Introduction to GitHub

**What is GitHub?**

In earlier exercises, we called out that Git is a distributed version control system. Distributed means that each developer has a copy of the whole repository on their local machine.

Each copy is a peer of the others. But we can host one of these copies on a server and then use it as a remote repository for the other copies. This lets us synchronize work between copies through this server. Any of us can create a Git server like this one, and many companies have similar internal services. But if we don't want to set up a Git server ourself and host oour repositories, we can use an online service like GitHub.

GitHub is a web-based Git repository hosting service. On top of the version control functionality of Git, GitHub includes extra features like bug tracking, wikis, and task management. GitHub lets us share and access repositories on the web and copy or clone them to our local computer, so we can work on them. GitHub is a popular choice with a robust feature set, but it's not the only one. Other services that provide similar functionality are BitBucket, and GitLab.

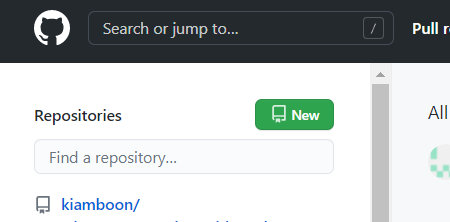
For the rest of this exercise, we'll be using GitHub for our examples. But feel free to use the tool that best fits our needs. GitHub provides free access to a Git server for public and private repositories. It limits the number of contributors for the free private repositories, and offers an unlimited private repository service for a monthly fee. We'll be using a free repository for our examples, which is fine for educational use, small personal projects, or open source development.

A word of caution on how we can manage these repos though. If hackers get hold of information about our organization's IT infrastructure, they can use it to try and break into our network. So make sure you treat this information as confidential. For real configuration and development work, we should use a secure and private Git server, and limit the people authorized to work on it.

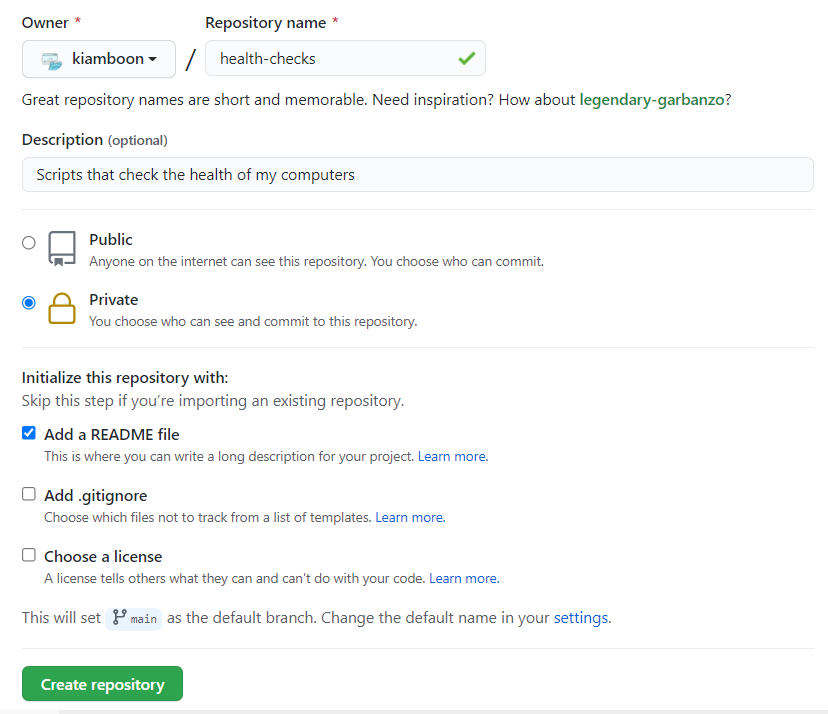
To use GitHub, the first thing we need to do is create an account if we don't have one already. Signing up online is free and relatively simple. Once we've done that, you can either create your own repos or contribute to repos from other projects. If you don't have a GitHub account yet, now is a good time to create one. Visit github.com to sign up for their service.

**Basic Interaction with GitHub?**

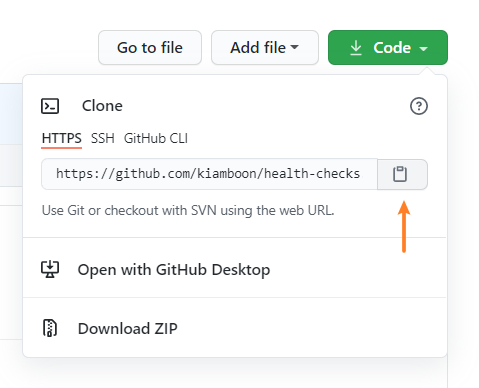
As we called out, GitHub is an online service. To use it, we first need to create an account on the site. Once we have your account, we’re ready to create our brand-new repository on GitHub.



