Moffitt Bio-Data Club Inaugural Meeting (October 2018)

An introduction to version control with git

Paul Stewart

2018-10-19

Club page: https://pstew.github.io/biodataclub/

Why should I use version control?

19 Answers

active oldest votes

247

Have you ever:

- Made a change to code, realised it was a mistake and wanted to revert back?
- Lost code or had a backup that was too old?
- Had to maintain multiple versions of a product?
- Wanted to see the difference between two (or more) versions of your code?
- Wanted to prove that a particular change broke or fixed a piece of code?
- Wanted to review the history of some code?
- Wanted to submit a change to someone else's code?
- Wanted to share your code, or let other people work on your code?
- Wanted to see how much work is being done, and where, when and by whom?
- Wanted to experiment with a new feature without interfering with working code?

In these cases, and no doubt others, a version control system should make your life easier.

To misquote a friend: A civilised tool for a civilised age.

share edit flag

edited Nov 6 '13 at 0:52

answered Sep 11 '09 at 0:42

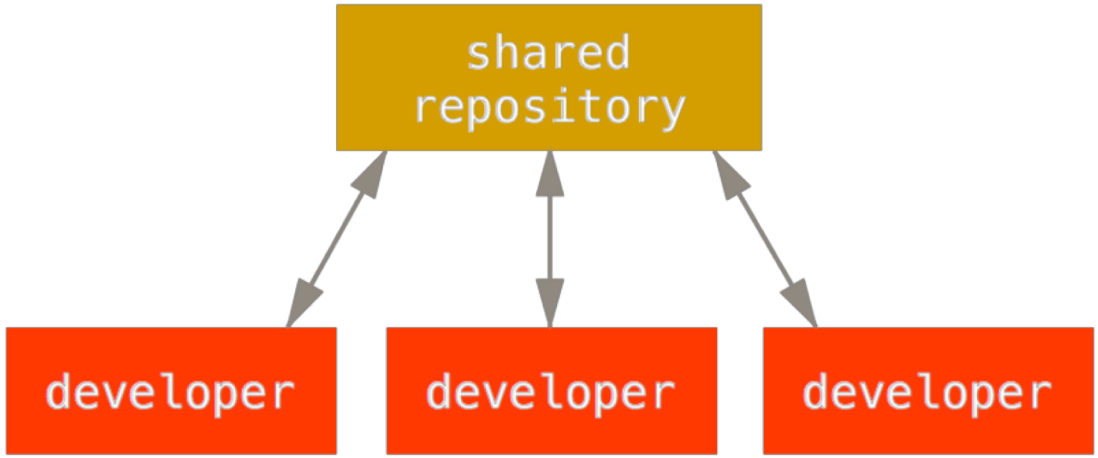
si618

14.2k • 12 • 59 • 79

https://stackoverflow.com/questions/1408450/why-should-i-use-version-control

What is git?

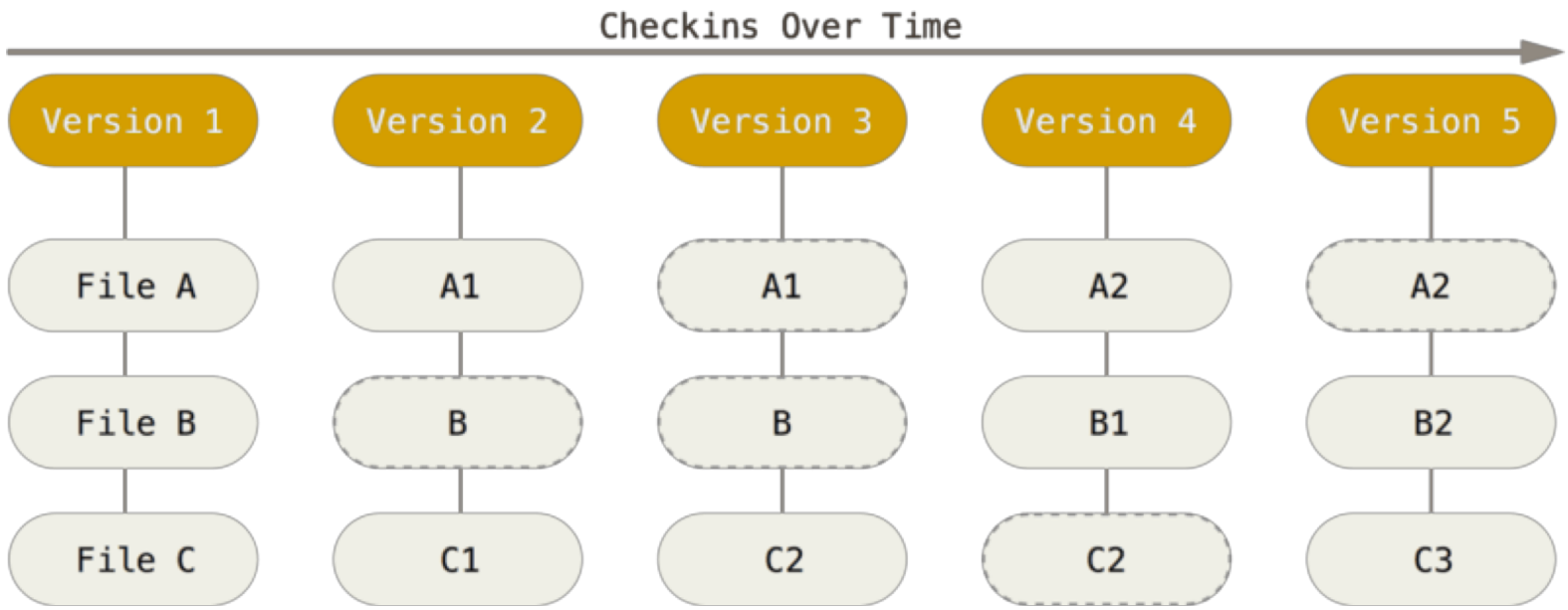
- A system that records changes to a set of files, also known as **version control** (git terminology will appear in **bold** throughout)
- Allows for collaborative development
- Allows you to know who made what changes when
- Allows you to revert any changes and go back to a previous state
- git is a distributed version control system, so each person has a complete copy of the repository on their **local** machine.



https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows

How is git used?

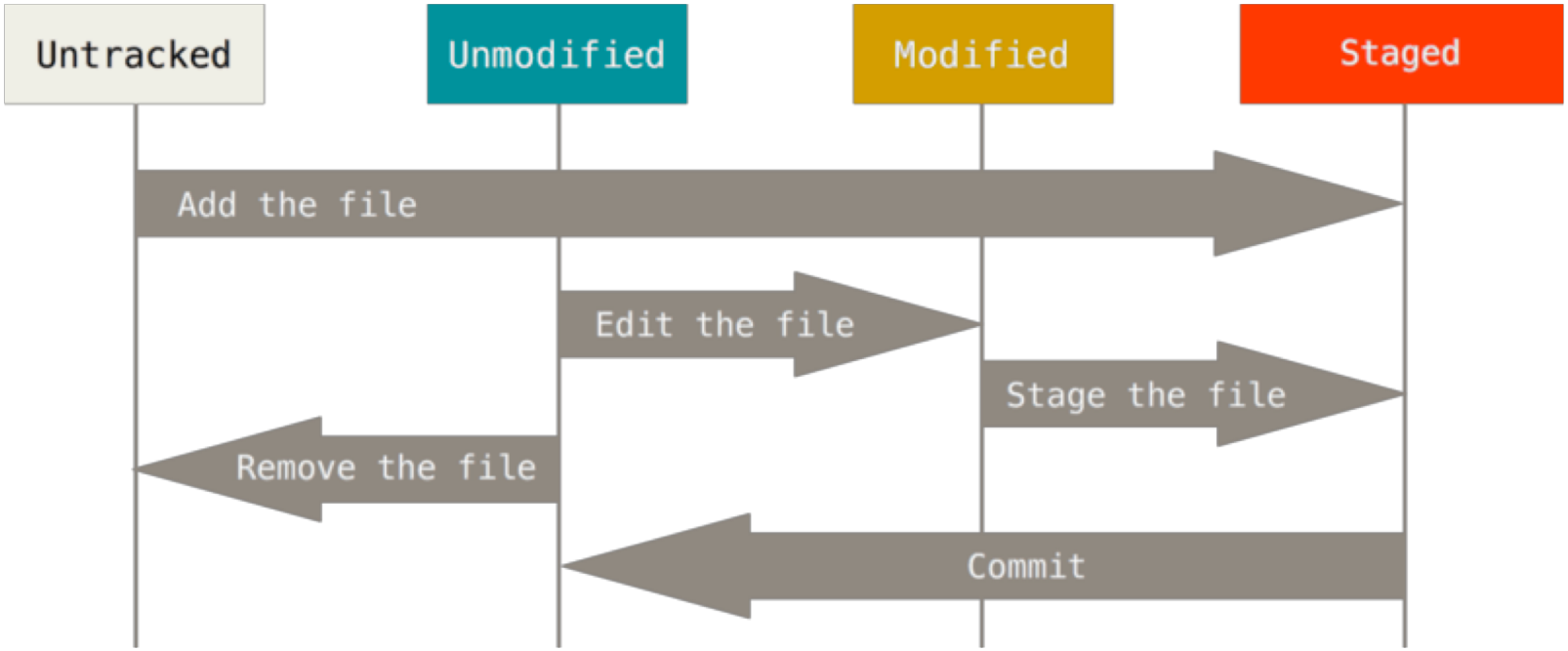
- The purpose of git is to track snapshots of files as they change over time.
- git stores information in a **repository**.
- A **commit** is the act of creating a snapshot of files.



