ICT POWER.IT

Microsoft

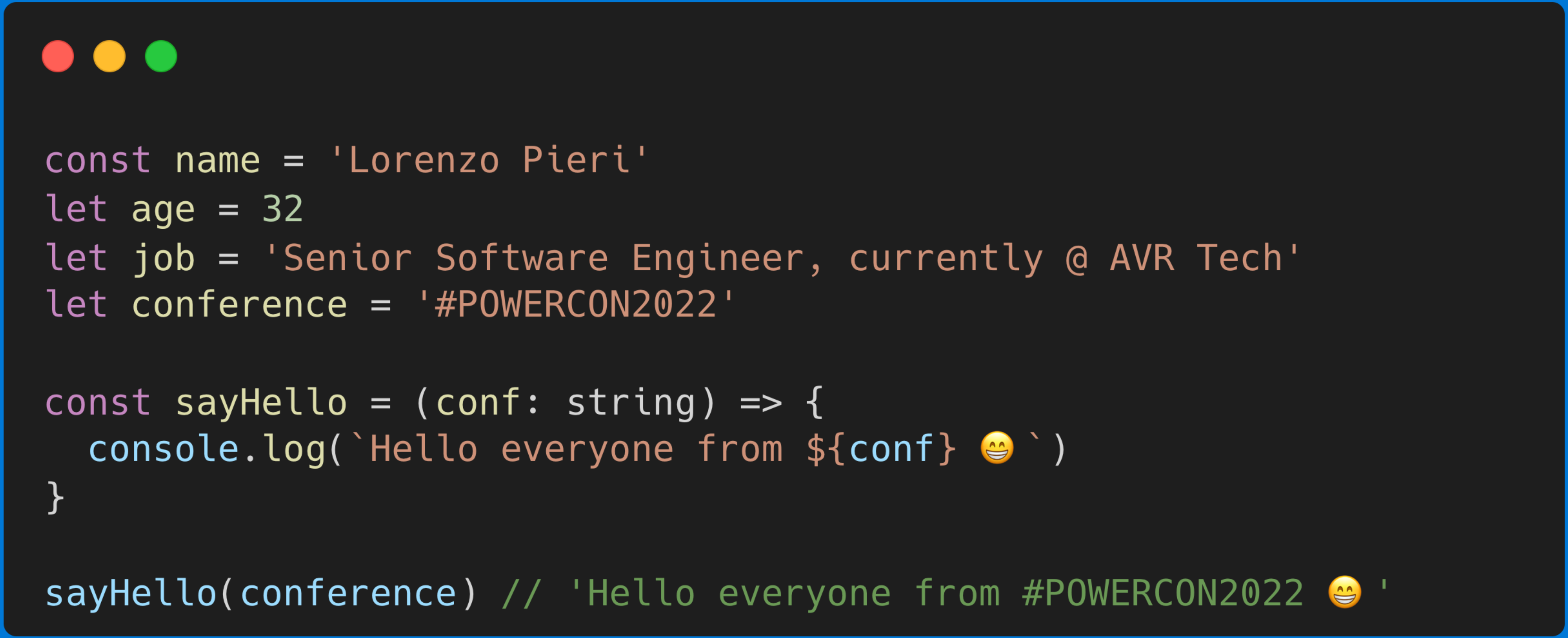
#POWERCON2022
Git Version Control, Zero to Hero

Lorenzo Pieri
Senior Software Engineer
404answernotfound.public@gmail.com

@404answnotfound

lorenzopieri

Who am I?



```
const name = 'Lorenzo Pieri'
let age = 32
let job = 'Senior Software Engineer, currently @ AVR Tech'
let conference = '#POWERCON2022'

const sayHello = (conf: string) => {
  console.log(`Hello everyone from ${conf} 😁 `)
}

sayHello(conference) // 'Hello everyone from #POWERCON2022 😁 '
```

Agenda

- What is version control and what is git
- A little history of git
- Why would you use git version control (use case)
- So many commands! Let's discover them!
- Example
- Where to go from here

What is version control

Version Control allows us to:

- Track and manage all the changes to our codebase, but also our documentation
- Branch out to create new features without breaking current implementations
- Work alongside other people and collaborate on a unified codebase
- Merge our changes so that everyone on the team can take advantage of them
- Revisit our projects' history and logs without any hassle

@surface from Unsplash website

That's a lot of big words

Version Control is just a very useful practice to allow its users to track everything about the project they're working on

Deleted a file months ago from your project and now you need it again? **No problem**.

The entire history of your project is at your reach! (if done properly)

Praveen Thirumurugan from Unsplash website

Just like a time machine, for your projects

Delorean Rental from Unsplash website

What is git, really

Git is, to this day, the most known and used **version control system** used by quite a few Developers from anywhere in the world.

Companies now a days deem git to be a technical skill that should be in everyone's toolbelt.

Taken directly from git's website:

Git is a free and open source *distributed version control system designed to handle everything from small to very large projects with speed and efficiency*

v2.6.0-beta.2
build: release 2.6.0-
build: build 2.6.0-beta.2
feat: dynamic directive arguments for
origin/dynamic-directive-argumen
perf: improve scoped slots change
test: test cases for v-on/v-bind d
refactor: v-bind dynamic argum
test: fix tests, resolve helper c
fix: fix middle modifier
feat: handle dynamic argu
origin/slot-optimizatio
feat: dynamic directive
refactor: extend dom-
fix: fix checkbox eve
test: fix tests in IE/E
refactor: simplify
BACKERS.md
LICENSE
package.json

History of git

Git development started in 2005 by **Linus Torvalds** and the **Linux Development community**.

But why was it needed? Version Control Systems already existed in the day. Not as git, but they existed.

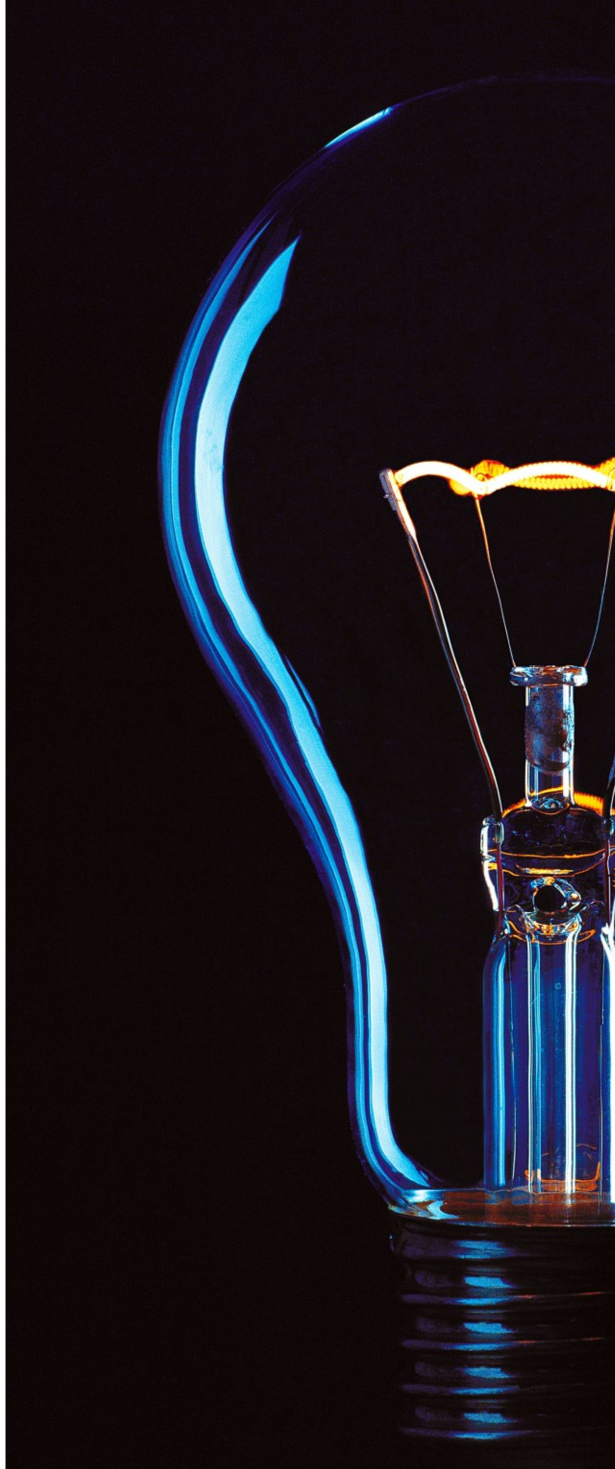
The apparent reason relates to choices taken by the Linux Development community and the commercial company that developed **BitKeeper**, the VCS used by the Linux Development community up to that day.

