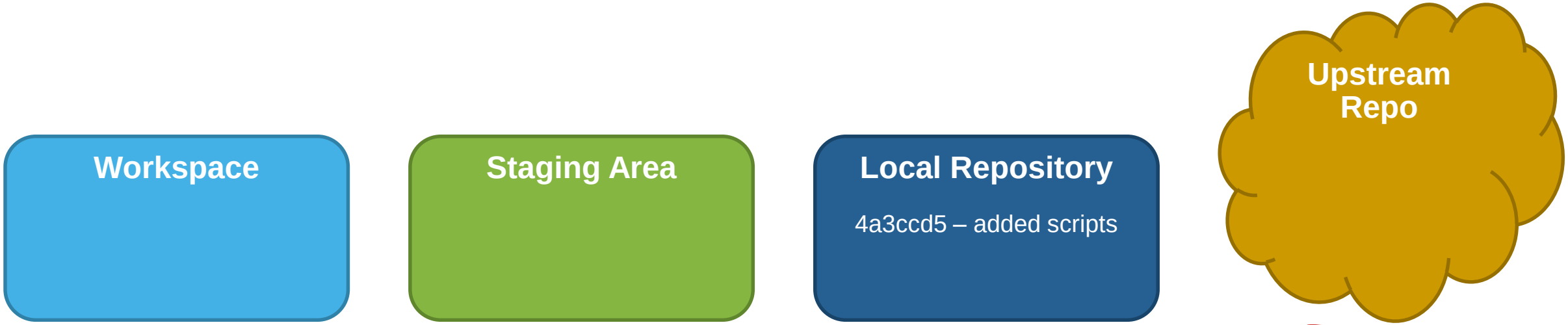
- Github actually walks you through this
- `git remote add origin https://github.com/amklinv/krylov.git`

Argonne NATIONAL LABORATORY

ECP EXASCALE COMPUTING PROJECT

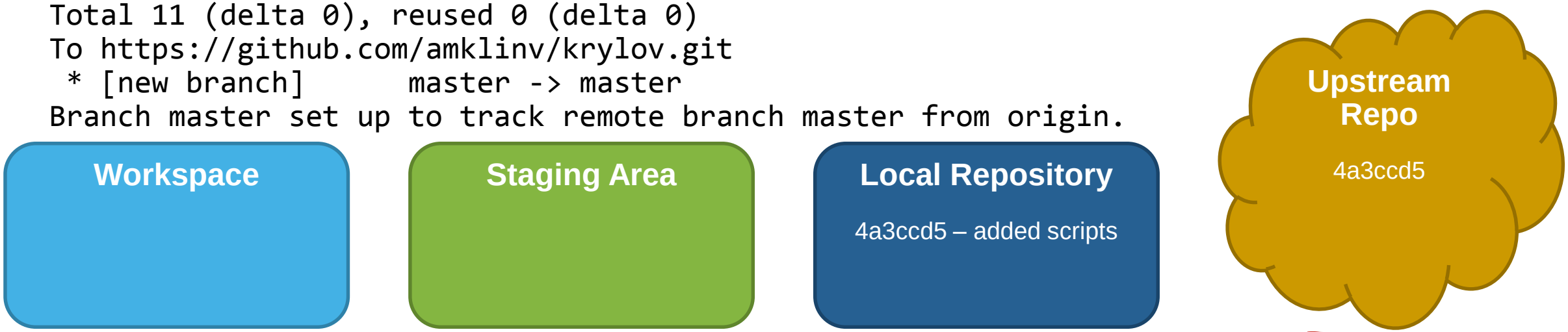
How do I update the upstream (github) repo with my local changes?