https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

The lifecycle of files in a git repository

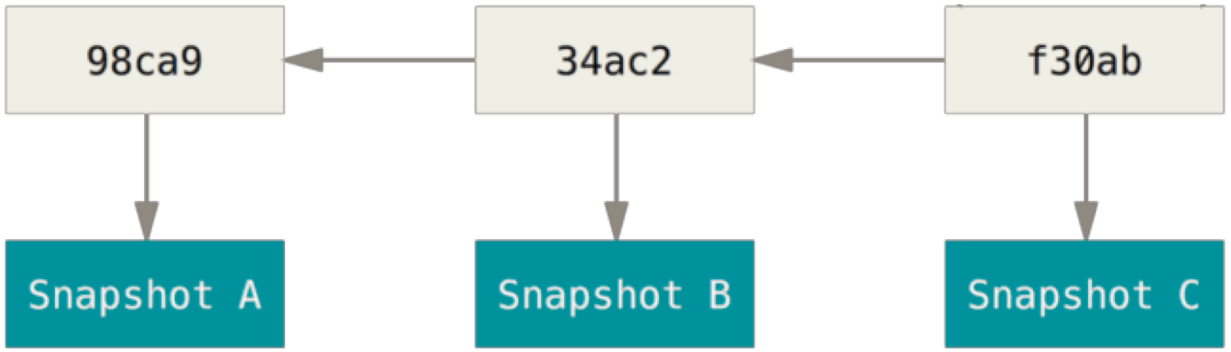
- Each file in your working directory can be in one of two states: tracked or untracked.
- Tracked files are files that were in the last snapshot; they can be unmodified, modified, or staged.
- Tracked files are files that git knows about.



https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

Commits and their parents

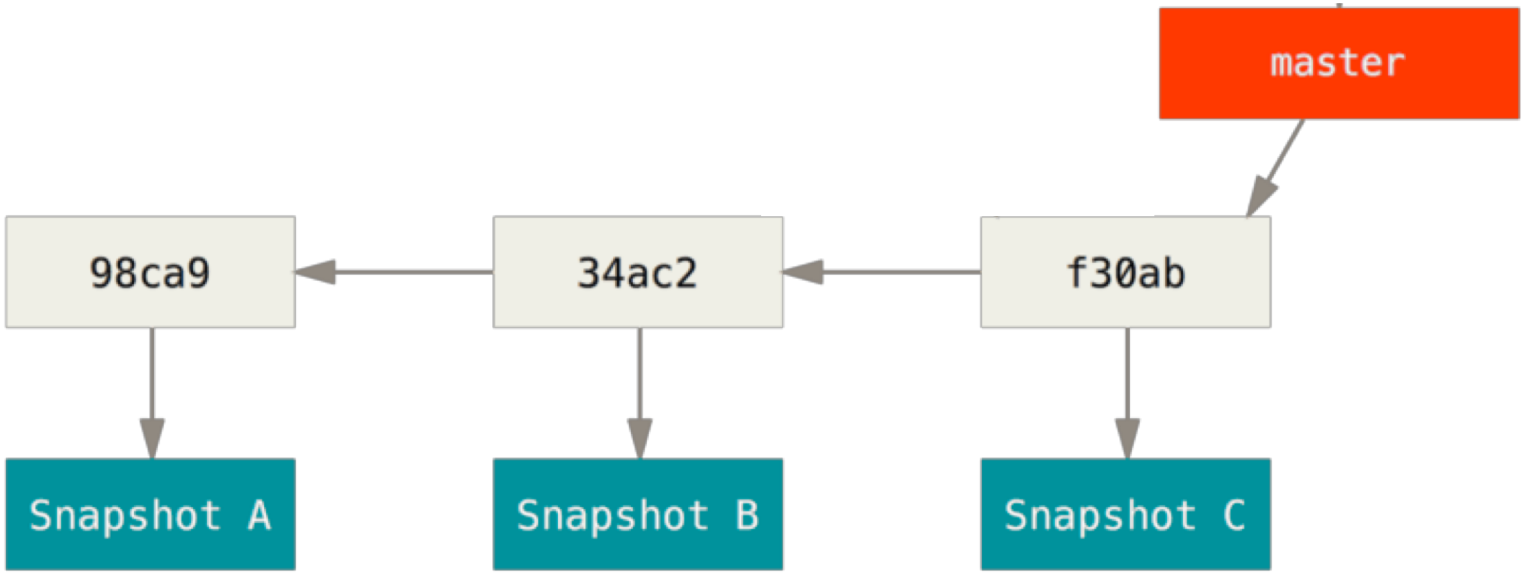
- Each commit has a unique checksum pointer or “address”.
- Commits also track author’s name, author’s email, commit message, and pointers to parents.



https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Commits and branches

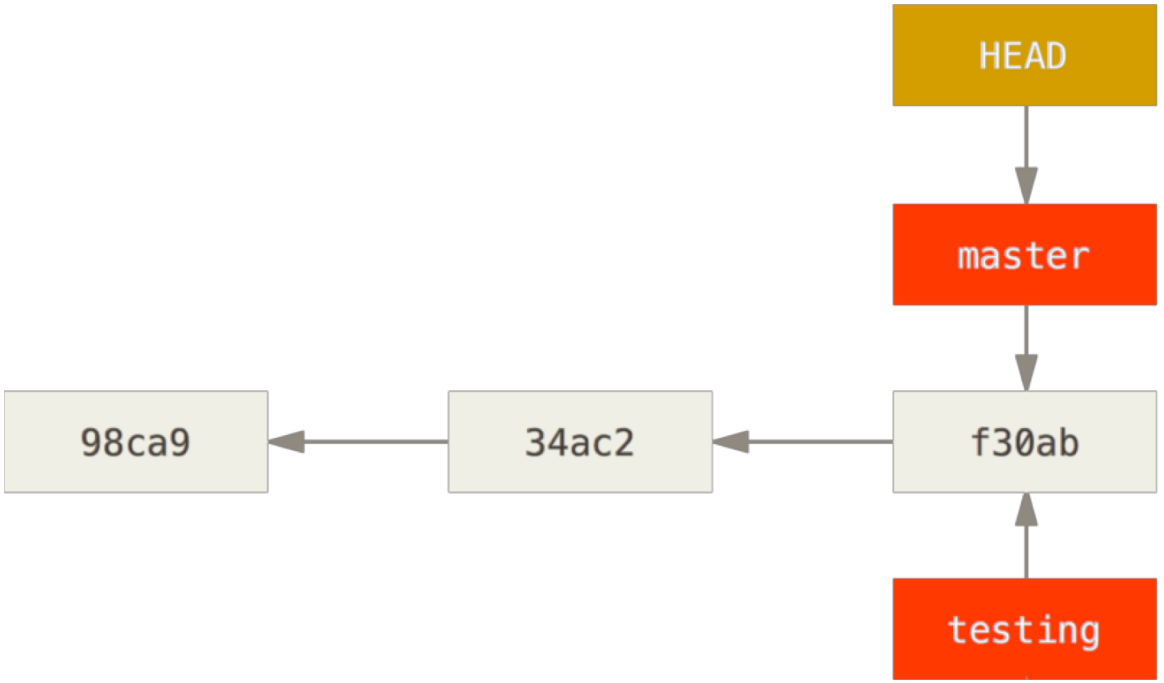
- A **branch** is a pointer to a commit.
- The default branch in git is called **master**, referred to as the “master branch”.