History of git

The new system would serve goals that were not present in the previous VCS at that time.

They needed **speed**, **simple design, parallel and non-linear development**, **fully distributed** and **able to handle large projects** (such as the Linux Kernel) Taken from git-scm.com

Thus **git** was born and it evolved and matured to be simple, efficient and powerful enough to stand any project and any codebase



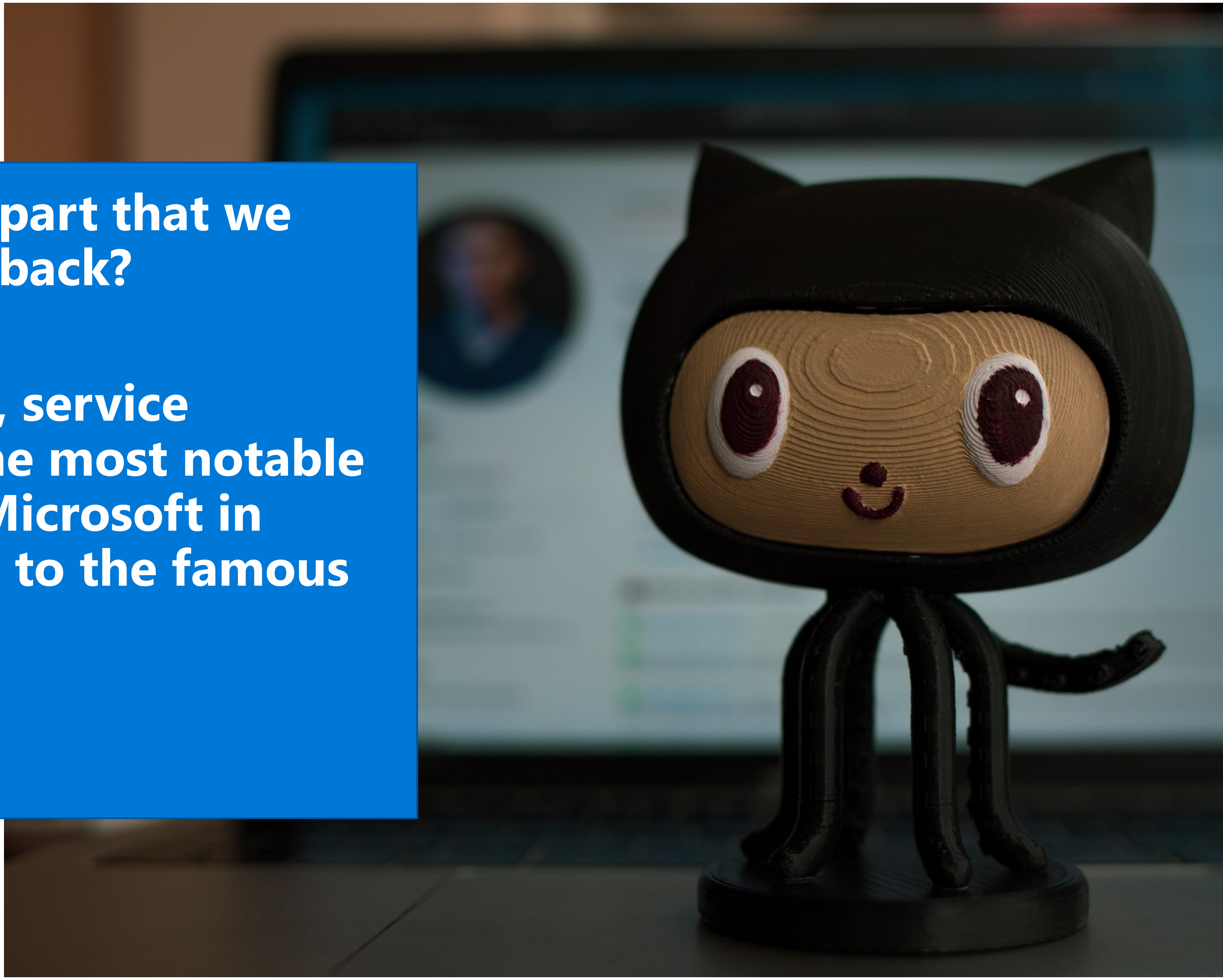
Where to use git

git is, first and foremost, a technology that comes without bias for its usages.

Linux, **MacOS** or **Windows**. You can use it pretty much anywhere, you just need to install it and most of the times it might come already installed for you.

So, if you don't have a specific GUI for the job, all you need to do is open your OS command line interface (terminal, console, cmd or powershell)

Where to use git



But what about the distributed part that we were talking about a few slides back?

That's where, most of the times, service providers come into play and the most notable of them is Github**, acquired by Microsoft in 2018, bringing even more value to the famous phrasing**