* We'll start by clicking the ‘New’ in green button to create a repository.



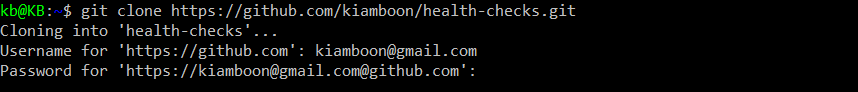
* This will take us to the repo creation wizard. The wizard is pretty straightforward.
* The first thing we need to do is give a name for our repo. We'll call this repo health-checks.
* After that comes a description of what the repo will be used for.
* Then we need to select whether we want the repo to be public or private. We'll go with private for now.
* Finally, the wizard can help us get started with some few initialization files like a README, a gitignore, or license file. We'll go with just the README for now, and then create the repo.



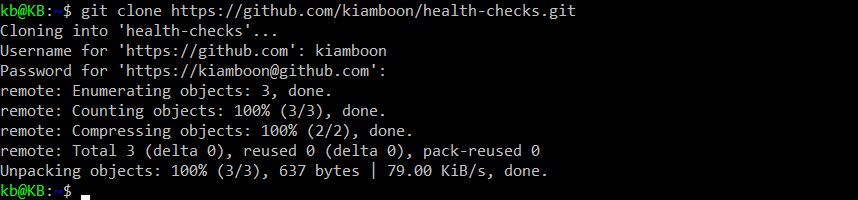
* Using the wizard, we created the repo and have a fresh remote repository ready to go. First step is to create a local copy of the repository.
* Click the ‘code’ in green button, we'll do that by using the git clone command followed by the URL of the repo. GitHub conveniently lets us copy the URL from our repo from the interface so that we don't have to type it.



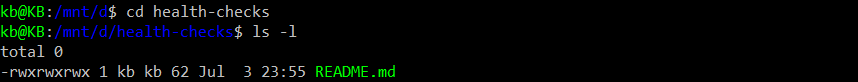
* We're now ready to clone the repo into our computer. Open the Ubuntu and then do that by calling **git clone** and paste in the URL we copied.



* To do this, GitHub will ask for our username and password.
* Note: Because this is a private repo, account with 2FA enabled will not able to use git over https. Alternatively, create a personal access token at <https://github.com/settings/tokens>, and use token as username and enter with blank password. We won’t need any credentials to access all of our private repos.



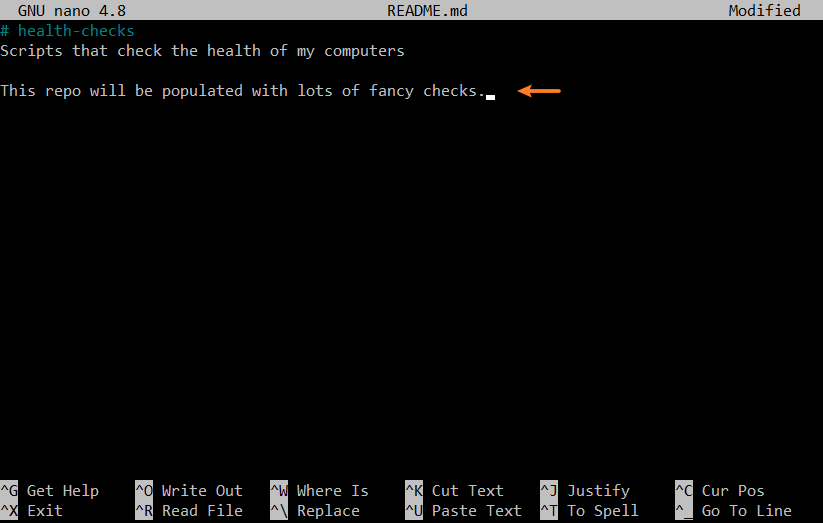
* Just like that, we've downloaded a copy of the remote repository from GitHub onto the local machine.
* This means that we can perform all the git actions that we've learned up till now.



* Since the repo is called health-checks, a directory with that name was automatically created for us and now has the working tree of the Repository in it. So let's change that directory and look at the contents.
* Our repo is basically empty. It only has the README.md file that GitHub created for us. This file is in a special format called **markdown**.