https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Adding a branch

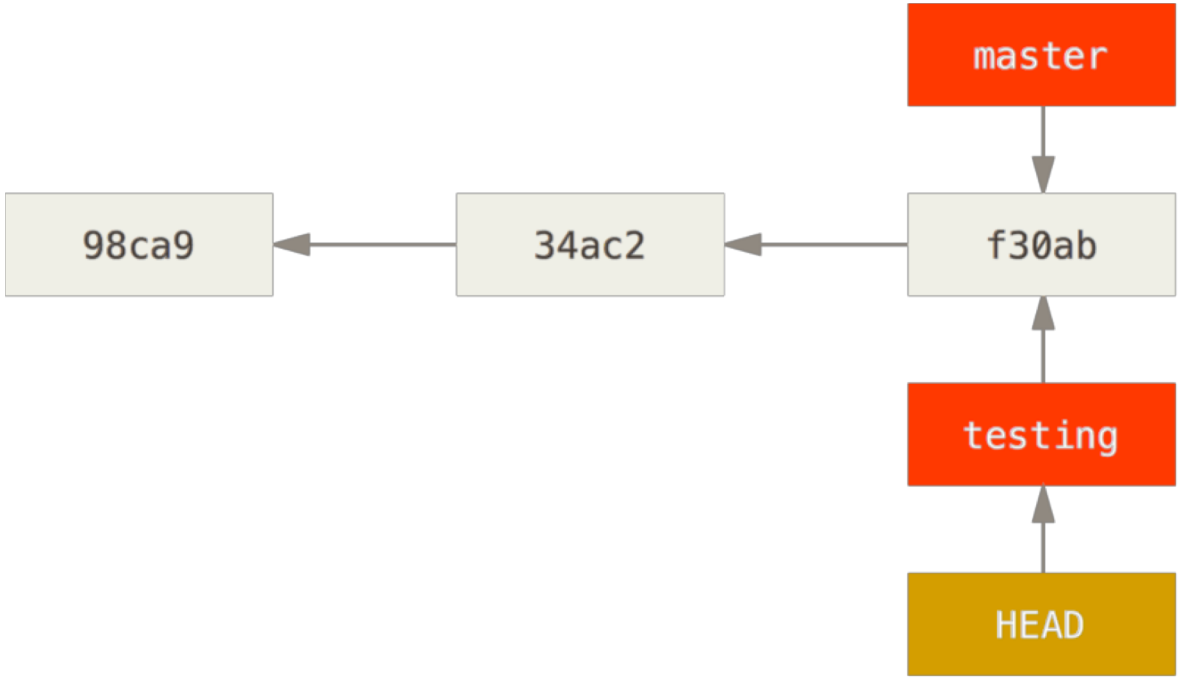
- Using a new branch allows you to run some test analyses or try out a new feature that you might not want to keep later.
- Branches allows collaborators to work on different pieces of the same code simultaneously.
- Pretend we created a new branch called “testing”.
- We now need to distinguish what branch we’re currently working on.
- git keeps track of the branch you are working on by a pointer called **HEAD.**



https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Switching branches

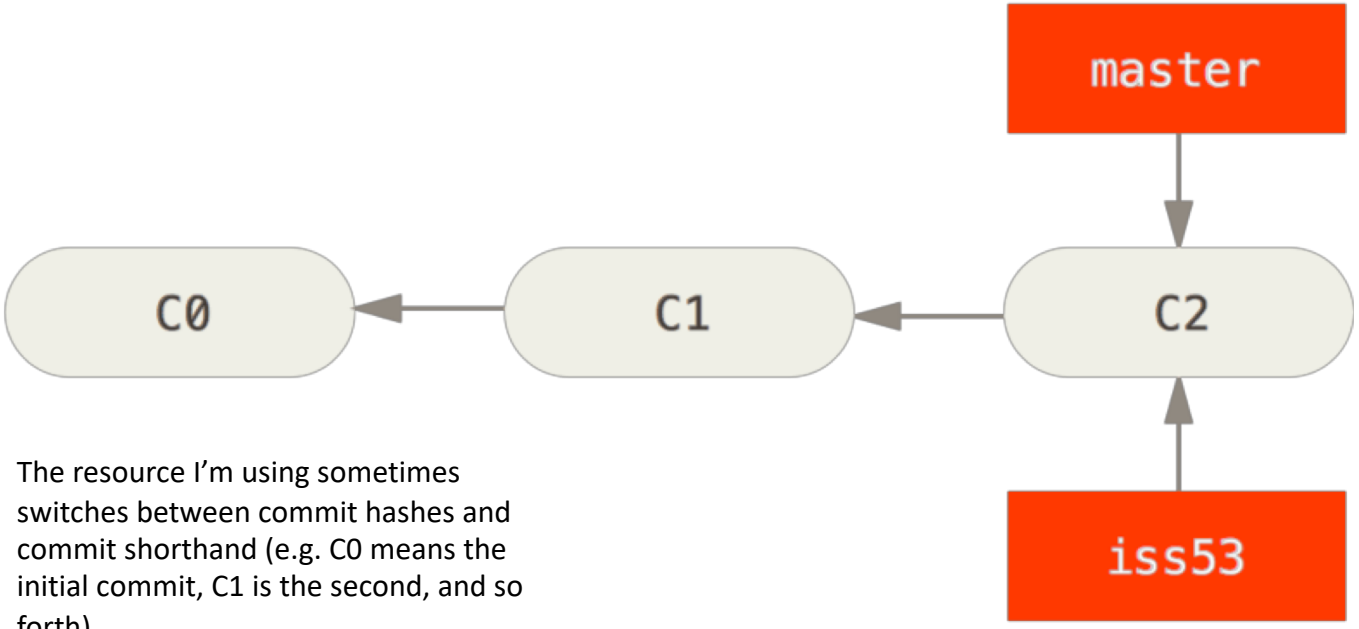
- Switching a branch is referred to as a **check out .**
- HEAD is now on “testing”, which means “testing” is checked out.
- Right now master and testing are identical, but any changes made to “testing” will not be reflected in “master”.



https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell

Make changes to a branch

- Pretend we've made some commits under "master" and have just created a branch "iss53".

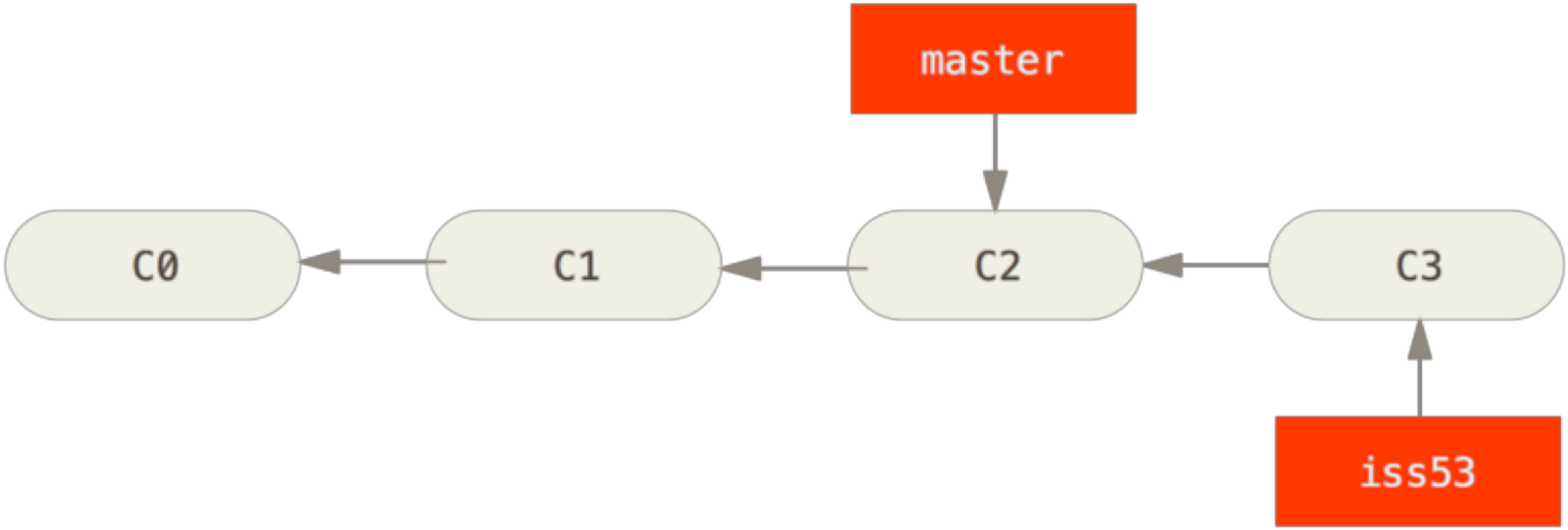


The resource I'm using sometimes switches between commit hashes and commit shorthand (e.g. C0 means the initial commit, C1 is the second, and so forth)

https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Make changes to a branch

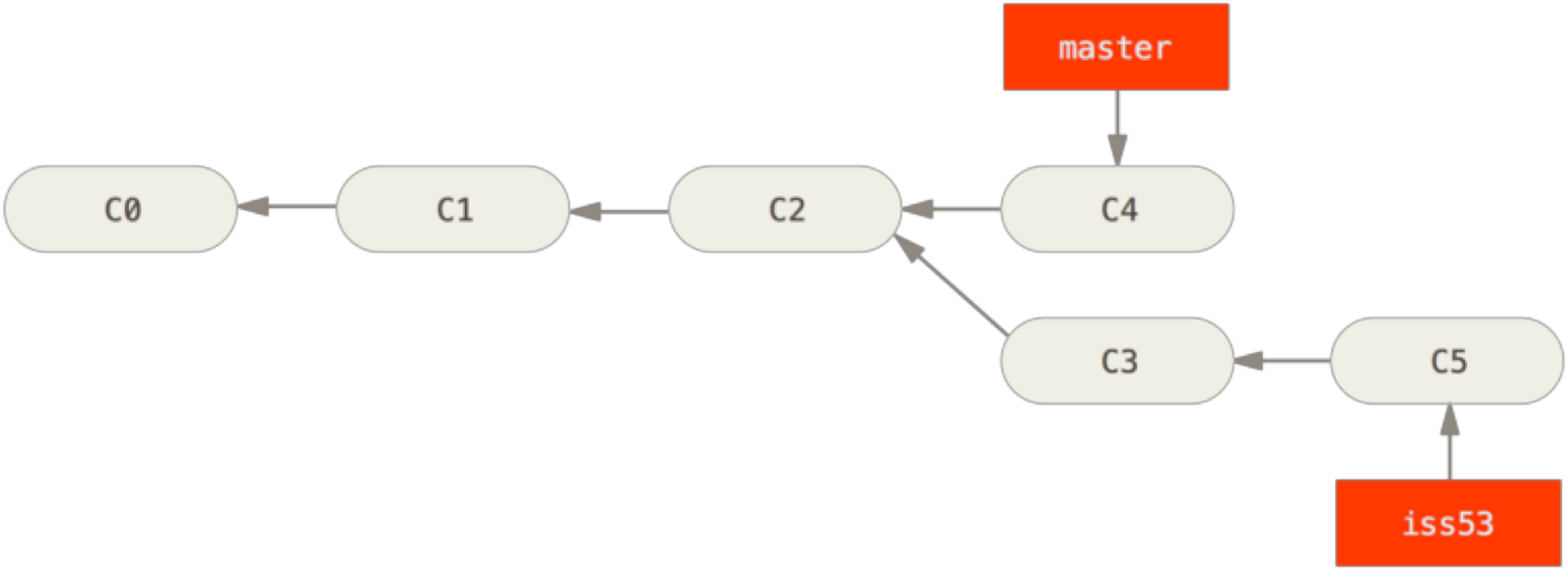
- Commits to “iss53” are not reflected in “master”.



