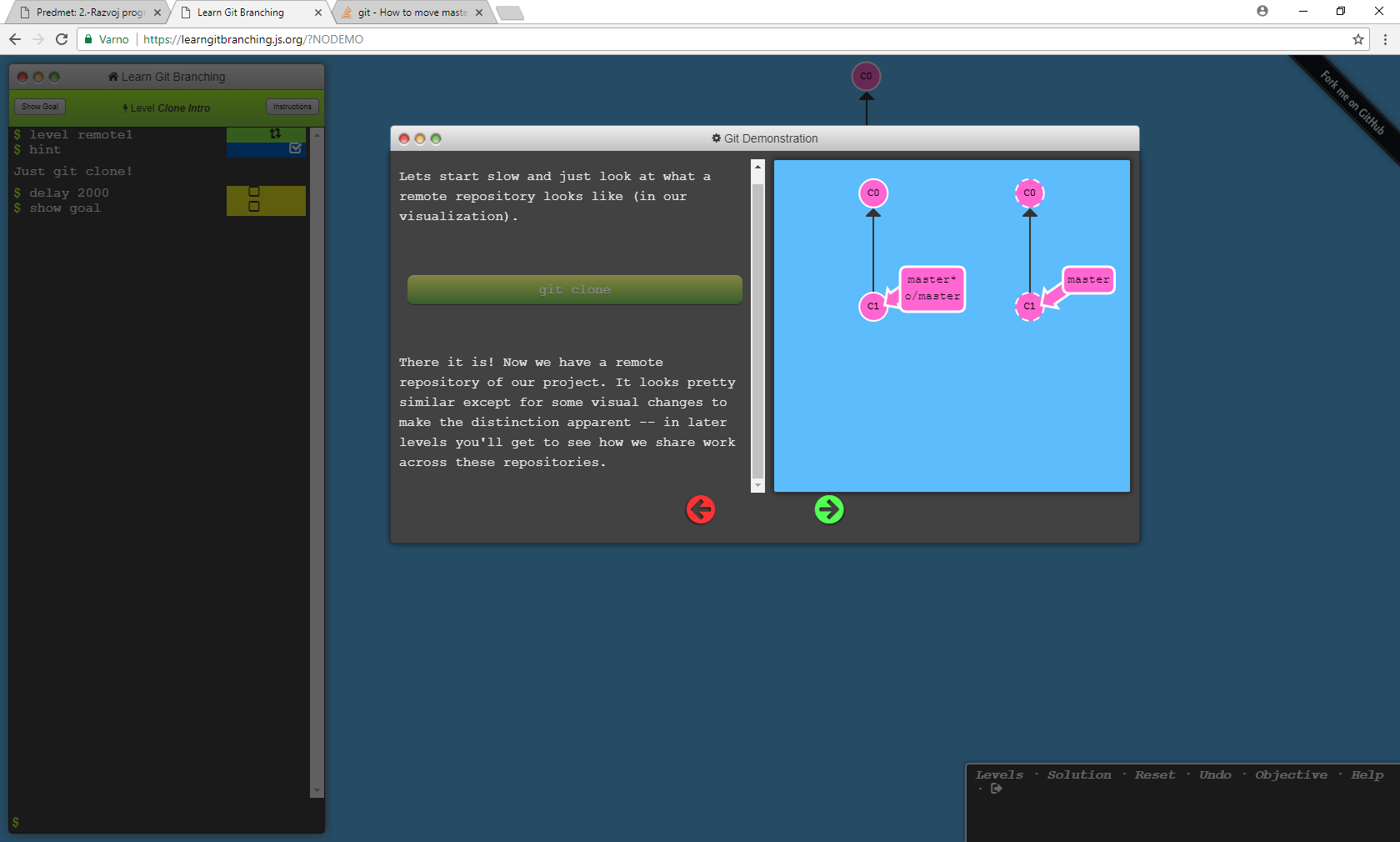
**Our Command to create remotes**

Up until this point, Learn Git Branching has focused on teaching the basics of *local*repository work (branching, merging, rebasing, etc). However now that we want to learn about remote repository work, we need a command to set up the environment for those lessons. git clone will be that command

Technically, git clone in the real world is the command you'll use to create *local* copies of remote repositories (from github for example). We use this command a bit differently in Learn Git Branching though -- git clone actually makes a remote repository out of your local one. Sure it's technically the opposite meaning of the real command, but it helps build the connection between cloning and remote repository work, so let's just run with it for now.