* Open the editor to go in README.md

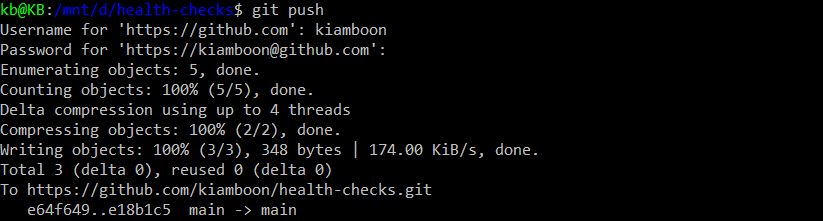


* Let's add a bit more content to it. And then save it and exit.

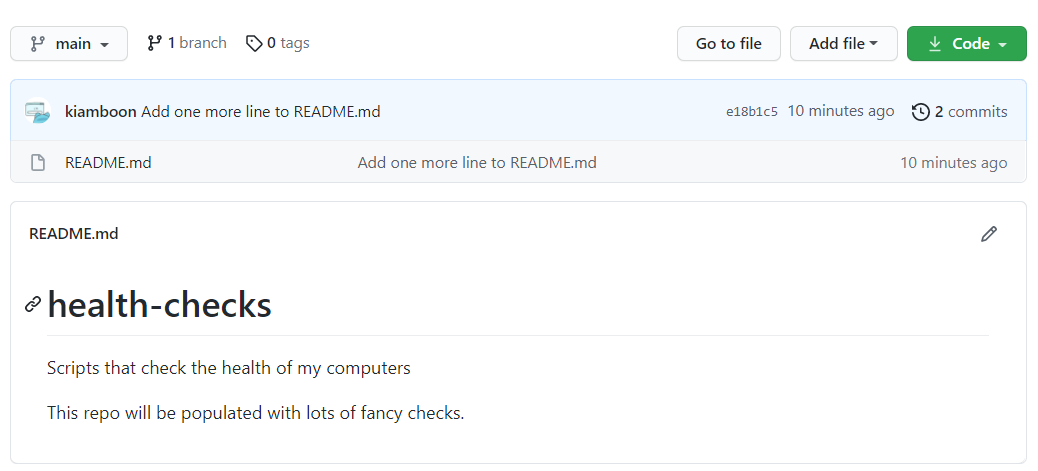


* We've changed this file. What do we need to do now? We need to stage the change and committed. We've seen a couple of different ways to do that. Let's use our shortcuts to do this in just one command.

Okay. We've modified our README file. But we've seen all this before. We got to remote repository set up on GitHub. So let's use it. We can send our changes to that remote repository by using the **git push** command which will gather all the snapshots we've taken and send them to the remote repository. In this case, we've only taken one snapshot. We'll talk more about what's going on behind the scenes with git push and remote repositories in later videos. But the mechanics are pretty simple.



* To push our modified README up to GitHub, we'll just call **git push**.
* Once again, we're asked for our password. After that, we see a bunch of messages from git related to the push.



* So if we check our repository on GitHub, we should see the updated message (refresh the page if needed).

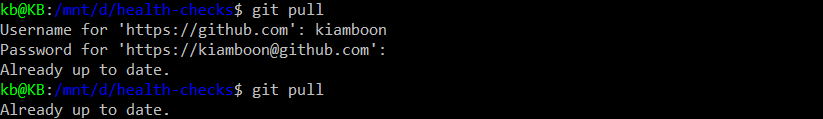
After making changes to our local repository, how do we update the remote repository to reflect our changes?

* Use the git push command to send snapshots to the remote repository. The git push command gathers all the snapshots we've taken and sends them to the remote repository.

Pretty cool, right? We've taken the local changes on our computer and pushed them out to a remote repository hosted on GitHub. You've probably noticed that we had to enter our password both when retrieving the repo and when pushing changes to the repo. There are a couple ways to avoid having to do this. One way is to create an **SSH key-pair** and store the public key in our profile so that GitHub recognizes our computer.



* Another option is to use a **Credential helper** which caches our credentials for a time window so that we don't need to enter our password with every interaction.
* Git already comes with a credential helper baked in. We just need to enable it. We do that by calling **git config - - global credential.helper cache**
* Now that we've enabled the credential helper, we'll need to enter our credentials once more. After that, they'll be cached for 15 minutes.



* To check this, we can try another git command, **git** **pull** which is the command we use to retrieve new changes from the repository. We'll enter our credentials on the first call to the command and they'll be cached, so we won't need to enter them again.

With that, we've seen how to create repositories on GitHub, clone our remote repository, push changes to it, and pull changes from it.