https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Make changes to a branch

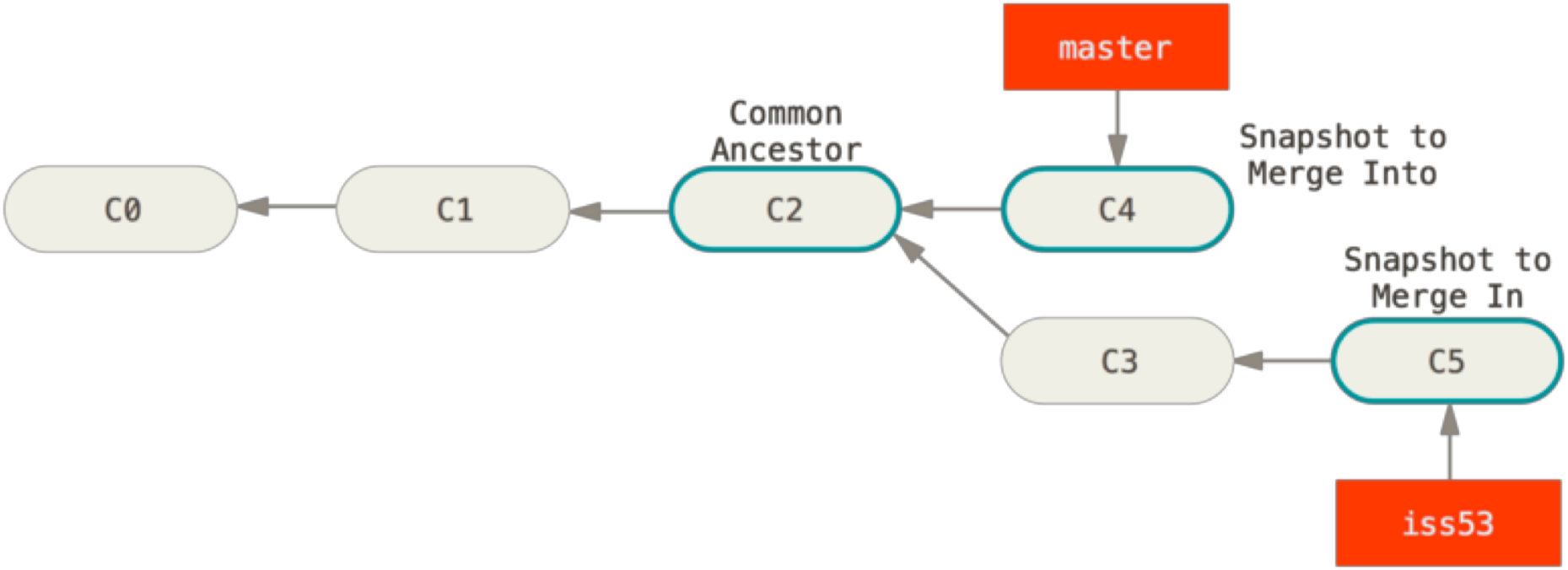
- Commits to “master” are not reflected in “iss53”.



https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Merge changes

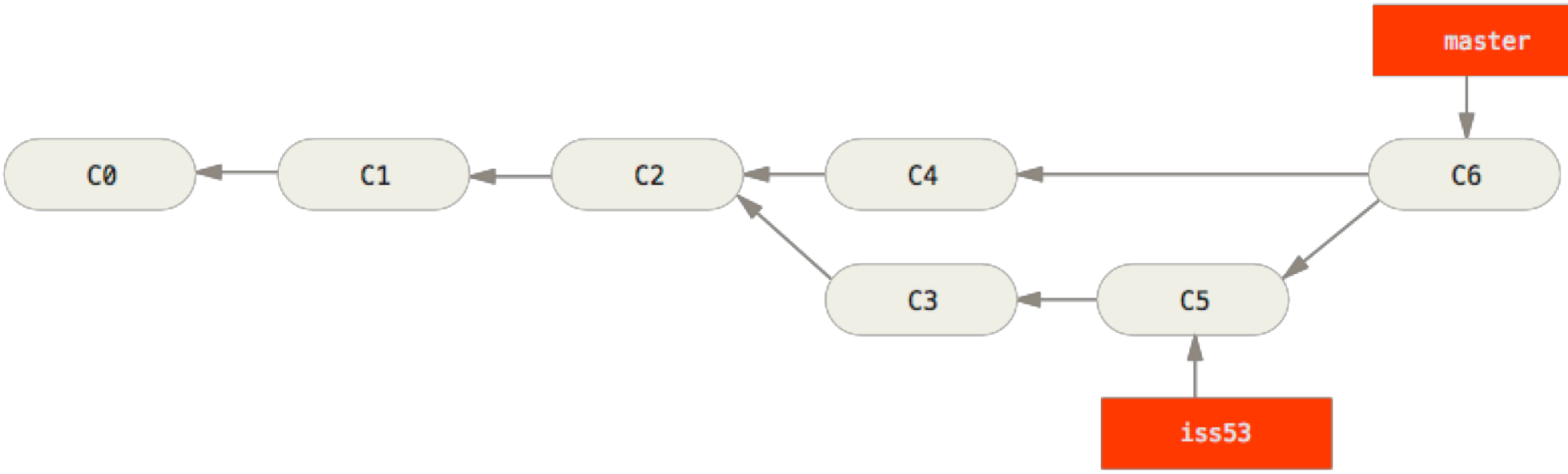
- **Merging** is the process of combining changes from different branches.



https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Merge changes

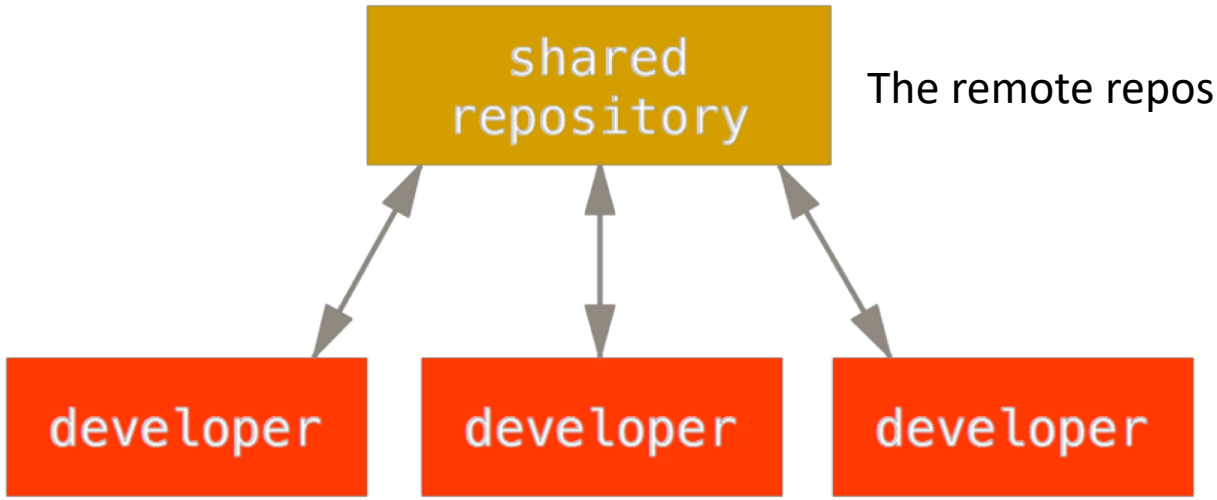
- When a merge takes place, git creates a new commit.



https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Working with a remote repository

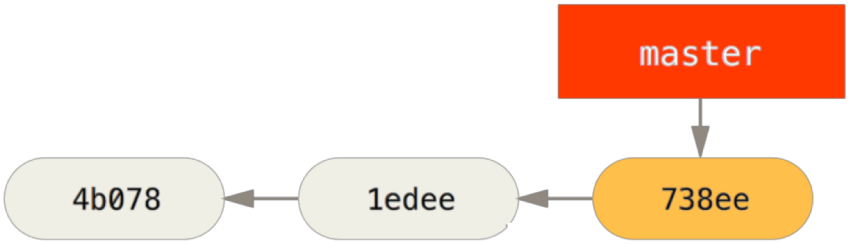
- A **remote repository** (aka simply **remote**) is a git repository that sits on an external server.
- The **origin** is the default nickname that git uses for a remote repository.
- Origin is used since it's much easier to refer to "origin" instead of the address "git@gitlab.moffitt.usf.edu:stewarpa/my_new_project.git".