**Git Remote Branches**

Now that you've seen git clone in action, let's dive into what actually changed.

The first thing you may have noticed is that a new branch appeared in our local repository called o/master. This type of branch is called a *remote* branch; remote branches have special properties because they serve a unique purpose.

Remote branches reflect the *state* of remote repositories (since you last talked to those remote repositories). They help you understand the difference between your local work and what work is public -- a critical step to take before sharing your work with others.

Remote branches have the special property that when you check them out, you are put into detached HEAD mode. Git does this on purpose because you can't work on these branches directly; you have to work elsewhere and then share your work with the remote (after which your remote branches will be updated).

### What is o/?

You may be wondering what the leading o/ is for on these remote branches. Well, remote branches also have a (required) naming convention -- they are displayed in the format of:

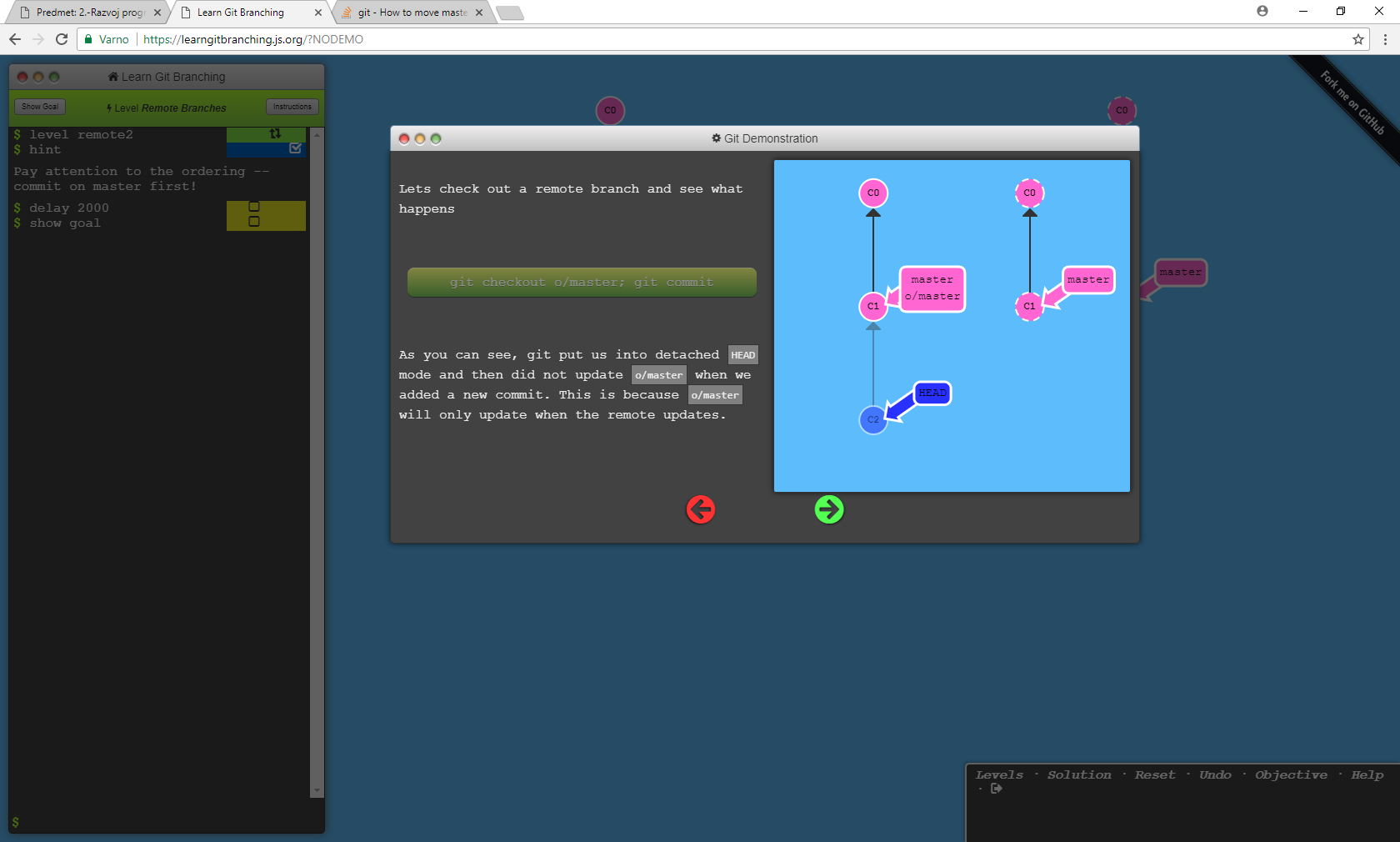
* <remote name>/<branch name>

Hence, if you look at a branch named o/master, the branch name is master and the name of the remote is o.

Most developers actually name their main remote origin, not o. This is so common that git actually sets up your remote to be named origin when you git clone a repository.

Unfortunately the full name of origin does not fit in our UI, so we use o as shorthand :( Just remember when you're using real git, your remote is probably going to be named origin!

That's a lot to take in, so let's see all this in action.