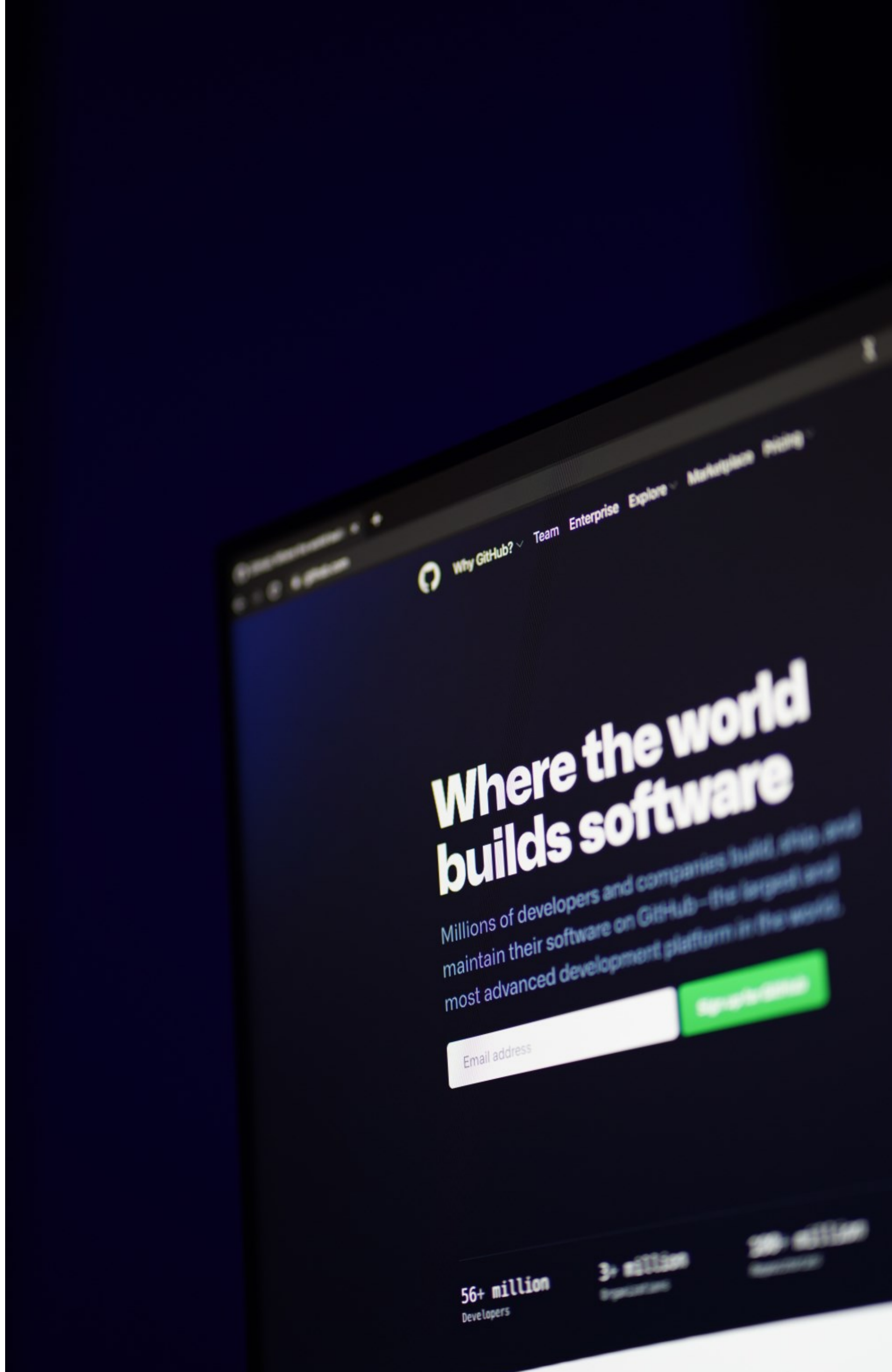
"Microsoft ❤ Linux"

Github, where the world builds software

Although quite the call on **Github's** subtitle, it is by large the most used hosting platform where **open** source (and **closed** source!) code lives and where thousands of thousands of **Developers** collaborate each day, **creating** new technology and **sharing** ideas.

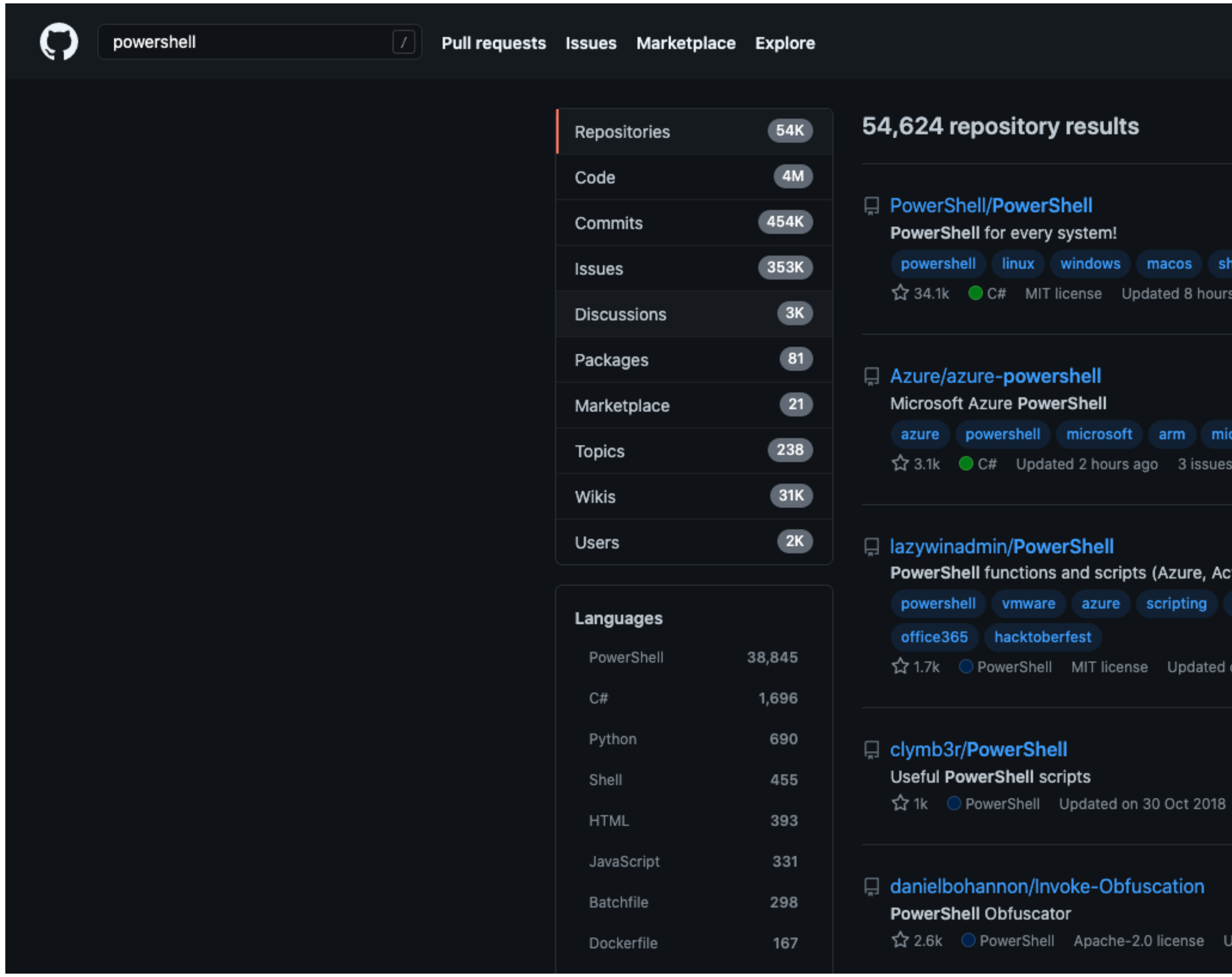
If you want to follow along and you don't have an account yet:

- go to https://github.com/ and **create one**
- add me on Github so we can share experiences (search for **404answernotfound**)


Why GitHub?
Team
Enterprise
Explore
Marketplace
Pricing
Where the world builds software
Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.
Email address
56+ million
Developers

Github is for everyone!

One quick search for just "**powershell**" and look at that!



powershell

Pull requests Issues Marketplace Explore

Repositories	54K
Code	4M
Commits	454K
Issues	353K
Discussions	3K
Packages	81
Marketplace	21
Topics	238
Wikis	31K
Users	2K

Languages

PowerShell	38,845
C#	1,696
Python	690
Shell	455
HTML	393
JavaScript	331
Batchfile	298
Dockerfile	167

54,624 repository results

PowerShell/PowerShell
PowerShell for every system!
powershell linux windows macos sh
34.1k C# MIT license Updated 8 hours

Azure/azure-powershell
Microsoft Azure PowerShell
azure powershell microsoft arm mic
3.1k C# Updated 2 hours ago 3 issues

lazywinadmin/PowerShell
PowerShell functions and scripts (Azure, Ac
powershell vmware azure scripting
office365 hacktoberfest
1.7k PowerShell MIT license Updated

clymb3r/PowerShell
Useful PowerShell scripts
1k PowerShell Updated on 30 Oct 2018

danielbohannon/Invoke-Obfuscation
PowerShell Obfuscator
2.6k PowerShell Apache-2.0 license U

Github is for everyone!