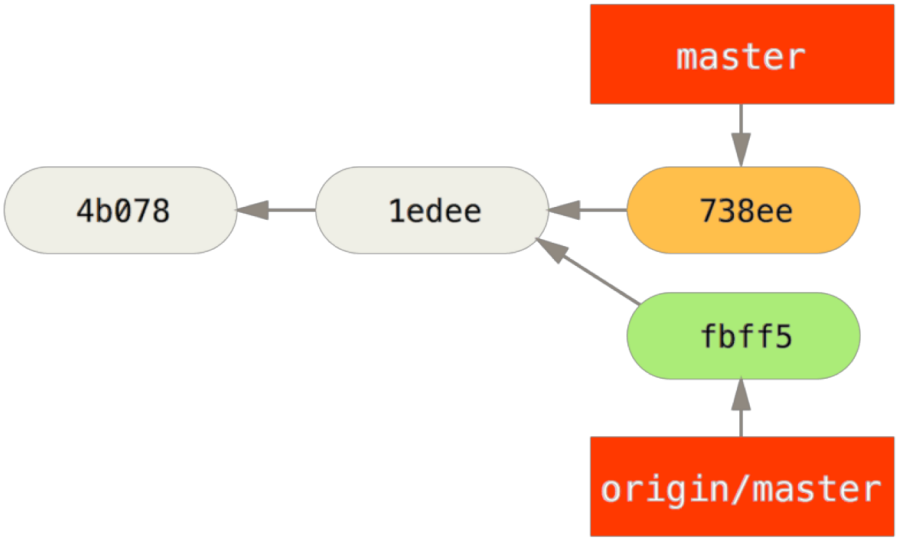
https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows

Working with a remote repository

- Pretend we have made some commits locally.



- Our collaborator informs us they have **pushed** some changes to the remote repository, and we would like to incorporate those changes ourselves.
- We perform a **fetch** to retrieve the collaborator's changes.
- Notice how this is feels very similar to branching.



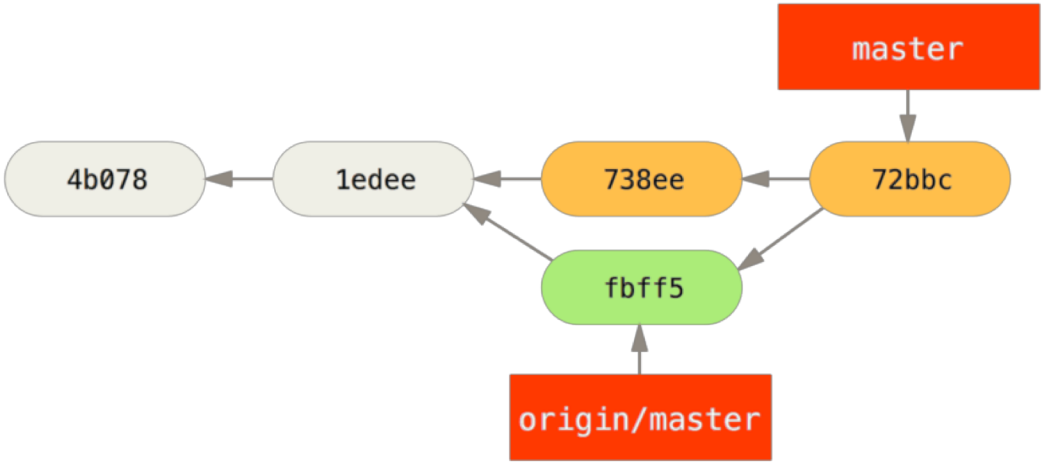
Our local copy

Fetched from remote

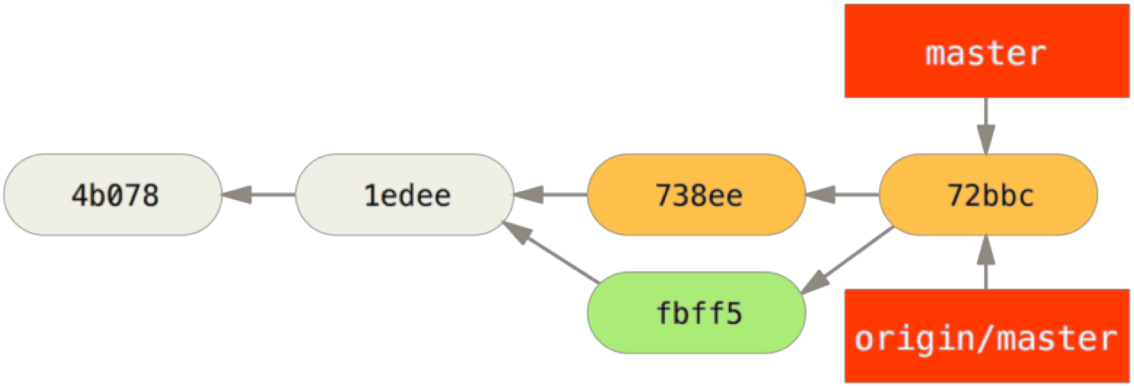
https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows

Working with a remote repository

- Our collaborator's code looks good, so we merge it with our own changes.



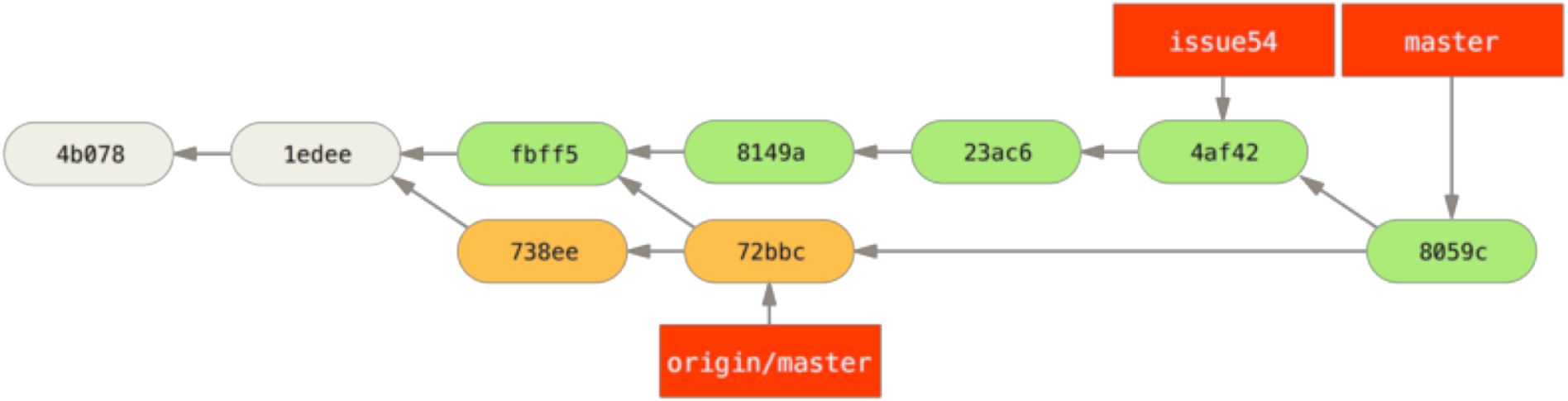
- We want our collaborator to be able to see their changes combined with our own, so now we push the changes back to the origin. This causes the origin/master branch and the master branch to be at the same commit.



https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows

Working with a remote repository

- In practice, this get much more complicated….