- `git push -u origin master`

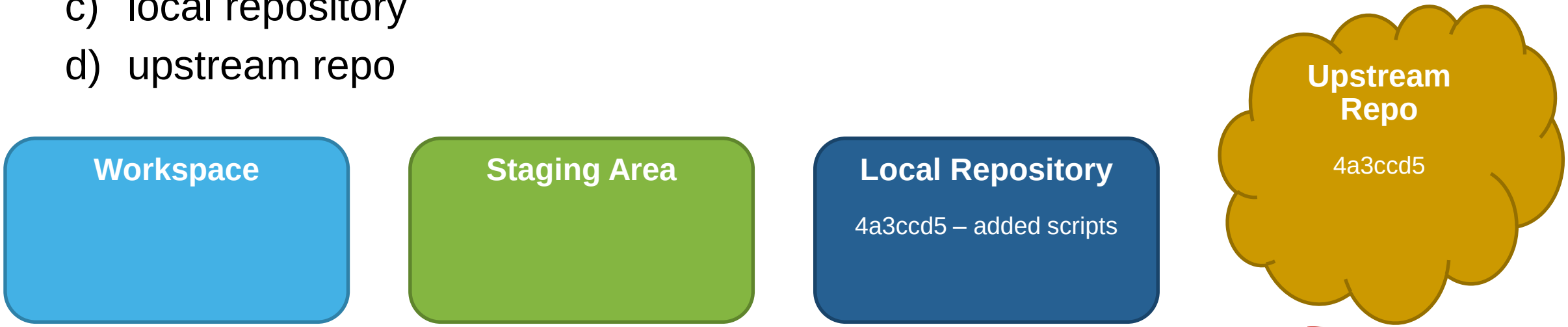
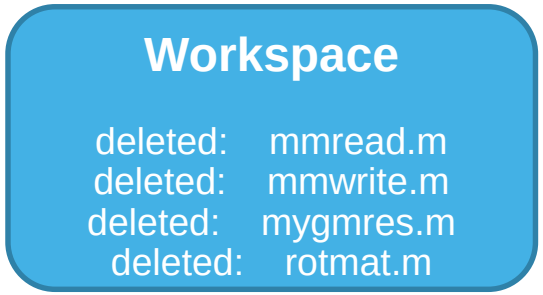
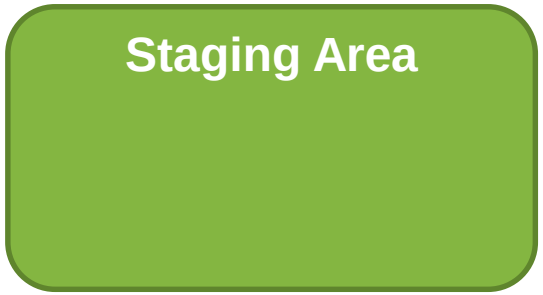
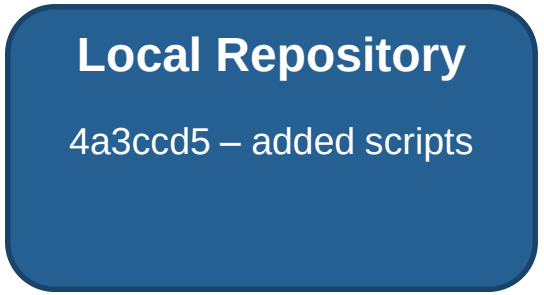
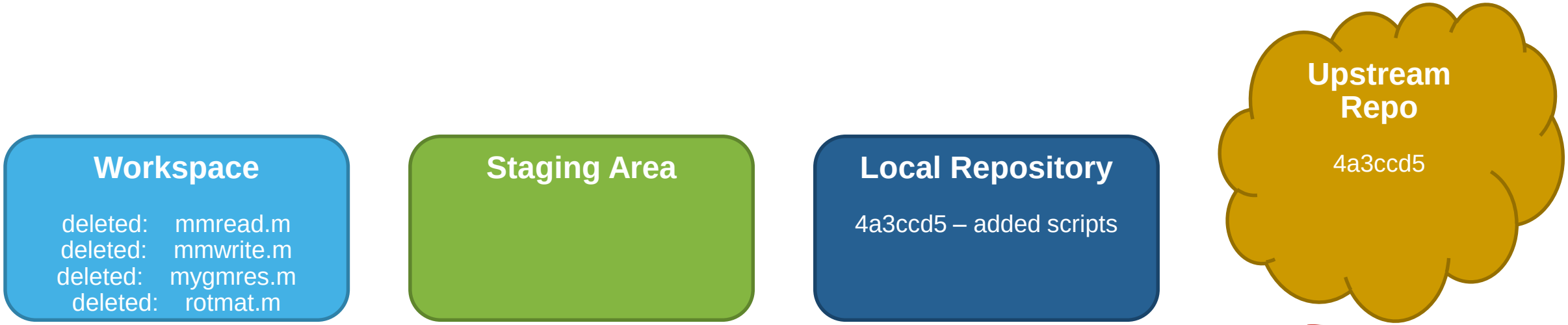
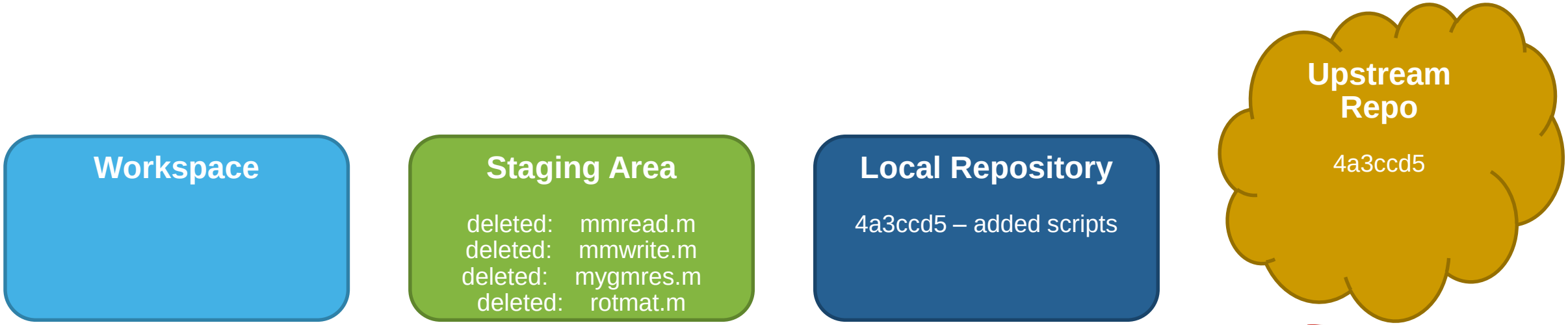
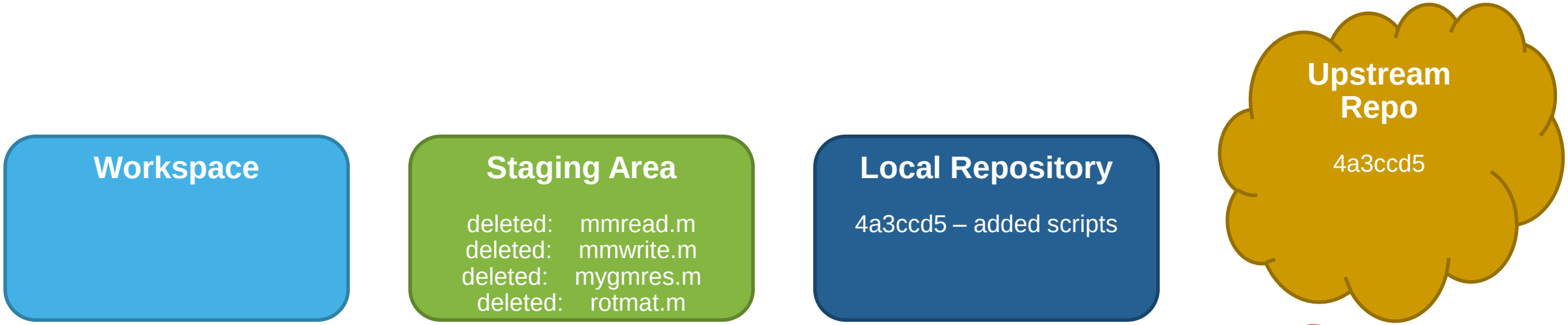
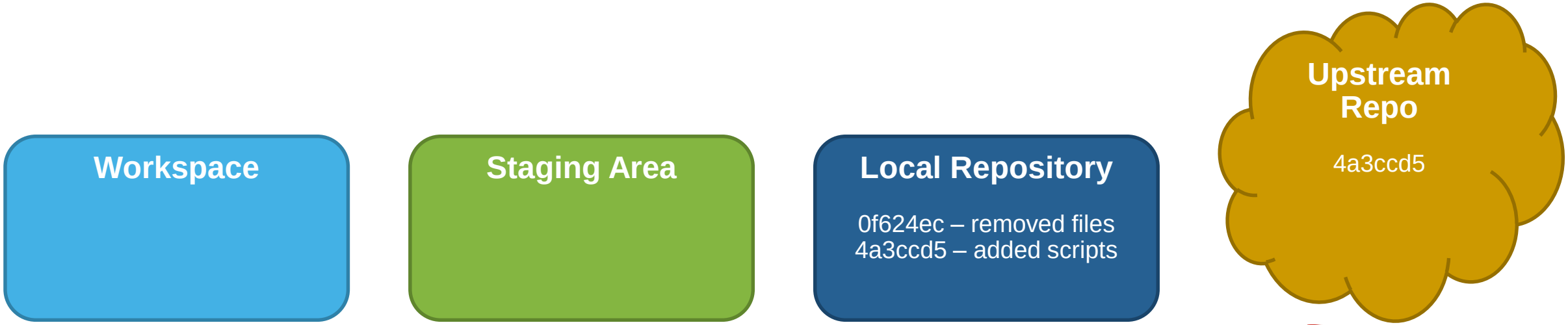
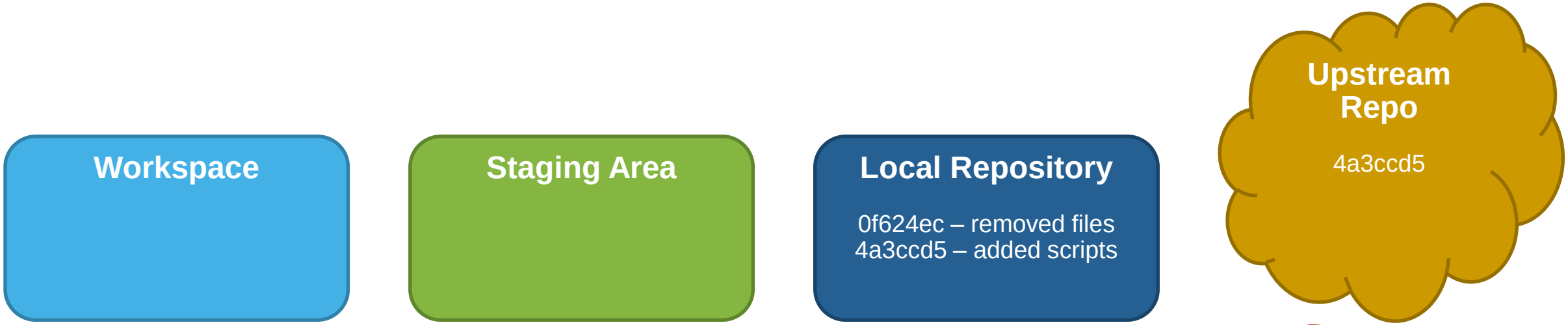
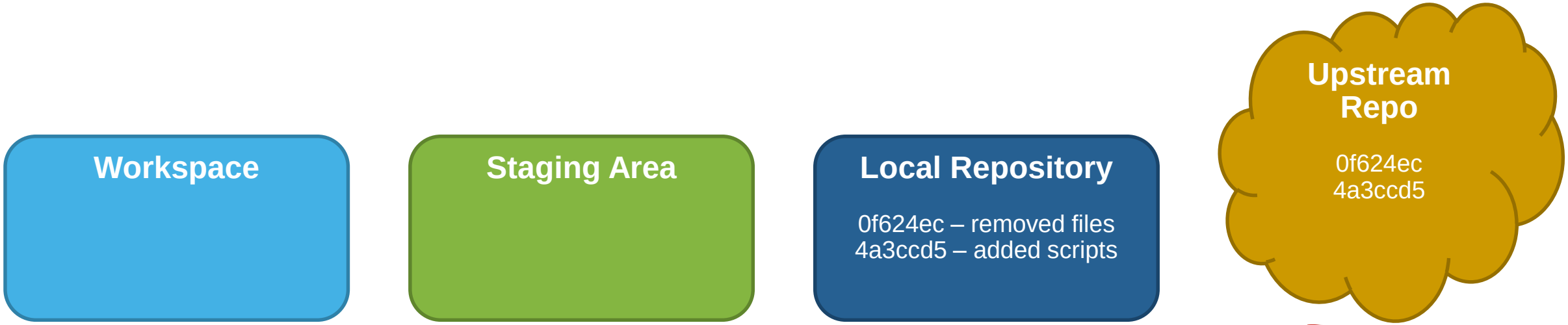