But that's not everything there is to Github!

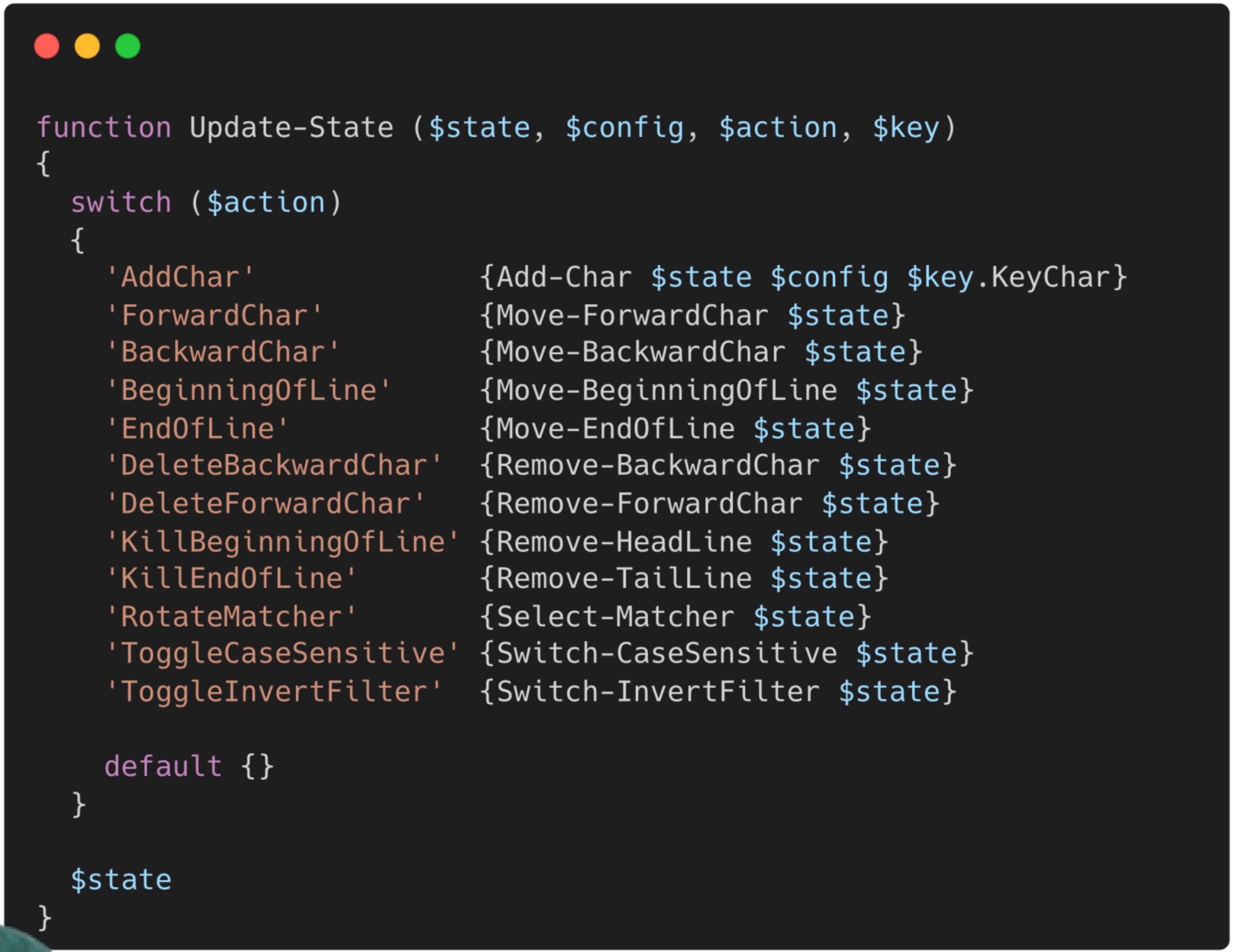
- Completely free for most of its usages and with generous free tiers for Premium usages
- User friendly to a multitude of usages, even for writing books!
- Trusted, well known and powerful system

So, what's in it for me?

Consider for a moment that you are in the midst of developing a Powershell script that's gonna save you from tedious, repetitive tasks

While developing this function you change parameters a few times to the point that you don't remember what you wrote previously. Sounds fair.

Problem is: it was working with previous parameters, now it's not.



```
function Update-State ($state, $config, $action, $key)
{
  switch ($action)
  {
    'AddChar'               {Add-Char $state $config $key.KeyChar}
    'ForwardChar'           {Move-ForwardChar $state}
    'BackwardChar'          {Move-BackwardChar $state}
    'BeginningOfLine'       {Move-BeginningOfLine $state}
    'EndOfLine'             {Move-EndOfLine $state}
    'DeleteBackwardChar'    {Remove-BackwardChar $state}
    'DeleteForwardChar'     {Remove-ForwardChar $state}
    'KillBeginningOfLine'   {Remove-HeadLine $state}
    'KillEndOfLine'         {Remove-TailLine $state}
    'RotateMatcher'         {Select-Matcher $state}
    'ToggleCaseSensitive'   {Switch-CaseSensitive $state}
    'ToggleInvertFilter'    {Switch-InvertFilter $state}

    default {}
  }

  $state
}
```

Example taken from Powershell Poco repository

Broken script, what now?

Lucky you, you already started using git and Github to store your scripts and history of them!

Let's go back a few commits (which are pretty much points in time of your code! Snapshots, if you prefer)

```
# Take me back to that commit!
git checkout a45a21f
```

```
function Update-State (
$state,
$config,
$action,
$key,
$prevParam,
$whatMadeItWorkBeforeParam
) {
  switch ($action)
  {
    'AddChar'             {Add-Char $state $config $
    'ForwardChar'         {Move-ForwardChar $state}
    'BackwardChar'        {Move-BackwardChar $state}
    'BeginningOfLine'     {Move-BeginningOfLine $sta
    'EndOfLine'           {Move-EndOfLine $state}
    'DeleteBackwardChar'  {Remove-BackwardChar $stat
    'DeleteForwardChar'   {Remove-ForwardChar $state
    'KillBeginningOfLine' {Remove-HeadLine $state}
    'KillEndOfLine'       {Remove-TailLine $state}
    'RotateMatcher'       {Select-Matcher $state}
    'ToggleCaseSensitive' {Switch-CaseSensitive $sta
    'ToggleInvertFilter'  {Switch-InvertFilter $stat

    default {}
  }

  $state
}
```