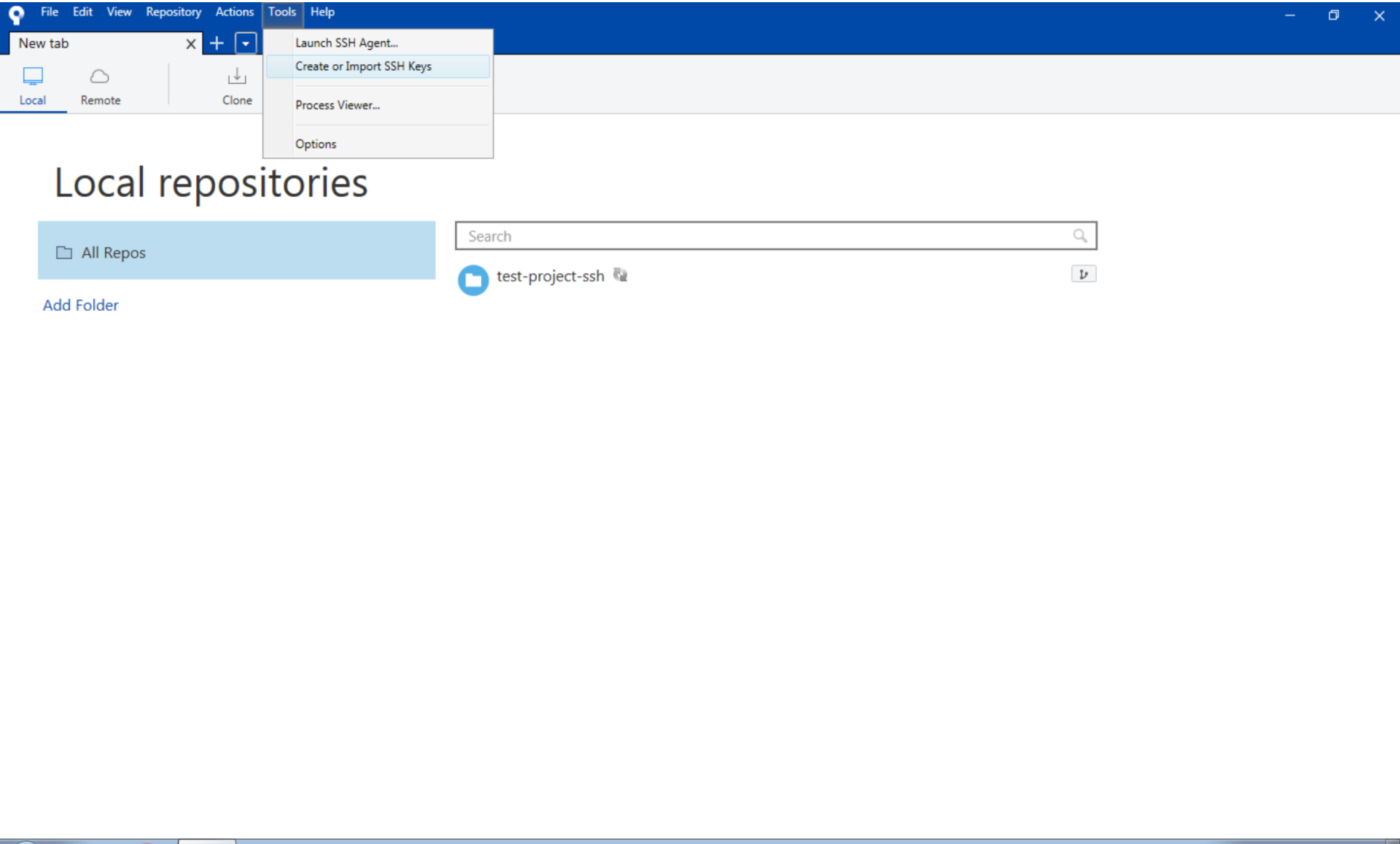


- …but lets stop here and do some hands-on learning so you can see how easy git is in practice.

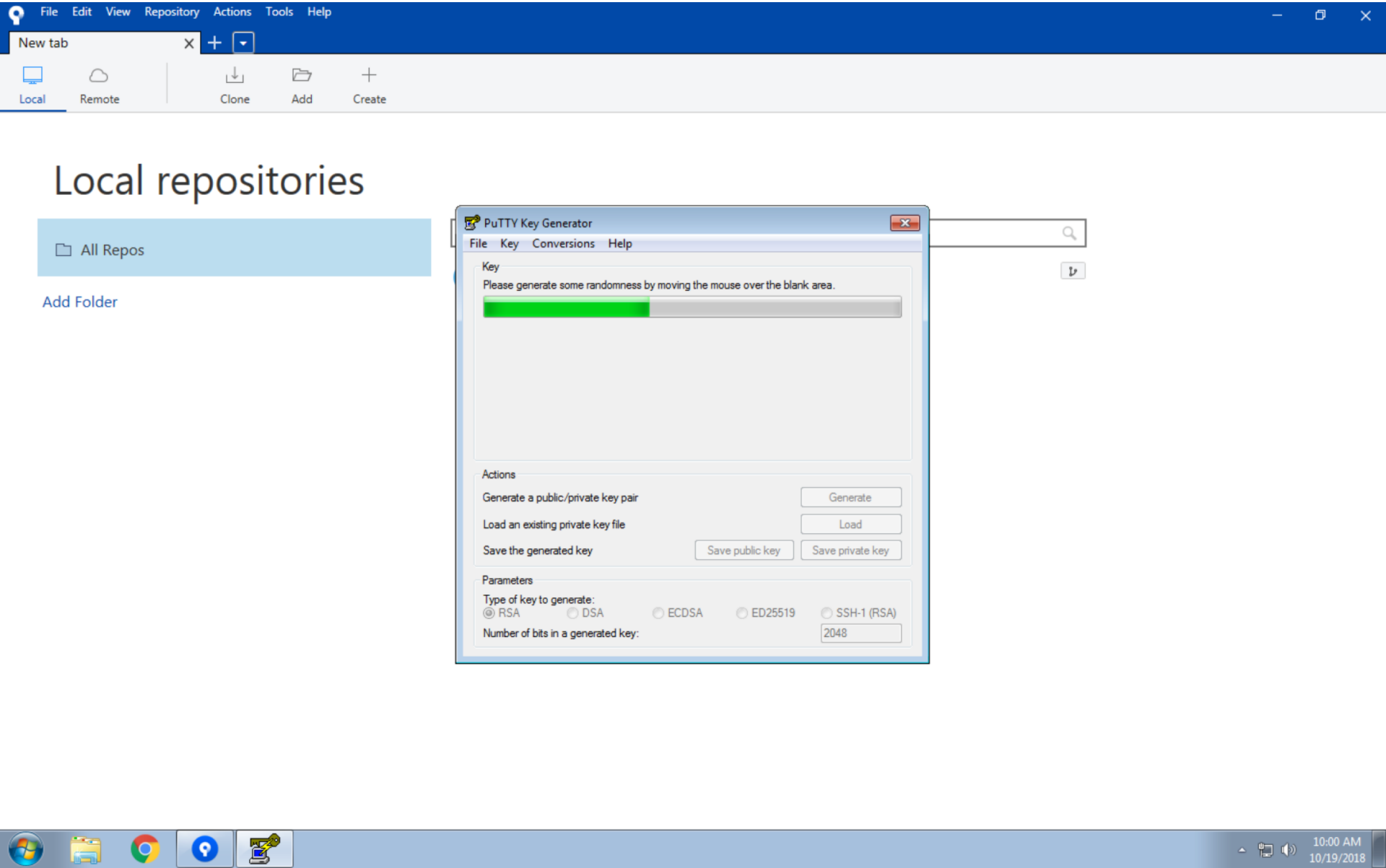
https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows

For Windows users installing Sourcetree

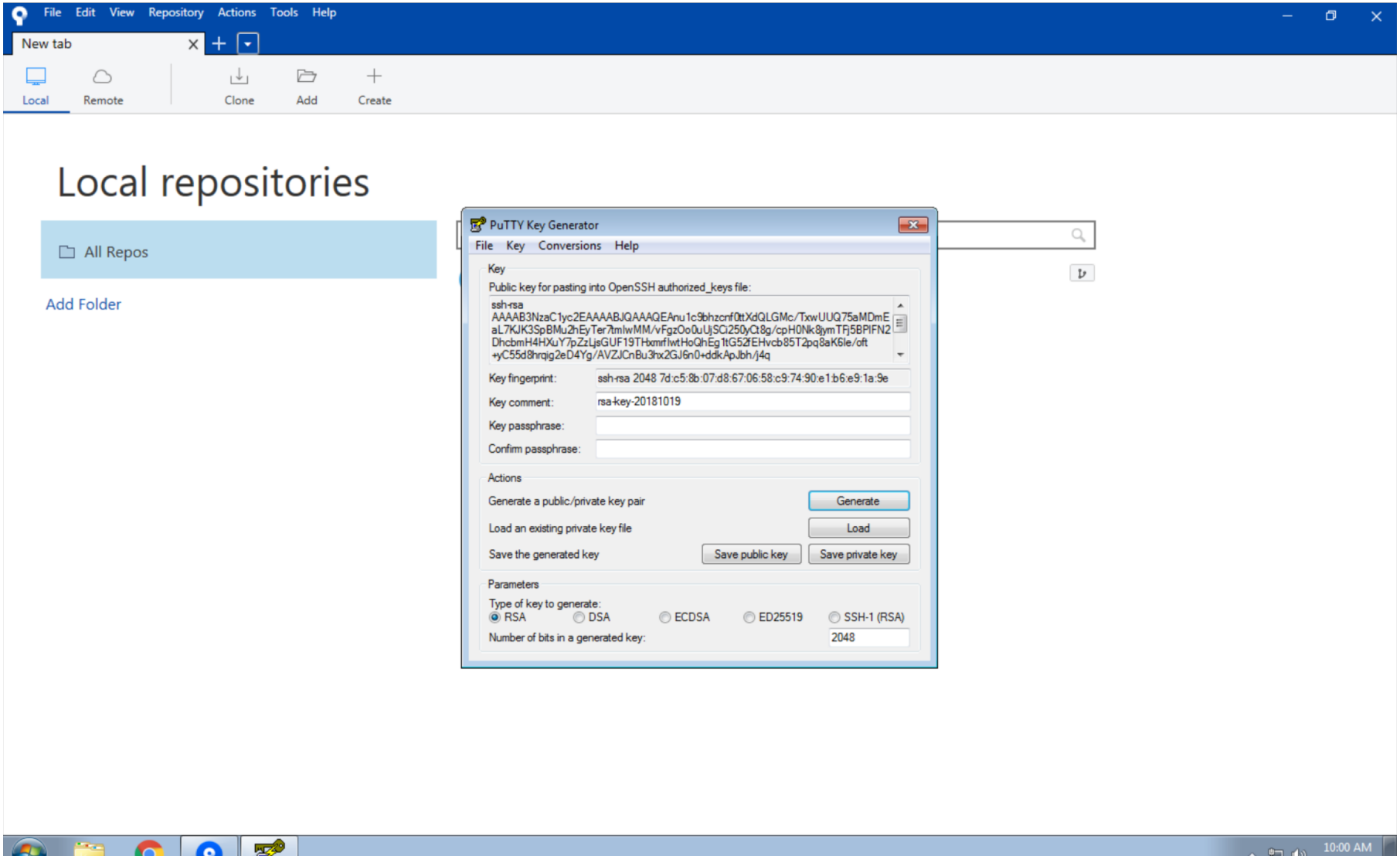
- Open Sourcetree and click Tools -> Create or Import SSH Keys