How do I update the upstream (github) repo with my local changes?

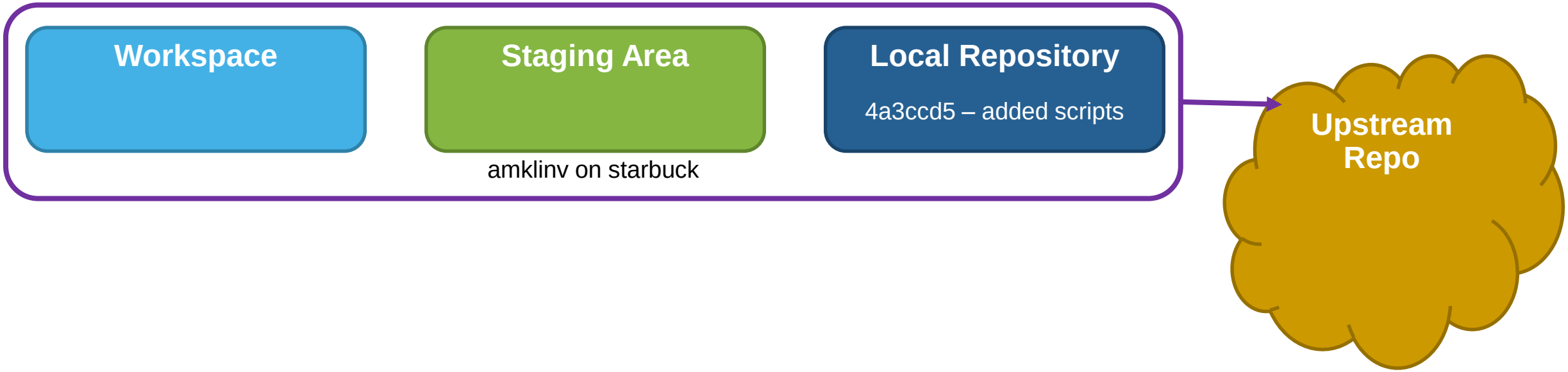
- ```
  [amklinv@klogin2 krylov]$ git push -u origin master
  Username for 'https://github.com': amklinv
  Password for 'https://amklinv@github.com':
  Counting objects: 11, done.
  Delta compression using up to 56 threads.
  Compressing objects: 100% (10/10), done.
  Writing objects: 100% (11/11), 6.84 KiB | 0 bytes/s
  done.
  Total 11 (delta 0), reused 0 (delta 0)
  To https://github.com/amklinv/krylov.git
   * [new branch]      master -> master
  Branch master set up to track remote branch master from origin.
  ```