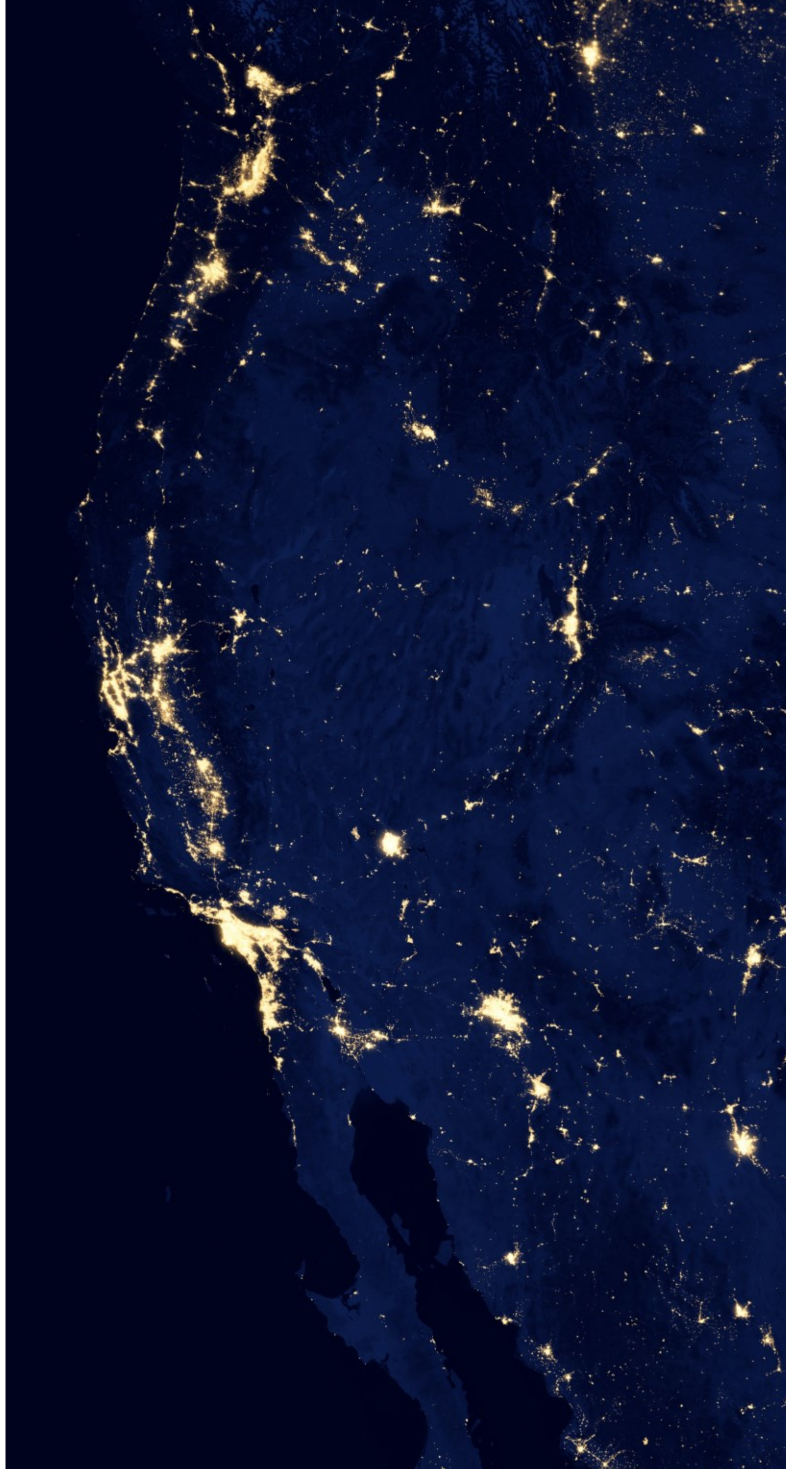
Example taken and modified from Powershell Poco repository

Learning git, one step at a time

But where does all this code resides? Github is on the Internet, so it must be someone's server, a cloud platform. Something like that, right?

Well, Github (along with other similar services like Gitlab or Bitbucket) is a provider of remote origins

A remote origin is the place on the Internet where your "git stuff" lives

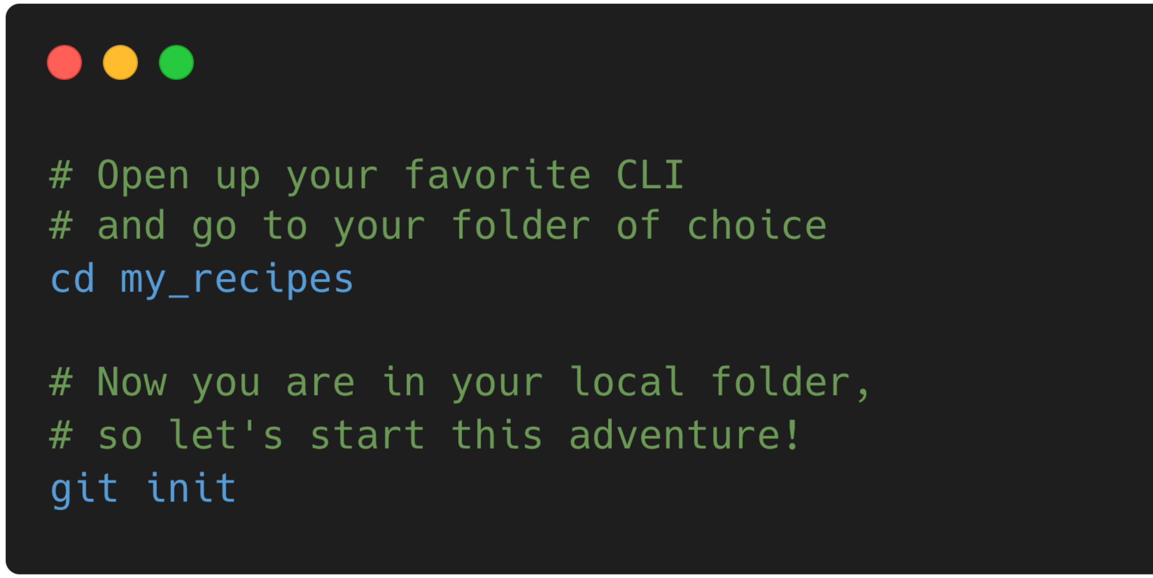


Learning git, one step at a time

So where's your code?

Well, your code (or configurations, or files, or pictures or books for that matter) **is still stored locally** on your computer until you create a remote repository for it so let's go for a little story to learn how to go from zero to hero with git!

Wanna follow along? :)



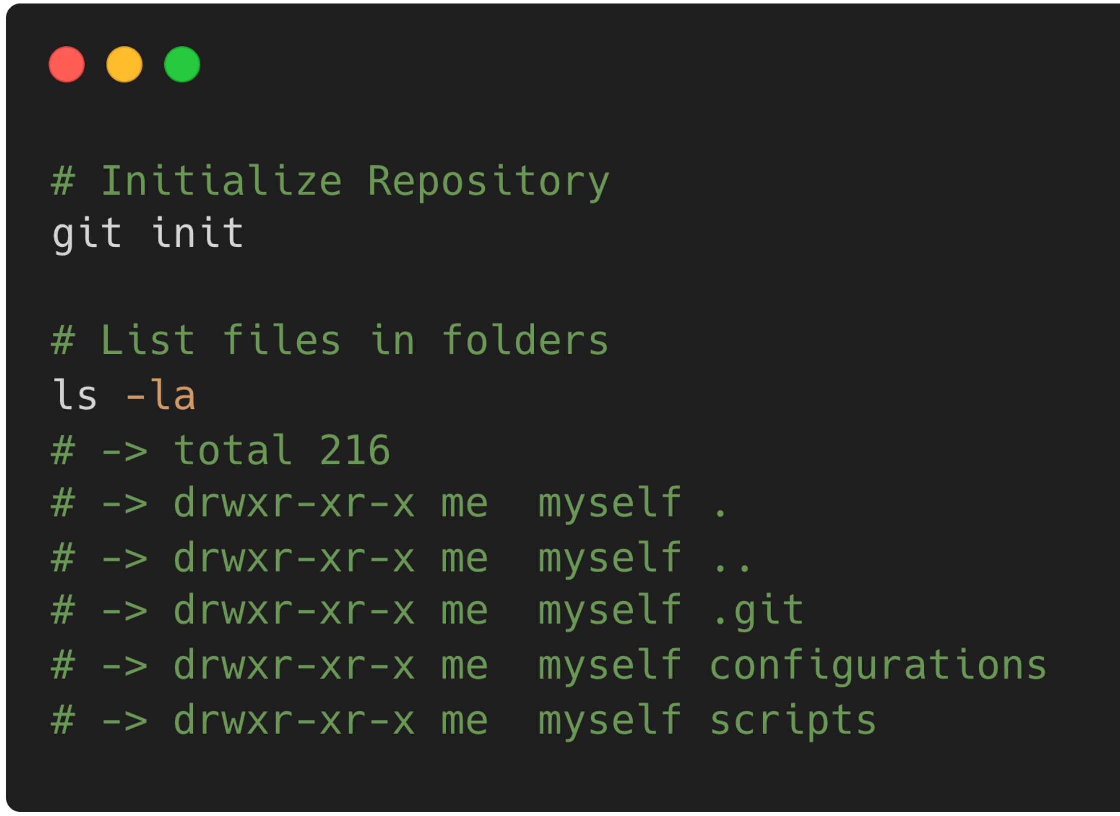
```
# Open up your favorite CLI
# and go to your folder of choice
cd my_recipes

# Now you are in your local folder,
# so let's start this adventure!
git init
```