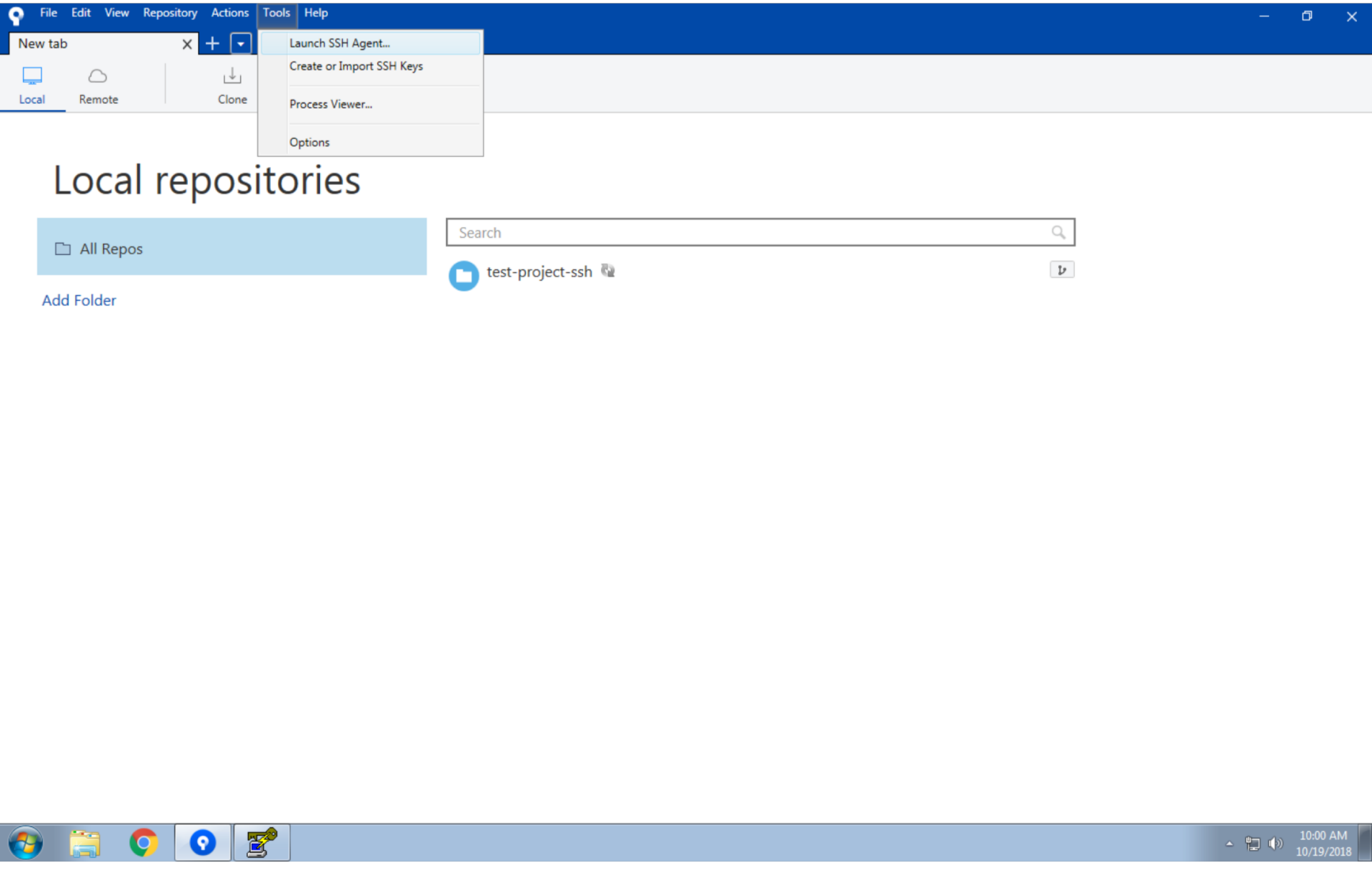
- Make sure “RSA” is checked and click “Generate”.
- Move your mouse around to generate the key.



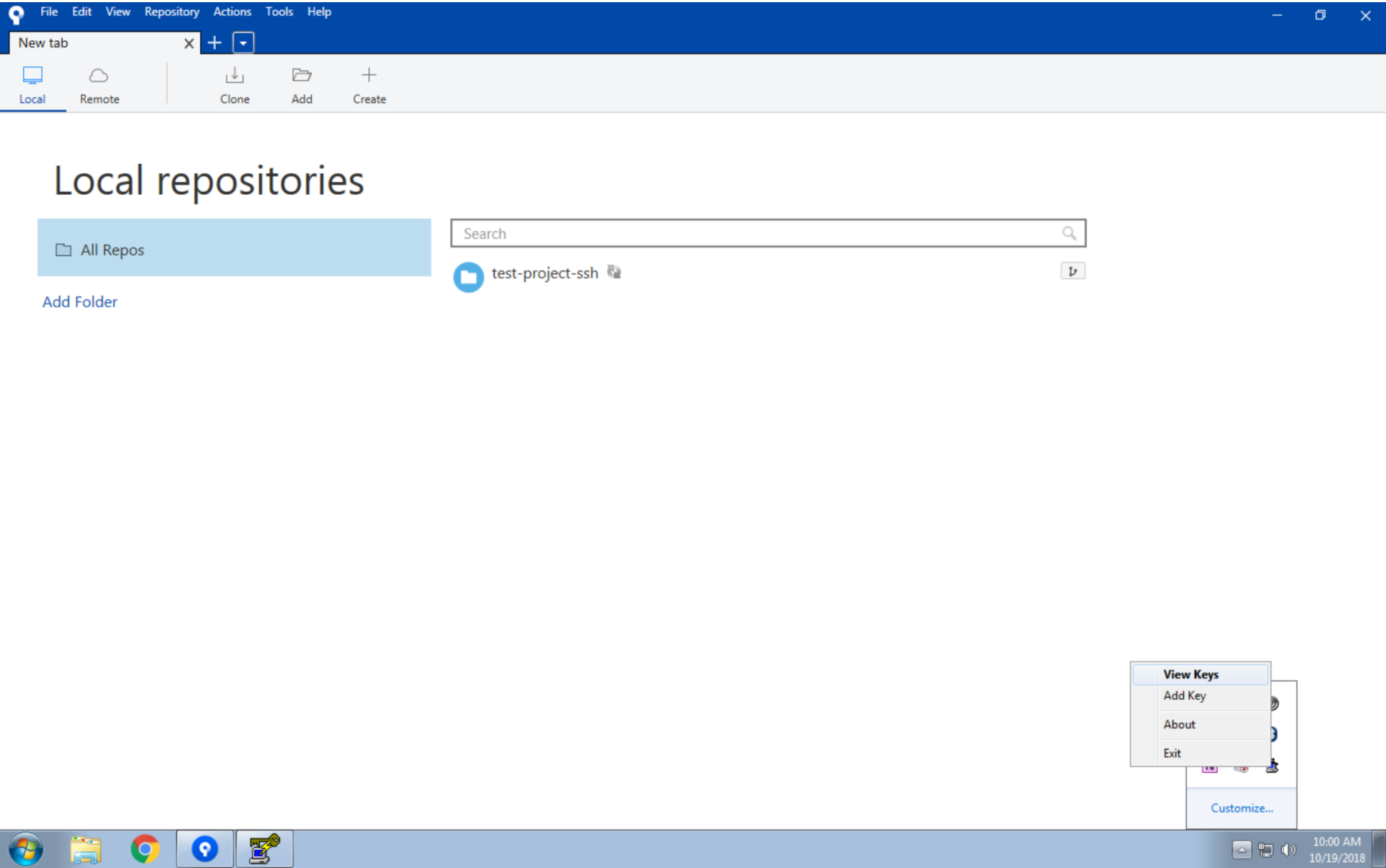
- You've now generated the key pair. The public key is shown but the private key is not.
- The public key needs to be pasted into GitLab, Git Hub, etc, while the private key needs to be loaded into your SSH agent, so copy it to your clipboard now or keep this window open.
- For future reference, click "Save public key" and save it somewhere on your hard drive.
- For future reference, click "Save private key" and save it somewhere on your hard drive.