Argonne NATIONAL LABORATORY

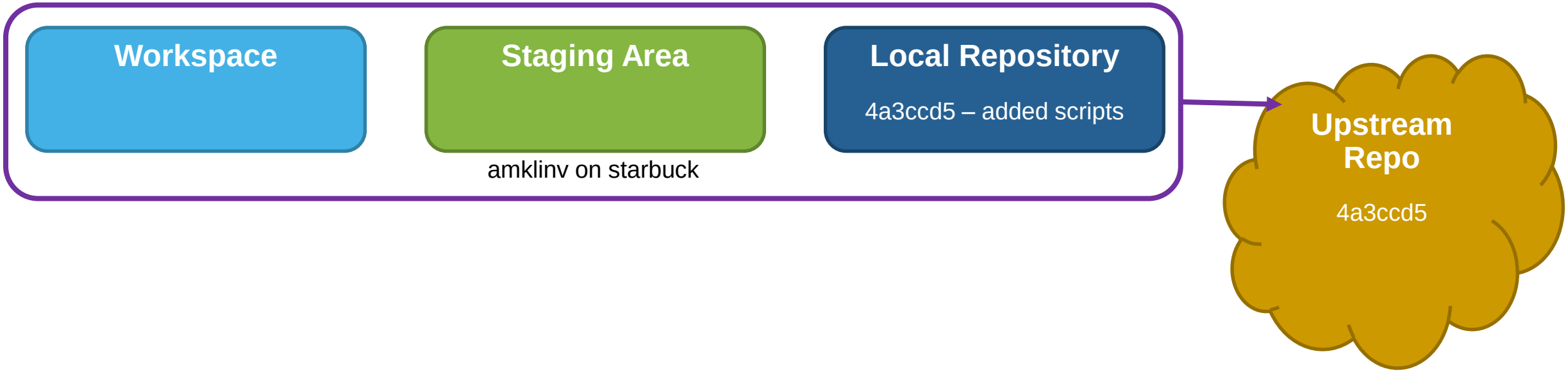
ECP EXASCALE COMPUTING PROJECT
  ```

Oops, I put some files in my repository I didn't mean to track. Now what?

- Remove the files locally
- What changed?
  a) workspace
  b) staging area
  c) local repository
  d) upstream repo



Oops, I put some files in my repository I didn't mean to track. Now what?

- Remove the files locally
- What changed?
  - **a) workspace**
  - b) staging area
  - c) local repository
  - d) upstream repo







Argonne NATIONAL LABORATORY

ECP EXASCALE COMPUTING PROJECT

Oops, I put some files in my repository I didn't mean to track. Now what?

- How do I update my staging area?
  a) git add/rm
  b) git commit
  c) git push



Oops, I put some files in my repository I didn't mean to track. Now what?

- How do I update my staging area?
  - **a) git add/rm**
  - b) git commit
  - c) git push



Oops, I put some files in my repository I didn't mean to track. Now what?

- How do I update my local repository?
  a) git add/rm
  b) git commit
  c) git push



Oops, I put some files in my repository I didn't mean to track. Now what?

- How do I update my local repository?
  a) git add/rm
  b) git commit
  c) git push



Oops, I put some files in my repository I didn't mean to track. Now what?

- How do I update the upstream repository?
  a) git add/rm
  b) git commit
  c) git push



Oops, I put some files in my repository I didn't mean to track. Now what?

- How do I update the upstream repository?
  a) git add/rm
  b) git commit
  c) git push



A funny thing happened when I tried to push…

- ```
  [amklinv@klogin2 krylov]$ git push
  Username for 'https://github.com': amklinv
  Password for 'https://amklinv@github.com':
  To https://github.com/amklinv/krylov.git
   ! [rejected]        master -> master (fetch first)
  error: failed to push some refs to 'https://github.com/amklinv/krylov.git‘
  hint: Updates were rejected because the remote contains work that you do
  hint: not have locally. This is usually caused by another repository pushing
  hint: to the same ref. You may want to first merge the remote changes (e.g.,
  hint: 'git pull') before pushing again.
  hint: See the 'Note about fast-forwards' in 'git push --help' for details.
  ```
  ```

Why did this happen?



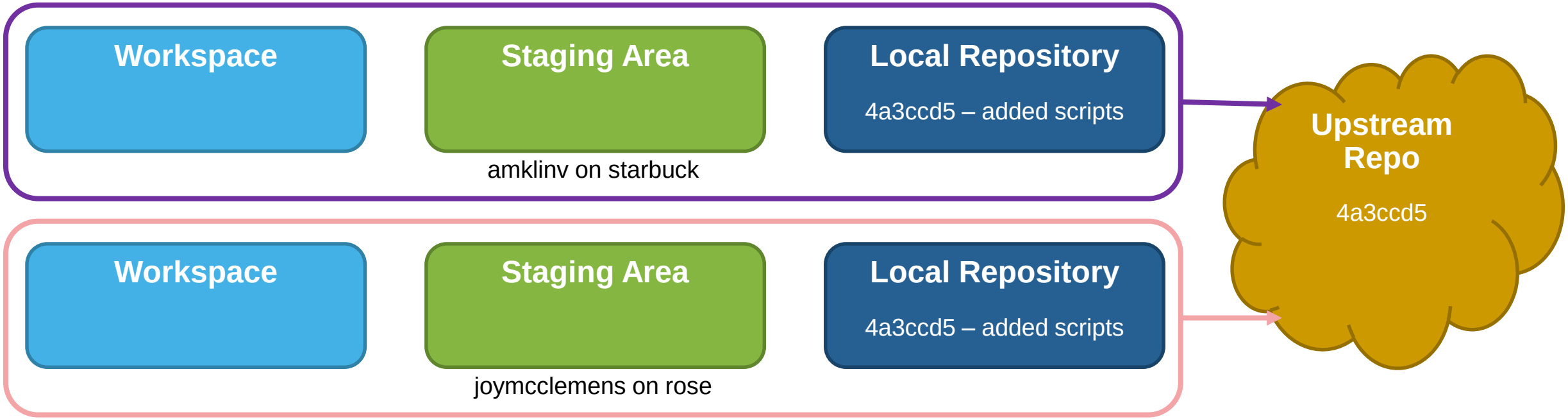
- I pushed my work to the upstream repository

Why did this happen?



