Creating a Repository

Once Git is configured, we can start using it.

We will continue with the story of Wolfman and Dracula who are investigating if it is possible to send a planetary lander to Mars.

Werewolf vs dracula by b-maze / Deviant Art. Mars by European Space Agency / CC-BY-SA 3.0 IGO. Pluto / Courtesy NASA/JPL-Caltech. Mummy © Gilad Fried / The Noun Project / CC BY 3.0. Moon © Luc Viatour / https://lucnix.be / CC BY-SA 3.0.

First, let's create a directory in `Desktop` folder for our work and then move into that directory:

```
$ cd ~/Desktop
$ mkdir planets
$ cd planets
```

{: .language-bash}

Then we tell Git to make `planets` a repository – a place where Git can store versions of our files:

```
$ git init
```

{: .language-bash}

It is important to note that `git init` will create a repository that includes subdirectories and their files—there is no need to create separate repositories nested within the `planets` repository, whether subdirectories are present from the beginning or added later. Also, note that the creation of the `planets` directory and its initialization as a repository are completely separate processes.

If we use `ls` to show the directory's contents, it appears that nothing has changed:

```
$ ls
```

{: .language-bash}

But if we add the `-a` flag to show everything, we can see that Git has created a hidden directory within `planets` called `.git`:

```
$ ls -a
```

{: .language-bash}

```
.	..	.git
```

{: .output}

Git uses this special subdirectory to store all the information about the project, including all files and sub-directories located within the project's directory. If we ever delete the `.git` subdirectory, we will lose the project's history.

Next, we will change the default branch to be called `main`. This might be the default branch depending on your settings and version of git. See the setup episode for more information on this change.

```
git checkout -b main
```

{: .language-bash} ~~~ Switched to a new branch 'main' ~~~ {: .output}

We can check that everything is set up correctly by asking Git to tell us the status of our project:

```
$ git status
```

{: .language-bash} ~~~ On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track) ~~~ {: .output}

If you are using a different version of `git`, the exact wording of the output might be slightly different.

Places to Create Git Repositories

Along with tracking information about planets (the project we have already created), Dracula would also like to track information about moons. Despite Wolfman's concerns, Dracula creates a `moons` project inside his `planets` project with the following sequence of commands:

```
$ cd ~/Desktop   # return to Desktop directory
$ cd planets     # go into planets directory, which is already a Git repository
$ ls -a          # ensure the .git subdirectory is still present in the planets directo
$ mkdir moons    # make a subdirectory planets/moons
$ cd moons       # go into moons subdirectory
$ git init       # make the moons subdirectory a Git repository
$ ls -a          # ensure the .git subdirectory is present indicating we have created a
```

{: .language-bash}

Is the `git init` command, run inside the `moons` subdirectory, required for tracking files stored in the `moons` subdirectory?

Solution

No. Dracula does not need to make the `moons` subdirectory a Git repository because the `planets` repository will track all files, sub-directories, and subdirectory files under the `planets` directory. Thus, in order to track all information about moons, Dracula only needed to add the `moons` subdirectory to the `planets` directory.

Additionally, Git repositories can interfere with each other if they are "nested": the outer repository will try to version-control the inner repository. Therefore, it's best to create each new Git repository in a separate directory. To be sure that there is no conflicting repository in the directory, check the output of `git status`. If it looks like the following, you are good to go to create a new repository as shown above:

```
$ git status
```

{: .language-bash} ~~~ fatal: Not a git repository (or any of the parent directories): .git ~~~ {: .output} {: .solution} {: .challenge} ## Correcting `git init` Mistakes Wolfman explains to Dracula how a nested repository is redundant and may cause confusion down the road. Dracula would like to remove the nested repository. How can Dracula undo his last `git init` in the `moons` subdirectory?

Solution – USE WITH CAUTION!

Background

Removing files from a Git repository needs to be done with caution. But we have not learned yet how to tell Git to track a particular file; we will learn this in the next episode. Files that are not tracked by Git can easily be removed like any other "ordinary" files with ~~~ $ rm filename ~~~ {: .language-bash}

Similarly a directory can be removed using `rm -r dirname` or `rm -rf dirname`. If the files or folder being removed in this fashion are tracked by Git, then their removal becomes another change that we will need to track, as we will see in the next episode.

Solution

Git keeps all of its files in the `.git` directory. To recover from this little mistake, Dracula can just remove the `.git` folder in the moons subdirectory by running the following command from inside the `planets` directory:

```
$ rm -rf moons/.git
```

{: .language-bash}

But be careful! Running this command in the wrong directory will remove the entire Git history of a project you might want to keep. Therefore, always check your current directory using the command `pwd`. {: .solution} {: .challenge}