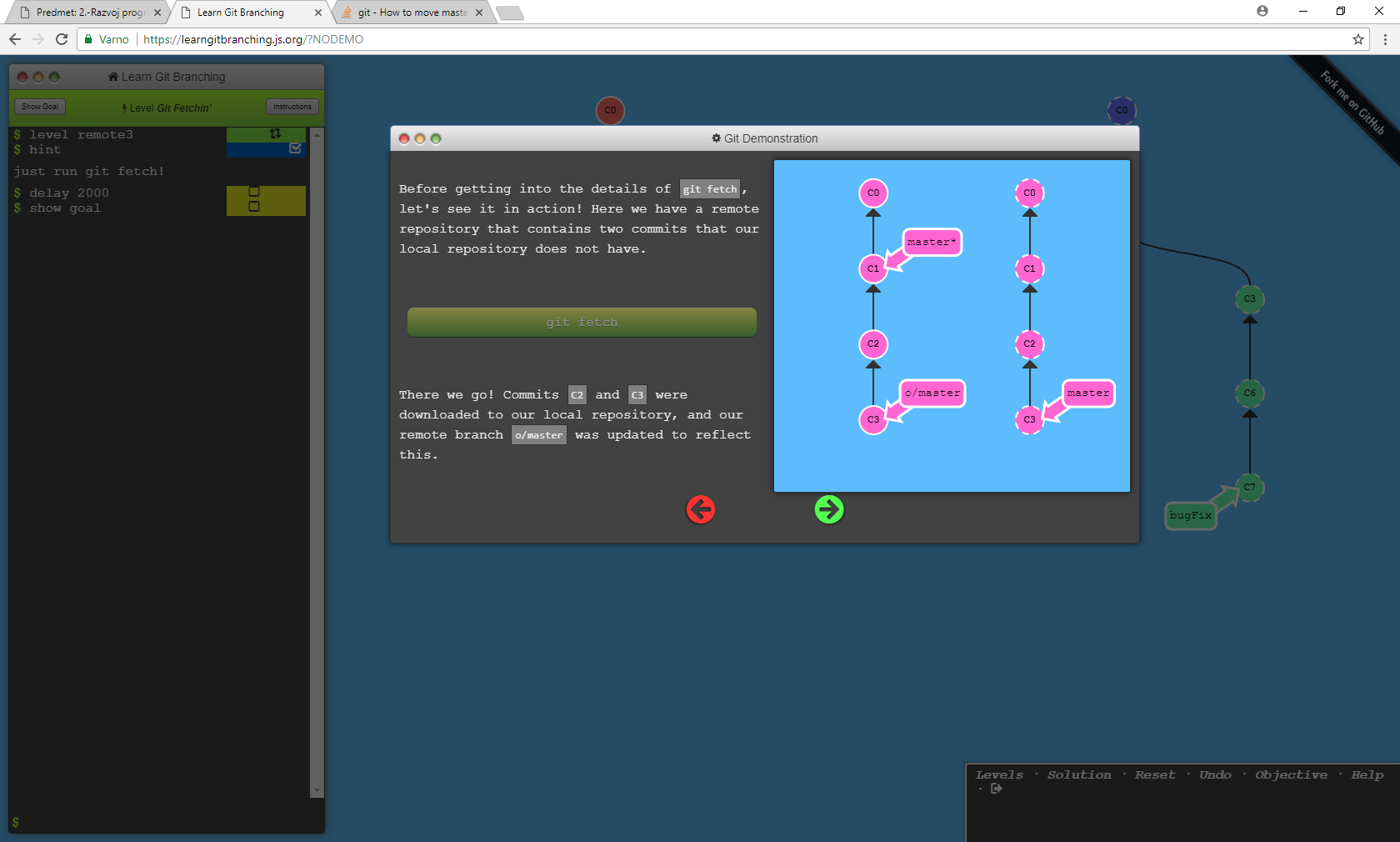


## Git Fetch

Working with git remotes really just boils down to transferring data to and from other repositories. As long as we can send commits back and forth, we can share any type of update that is tracked by git (and thus share work, new files, new ideas, love letters, etc.).

In this lesson we will learn how to fetch data from a remote repository -- the command for this is conveniently named git fetch.

You'll notice that as we update our representation of the remote repository, our remotebranches will update to reflect that new representation. This ties into the previous lesson on remote branches



### What fetch does

git fetch performs two main steps, and two main steps only. It:

* downloads the commits that the remote has but are missing from our local repository, and...
* updates where our remote branches point (for instance, o/master)

git fetch essentially brings our local representation of the remote repository into synchronization with what the actual remote repository looks like (right now).

If you remember from the previous lesson, we said that remote branches reflect the state of the remote repositories since you last talked to those remotes. git fetch is the way you talk to these remotes! Hopefully the connection between remote branches and git fetch is apparent now.

git fetch usually talks to the remote repository through the Internet (via a protocol like http:// or git://).

### What fetch doesn't do

git fetch, however, does not change anything about your local state. It will not update your master branch or change anything about how your file system looks right now.

This is important to understand because a lot of developers think that running git fetchwill make their local work reflect the state of the remote. It may download all the necessary data to do that, but it does not actually change any of your local files. We will learn commands in later lessons to do just that :D

So at the end of the day, you can think of running git fetch as a download step.