Learning git, init

What did we just do?!

We started our journey, that's what! The command git init is the startup command to tell *git (the software)* that the folder you are in has to become a git repository

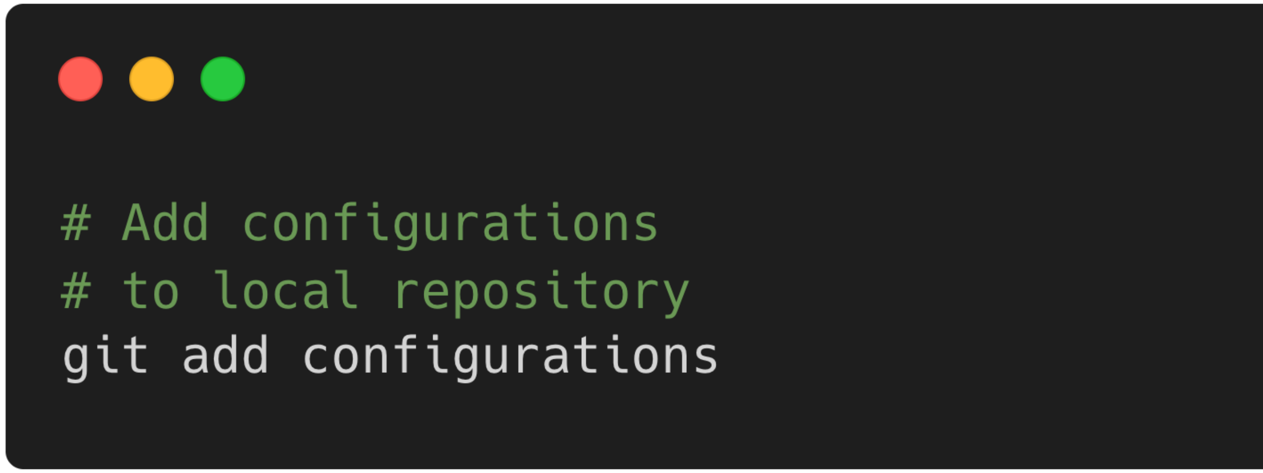
A repository is a workspace (bunch of files and folders) that are "watched" by git and on init, git creates a .git folder with all the necessary files it needs to work!



```
# Initialize Repository
git init

# List files in folders
ls -la
# -> total 216
# -> drwxr-xr-x me  myself .
# -> drwxr-xr-x me  myself ..
# -> drwxr-xr-x me  myself .git
# -> drwxr-xr-x me  myself configurations
# -> drwxr-xr-x me  myself scripts
```

Learning git, add a file

So, we now have a local repository but there is nothing to it. Our files and folders (configurations and scripts, in the example before) are just as they were before. They are inside a repository but they are not added to the repository, which is the explicit way to tell git to look out for changes on those files



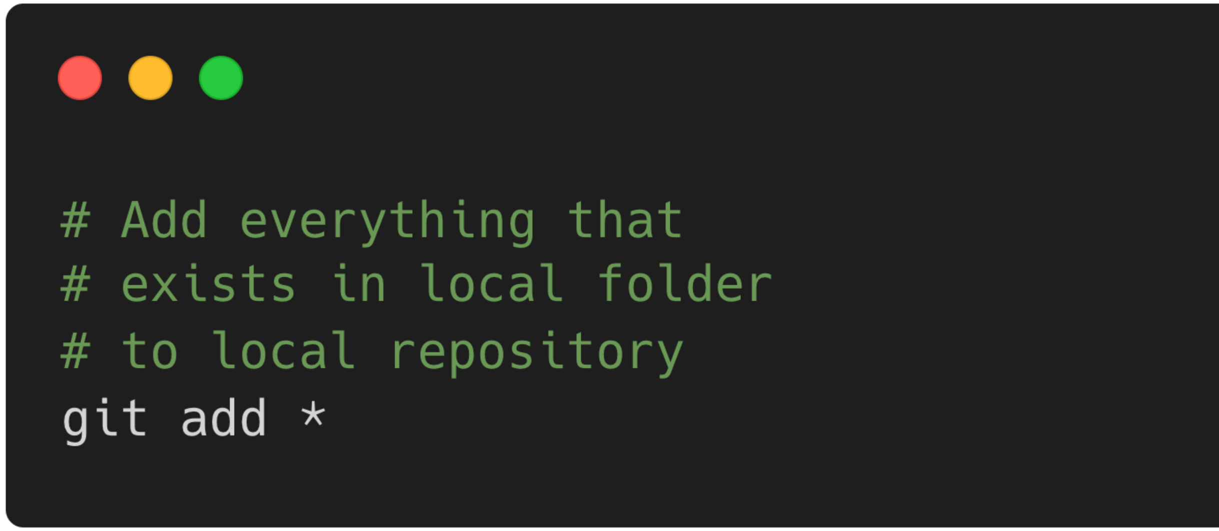
```
# Add configurations
# to local repository
git add configurations
```

Let's tell git to add our configurations to our local repository

Learning git, add all files and folders

Adding one file or folder at a time can be quite tedious, can't it.