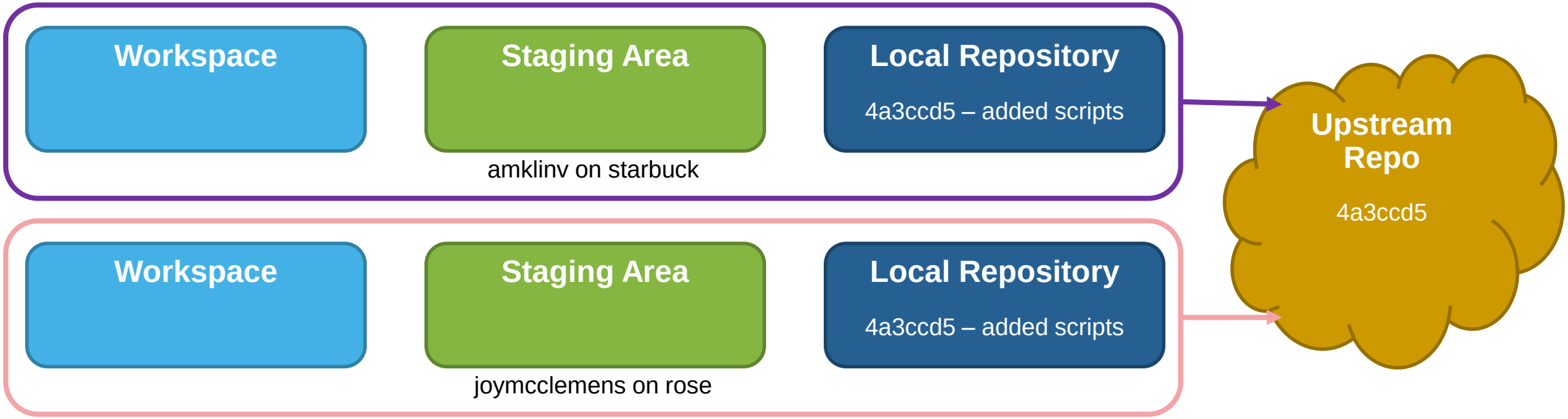
- I pushed my work to the upstream repository

Why did this happen?



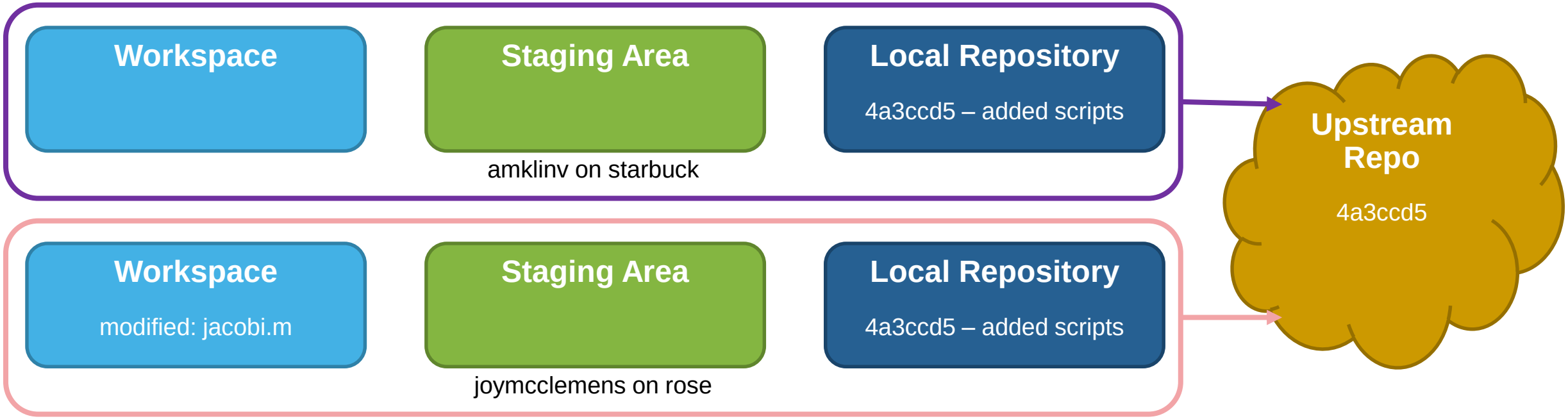
- joymcclemens gets a local copy by using
  `git clone https://github.com/amklinv/krylov.git`

Why did this happen?



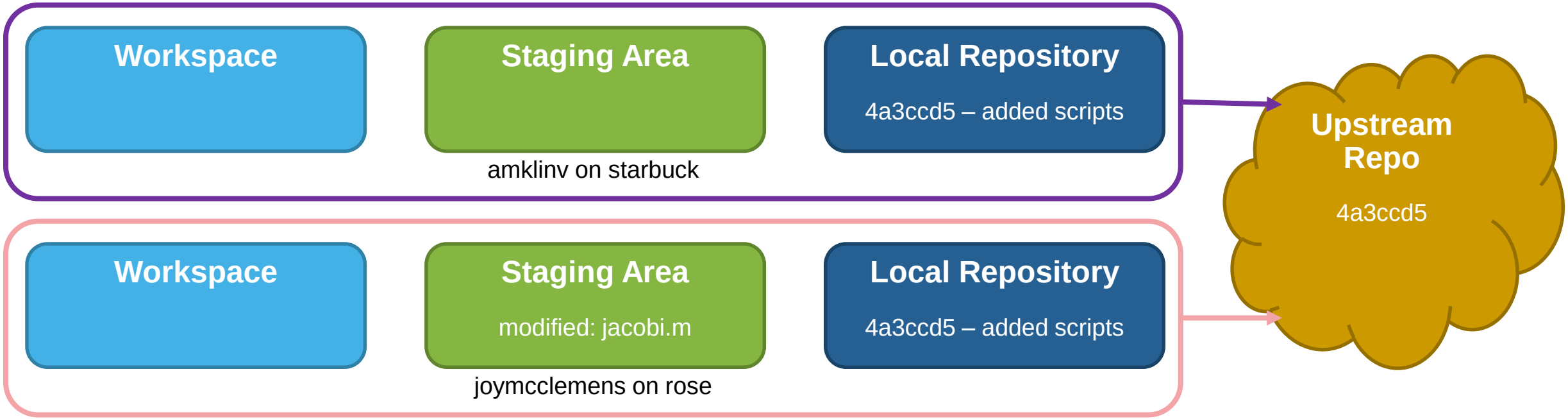
- joymcclemens decides one of my plots needs a legend

Why did this happen?