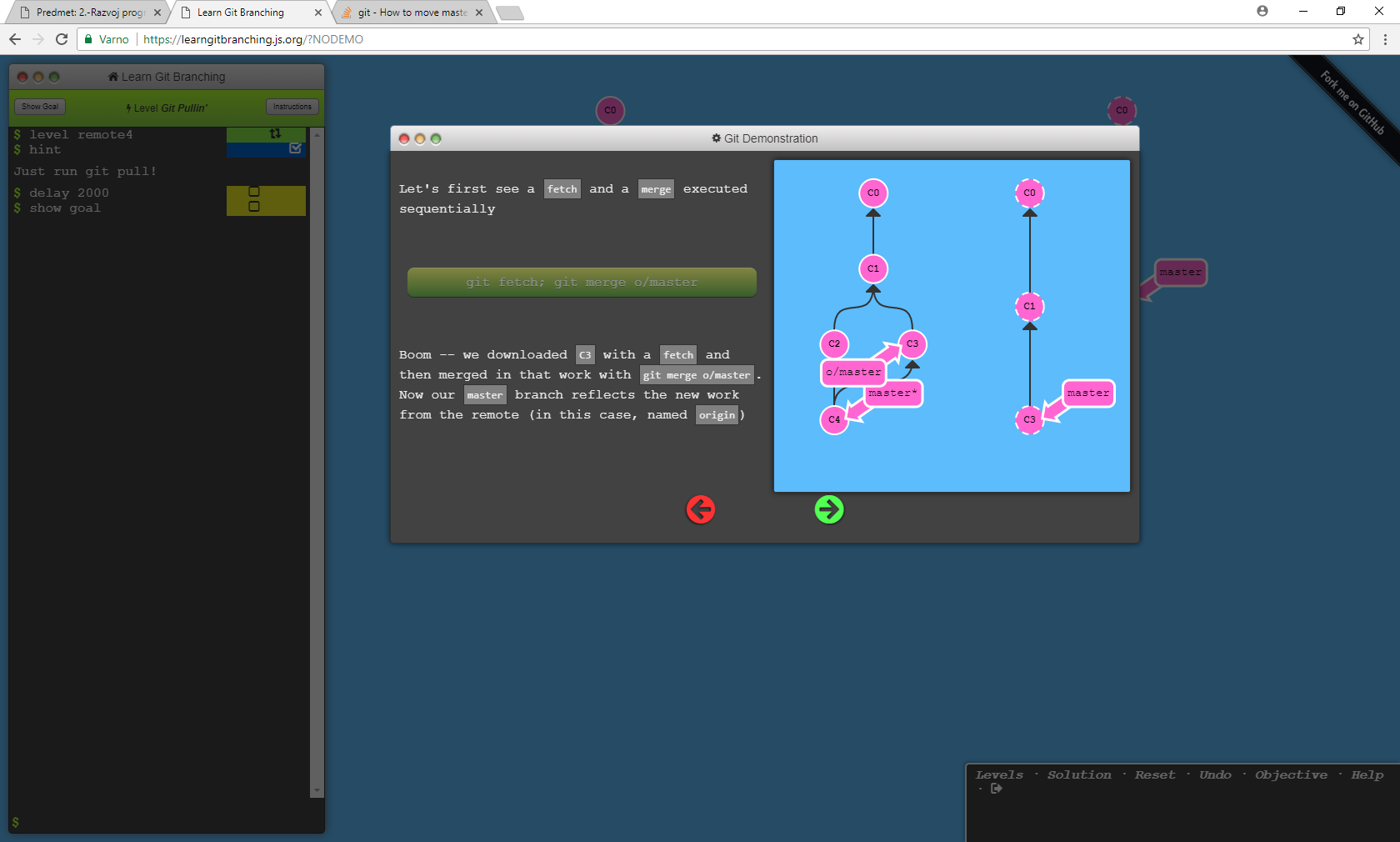
**Git Pull**

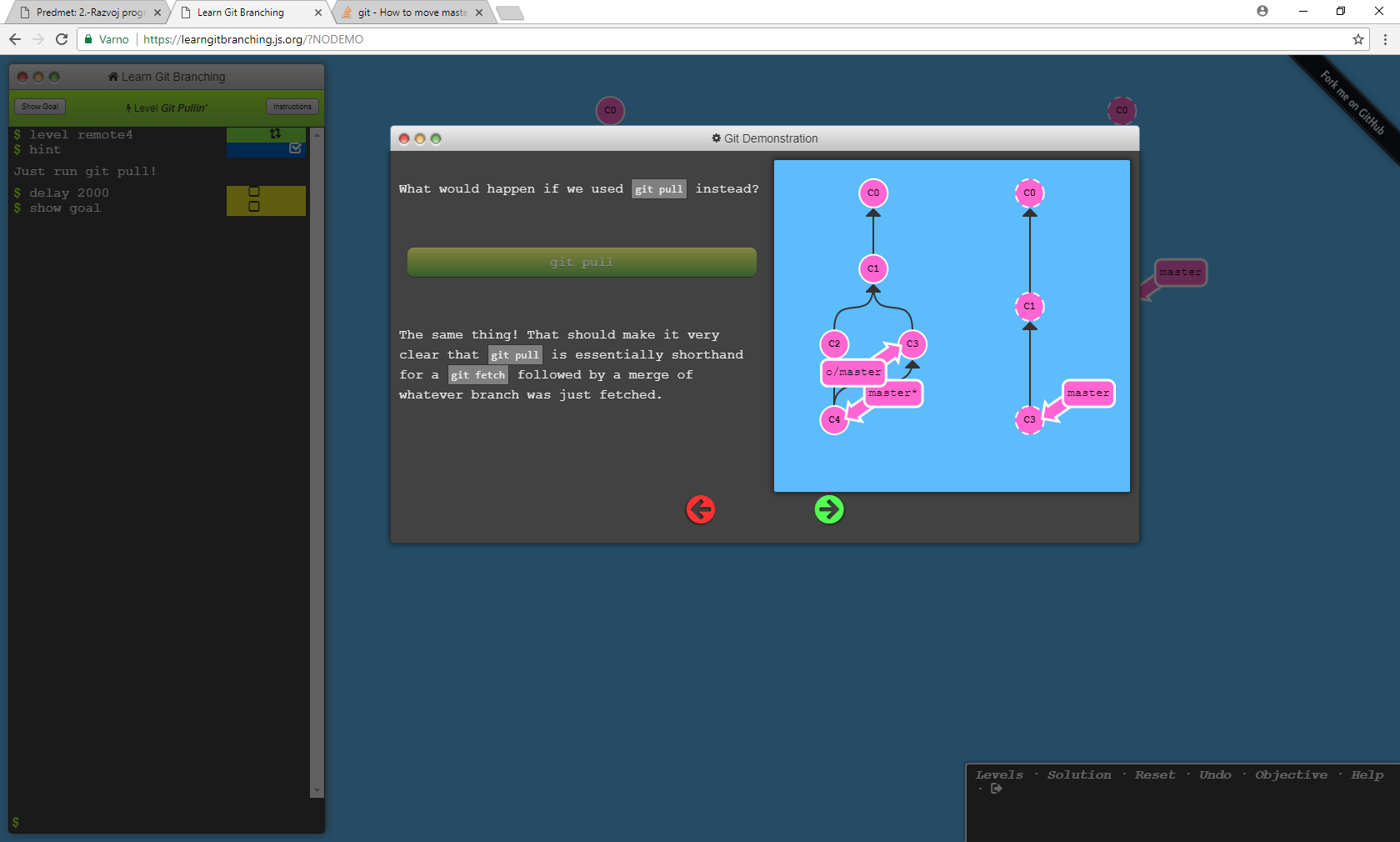
Now that we've seen how to fetch data from a remote repository with git fetch, let's update our work to reflect those changes!

There are actually many ways to do this -- once you have new commits available locally, you can incorporate them as if they were just normal commits on other branches. This means you could execute commands like:

* git cherry-pick o/master
* git rebase o/master
* git merge o/master
* etc., etc.

In fact, the workflow of *fetching* remote changes and then *merging* them is so common that git actually provides a command that does both at once! That command is git pull.





We will explore the details of git pull later (including options and arguments), but for now let's try it out in the level.

Remember -- you can actually solve this level with just fetch and merge, but it will cost you an extra command :P