Let's just add all existing files and folders from our local folder to our local repository and just forget about it

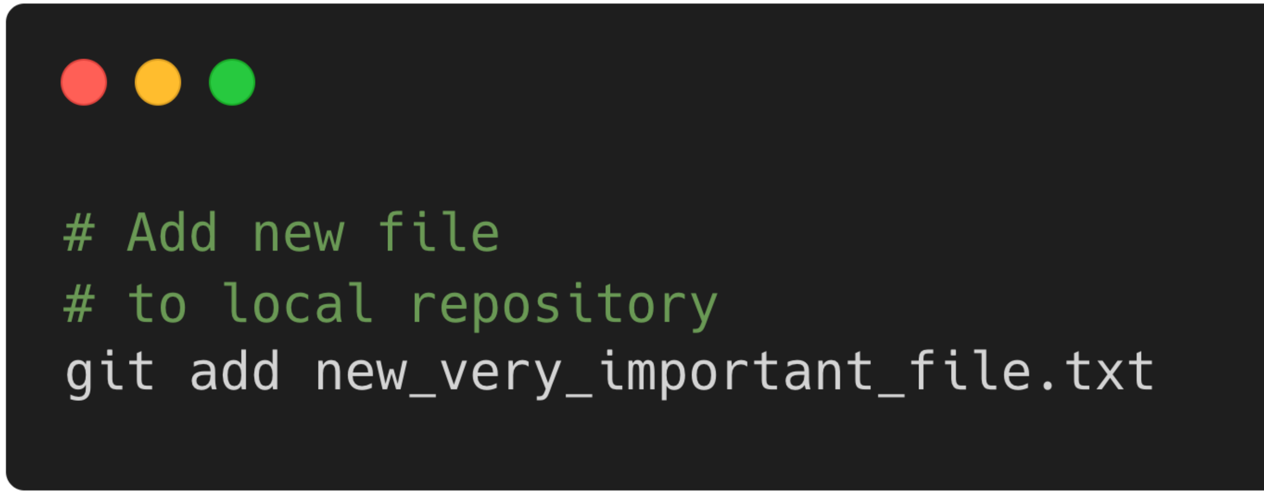


```
# Add everything that
# exists in local folder
# to local repository
git add *
```

Learning git, watch for changes

We added everything we had on our local folder to our local repository. Great.

Oh wait, we created a new file in the meantime. Was it automatically added? It wasn't. We have to take care of that ourselves.



```
# Add new file
# to local repository
git add new_very_important_file.txt
```

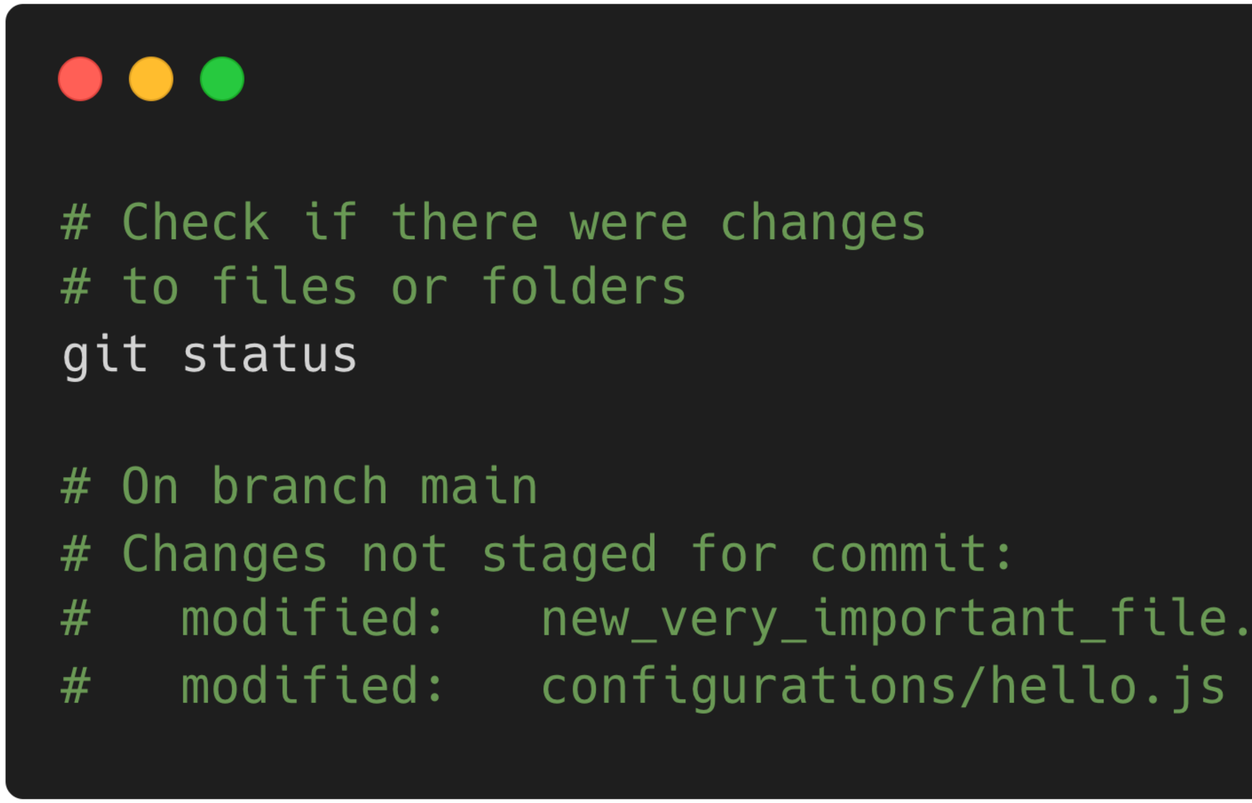
Learning git, watch for changes

Nice. We added all the files that exist in our local folder and they are being watched by our git installation.

What's important to know here is: we told git to add all the files and folders to its watchlist.

What does it mean? Every change to files or folders will be captured by git.

To know which changes happened, just type git status



```
# Check if there were changes
# to files or folders
git status

# On branch main
# Changes not staged for commit:
#   modified:   new_very_important_file.
#   modified:   configurations/hello.js
```

Learning git, committing your changes

You built your scripts and configurations, created a local git repository and added files and folders to it.

You checkout out the status of your local repository to find out that two files need to be added again to the stage. What's the stage though?

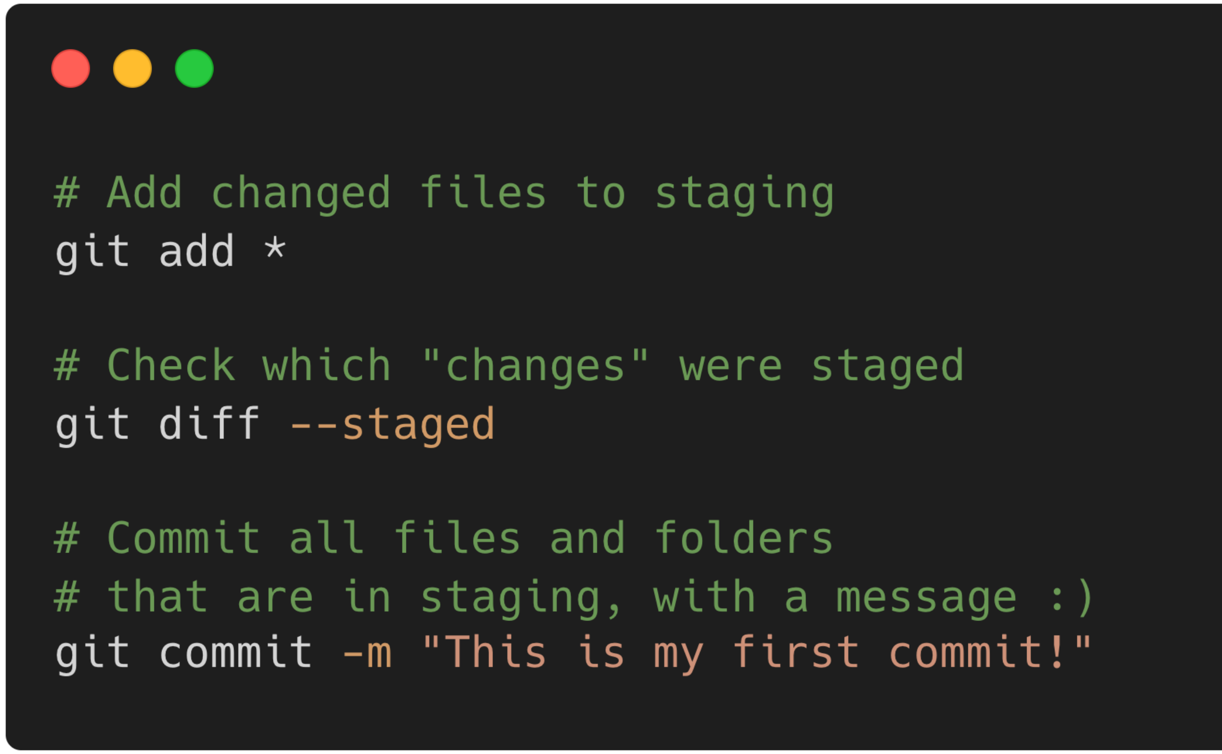
The stage, or better Staging, is a space where git handles all your changes to files and folders. Changes that are staged can be committed!