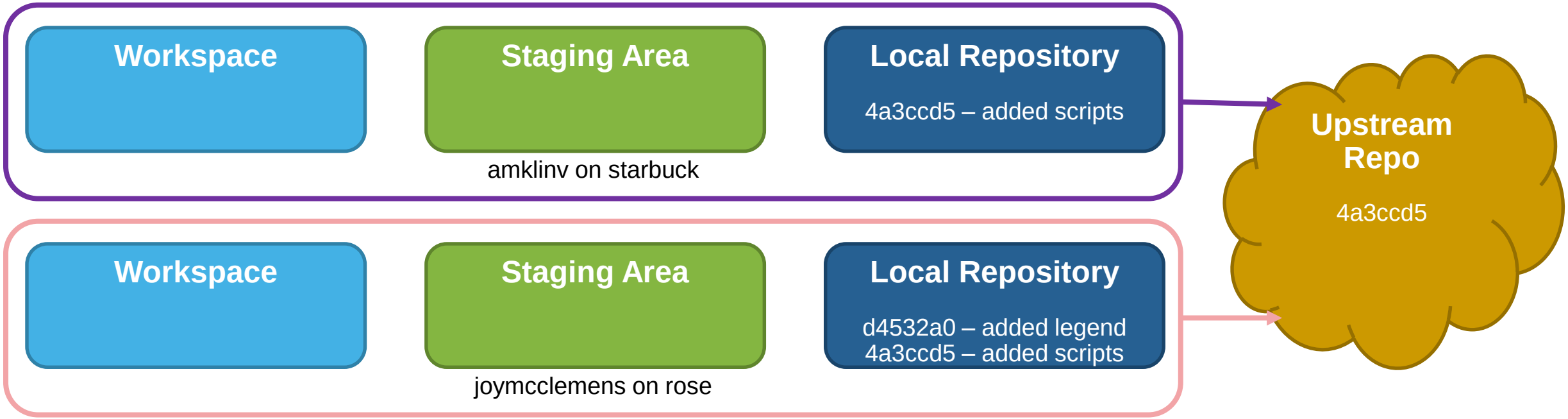
- She adds a legend to my Matlab script

Why did this happen?



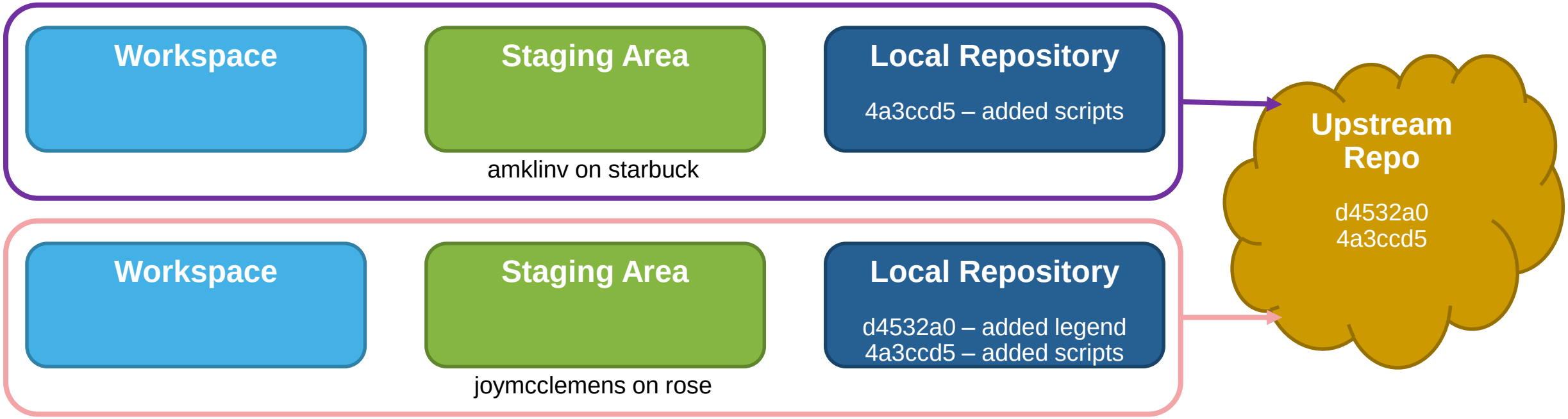
- She adds it to the staging area

Why did this happen?



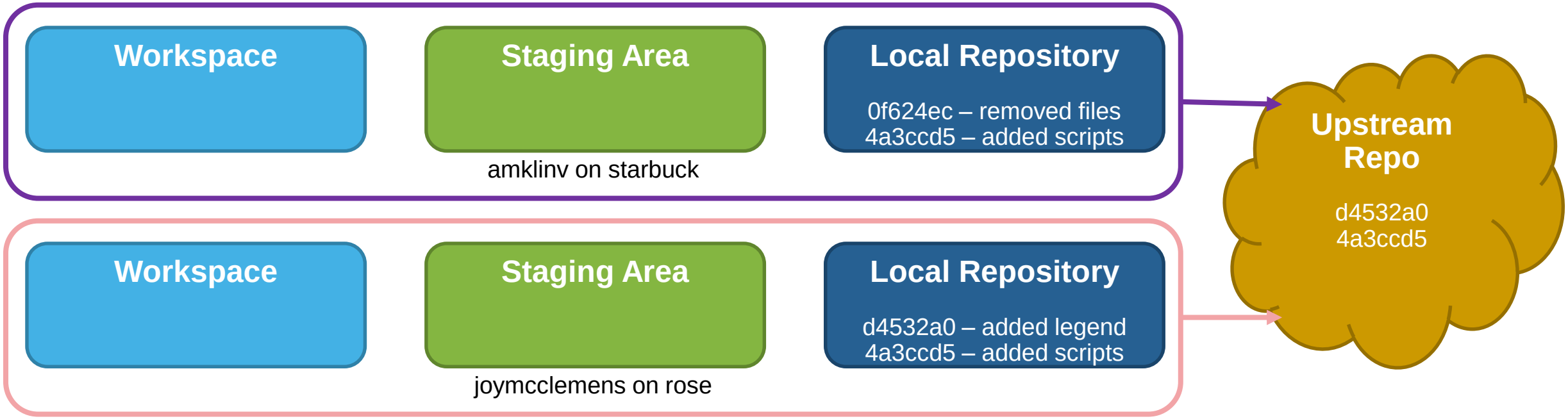
- She commits it to her local repository

Why did this happen?