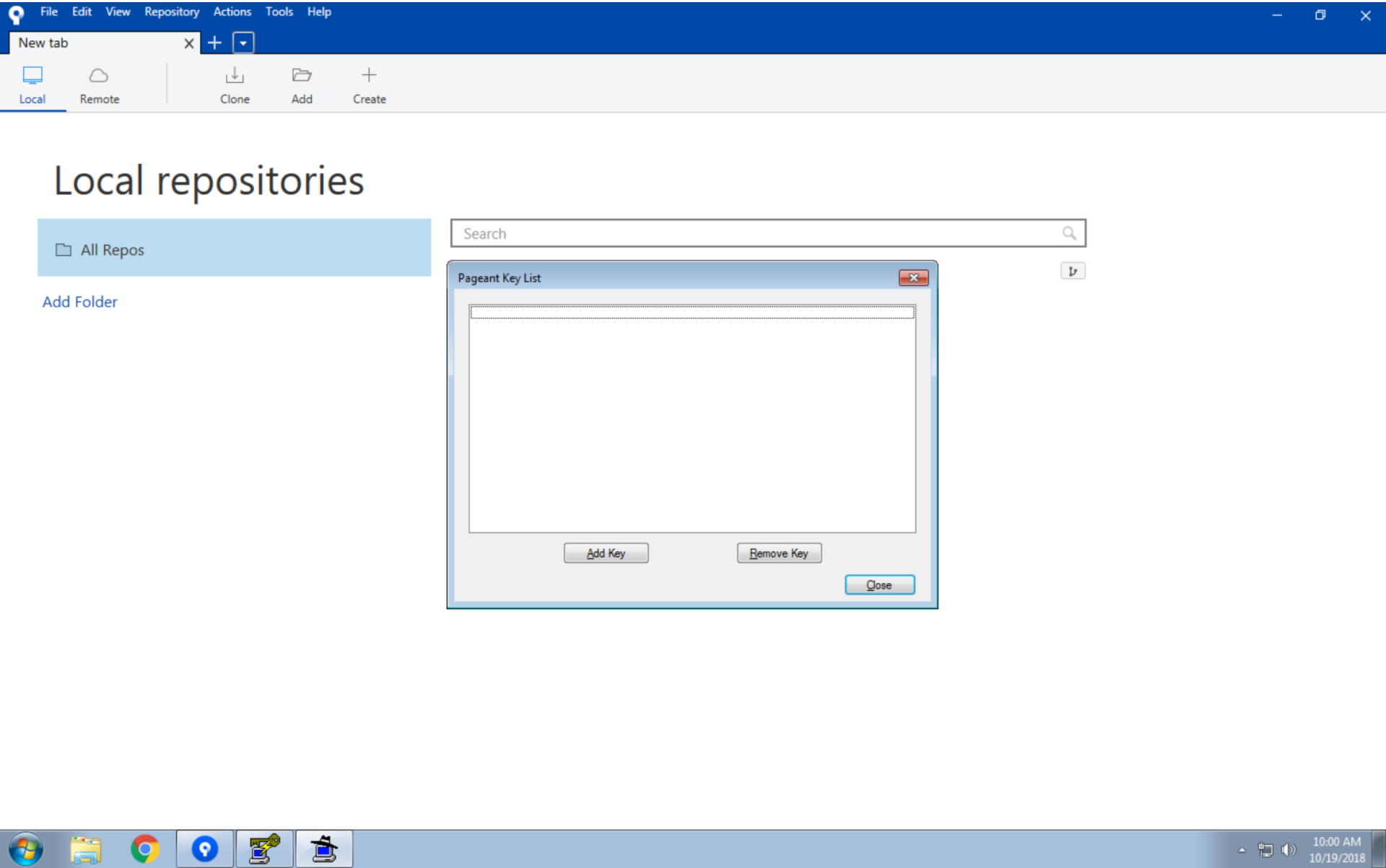
- Click Tools -> Launch SSH Agent…
- It might seem like nothing happened, but check your system tray (bottom right of the screen).



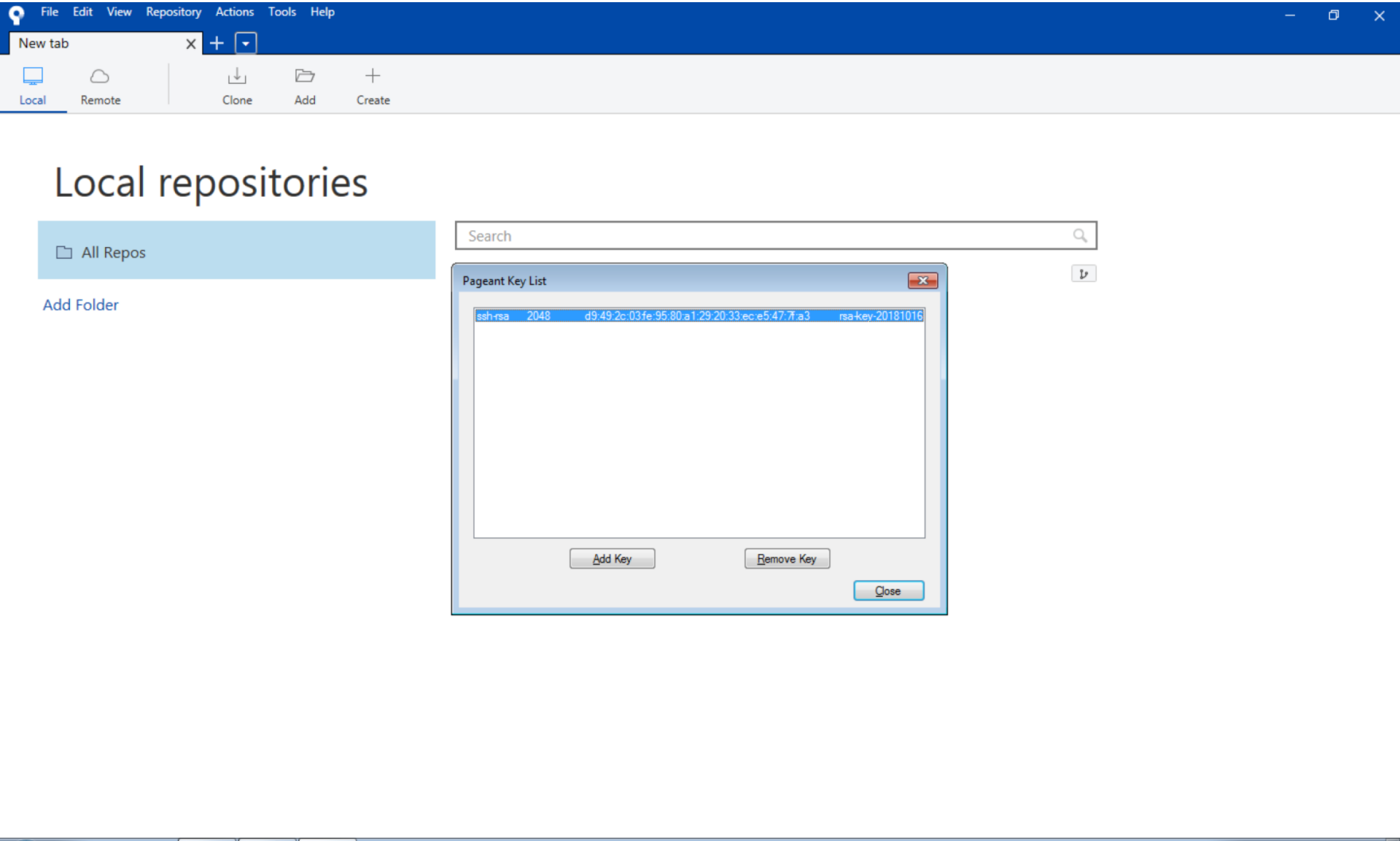
- Find the icon that looks like a computer with a hat. This is your SSH agent.
- Right click and click “View Keys”.



- If you just installed Sourcetree, you probably don't have any keys.
- Click "Add Key" and then navigate to the private key file (ends with .ppk) that you just created and saved with Sourcetree.



- Your key should now be added. You can close the “Pageant Key List” window.



- Log in to GitLab, Git Hub, etc. and find your user settings (in GitLab, it is the top right icon with the drop down menu under “Setings”)
- Find the section with SSH keys (this will be on the left hand side in GitLab)
- Take the **public** key you copied to your clipboard earlier or copy it now from your still-open Sourcetree window and paste it into the text box designated for SSH keys. You should give this key a name to help you manage them in the future(you will have unique keys for your work and home computers, for example). Click “Add Key” and you’re all set to push/pull!