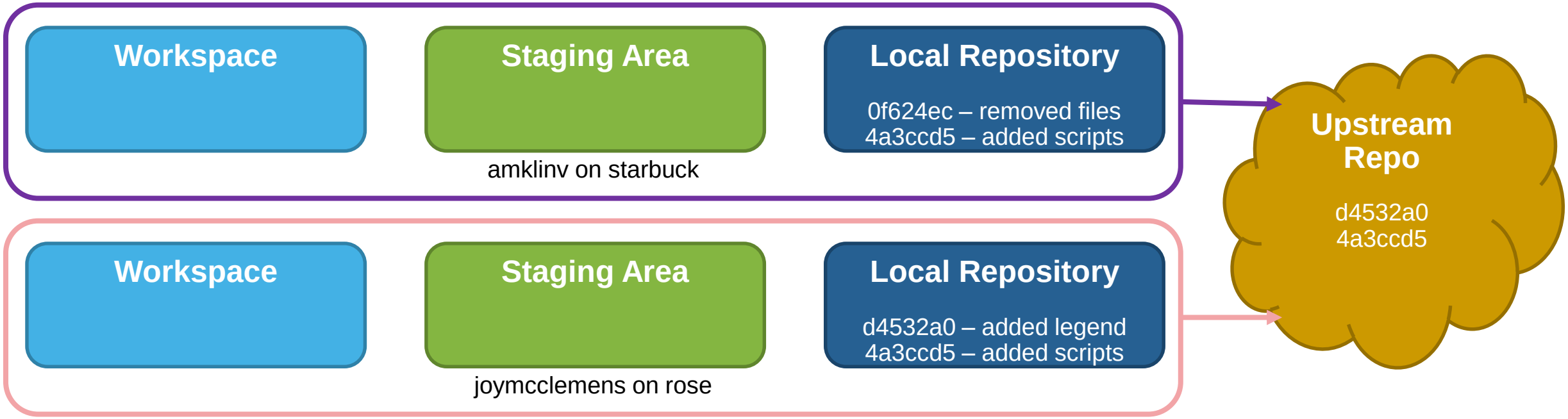
- She pushes it to the upstream repository on github

Why did this happen?



- Meanwhile, I modified my local repository

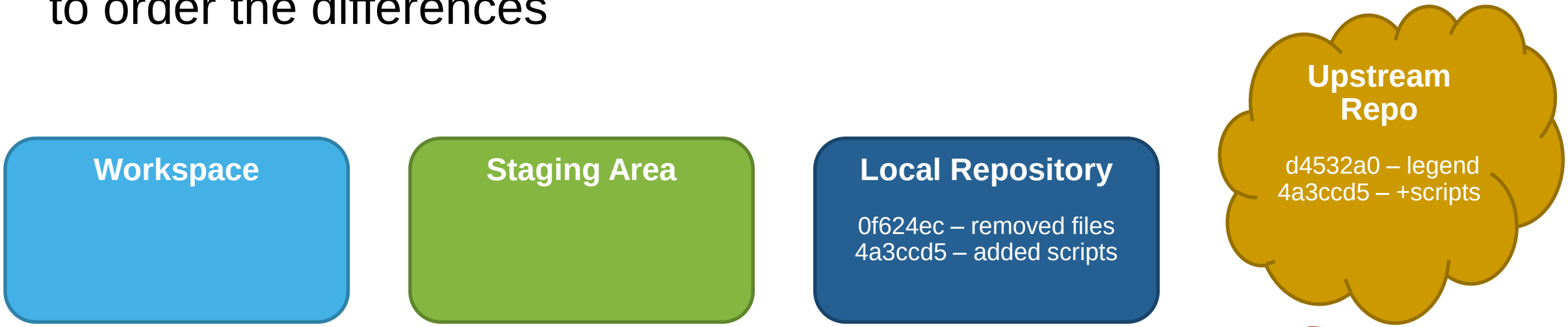
Why did this happen?



- Should “removed files” come before or after “added legend”?

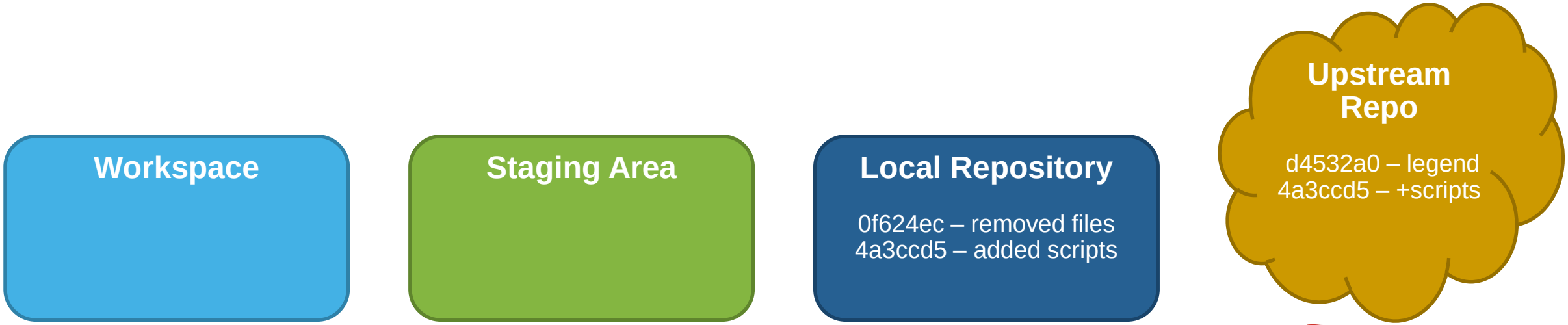
Why did this happen?

- The upstream repository has commits that the local repository doesn't
- We need to update the local repository first
- Git is trying to merge two linear histories, but it needs us to tell it how to order the differences



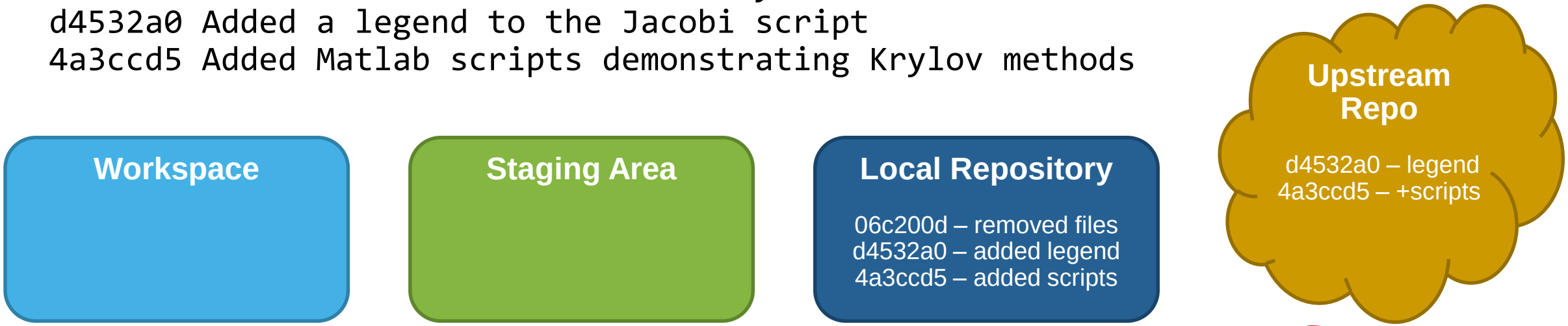
How do we update the local repository?

- `git pull --rebase`
- The rebase option tells git to stick our changes on top of the upstream repo



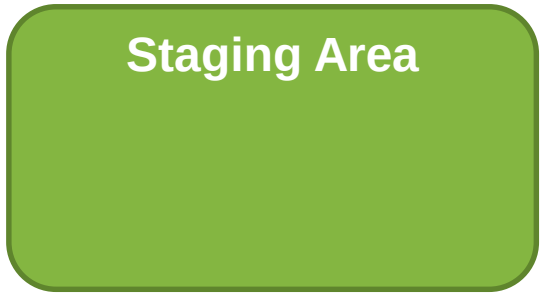
How do we update the local repository?

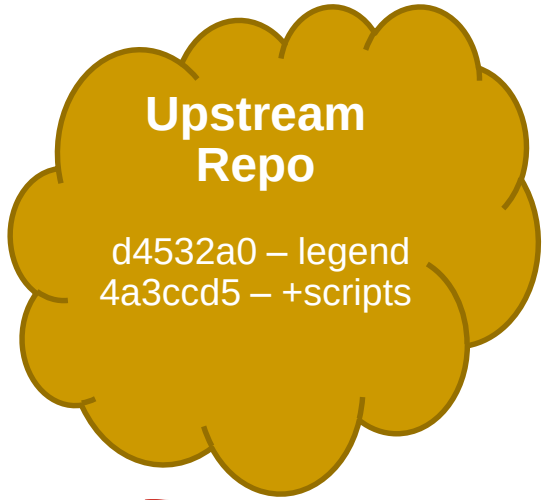
- **`git pull --rebase`**
- The rebase option tells git to stick our changes on top of the upstream repo
-
```
[amklinv@klogin2 krylov]$ git log --pretty=oneline
06c200d Removed files unrelated to krylov methods
d4532a0 Added a legend to the Jacobi script
4a3ccd5 Added Matlab scripts demonstrating Krylov methods
```



How do we update the upstream repository?

- ```
  [amklinv@klogin2 krylov]$ git push
  Username for 'https://github.com': amklinv
  Password for 'https://amklinv@github.com':
  Counting objects: 3, done.
  Delta compression using up to 56 threads.
  Compressing objects: 100% (2/2), done.
  Writing objects: 100% (2/2), 257 bytes | 0 bytes/s, done.
  Total 2 (delta 1), reused 0 (delta 0)
  remote: Resolving deltas: 100% (1/1), completed with 1 local object.
  To https://github.com/amklinv/krylov.git
     d4532a0..06c200d  master -> master
  ```





Argonne NATIONAL LABORATORY

ECP EXASCALE COMPUTING PROJECT
  ```

How do we update the upstream repository?

- ```
  [amklinv@klogin2 krylov]$ git push
  Username for 'https://github.com': amklinv
  Password for 'https://amklinv@github.com':
  Counting objects: 3, done.
  Delta compression using up to 56 threads.
  Compressing objects: 100% (2/2), done.
  Writing objects: 100% (2/2), 257 bytes | 0 bytes/s, done.
  Total 2 (delta 1), reused 0 (delta 0)
  remote: Resolving deltas: 100% (1/1), completed with 1 local object.
  To https://github.com/amklinv/krylov.git
     d4532a0..06c200d  master -> master
  ```


  ```

An important note about “removing” stuff from a repo

- We didn’t really “remove” anything from the repo.
- Those files are still part of the repo’s history, even though they’re not in the current snapshot
- Git purposefully makes it hard to ever permanently erase anything, though it is possible

Review

- Workspace – where you do your actual work
- Staging area – where you prepare commits
- Local repository – where the commits are stored on your machine
- Upstream repository – the distributed location where commits are stored (sometimes github)

Review

- init – creates a new local repository
- status – tells you about any staged/unstaged changes
- add – moves changes from the workspace to the staging area
- commit – moves changes from the staging area to the local repo
- log – tells you about the commits to the local repository
- checkout – undoes changes to the workspace
- push – moves changes from the local repo to the upstream repo
- pull – moves changes from the upstream repo to the local repo

Git tutorials

- https://try.github.io
- https://ndpsoftware.com/git-cheatsheet.html
- https://www.atlassian.com/git/tutorials
- https://git-scm.com/book
- https://swcarpentry.github.io/git-novice

Thank you for your attention!