Learning git, committing

Great job! You added your latest changes to your files and they now "live" in the staging zone.

Let's commit them by creating a commit.

A commit can be thought of as a point in time of your code, a history pin of your changes, a snapshot of your project.



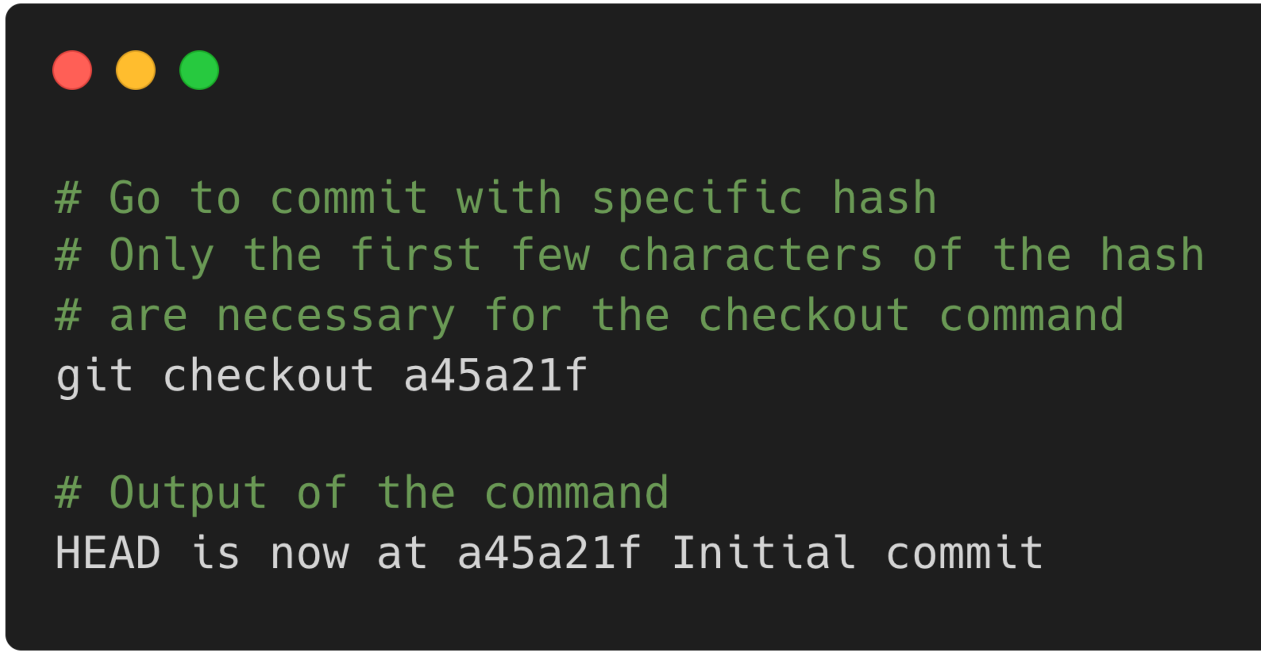
```
# Add changed files to staging
git add *

# Check which "changes" were staged
git diff --staged

# Commit all files and folders
# that are in staging, with a message :)
git commit -m "This is my first commit!"
```

Learning git, back in time

Alright. You committed your changes, created points in time of your project and all that remains is to know how to move back and forth in the timeline of your project!

A commit creates a commit hash, which is the snapshot id we will be using to move around the timeline

We can see the commit hash, author, date of the commit and also the message of the commit



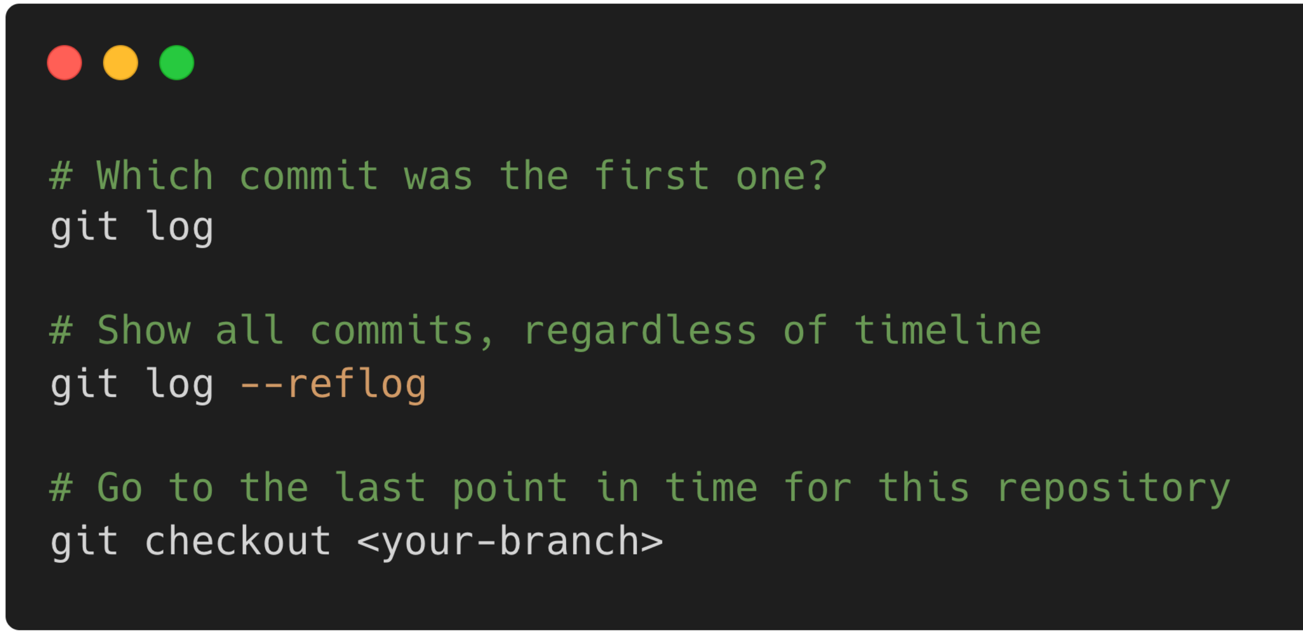
```
# Go to commit with specific hash
# Only the first few characters of the hash
# are necessary for the checkout command
git checkout a45a21f

# Output of the command
HEAD is now at a45a21f Initial commit
```

Learning git, and back to the present

Going back and forth can be tiresome so we have a really simple command to go back to the last point in time we committed.

Depending on the name of your local repository branch (which usually starts with master but should be changed to main) you can do:

git checkout master or git checkout main



```
# Which commit was the first one?
git log

# Show all commits, regardless of timeline
git log --reflog

# Go to the last point in time for this repository
git checkout <your-branch>
```

If you tried to "git log" while *back in time* you probably noticed that all commits "after" that point were gone. Add the –reflog flag to show all commits!

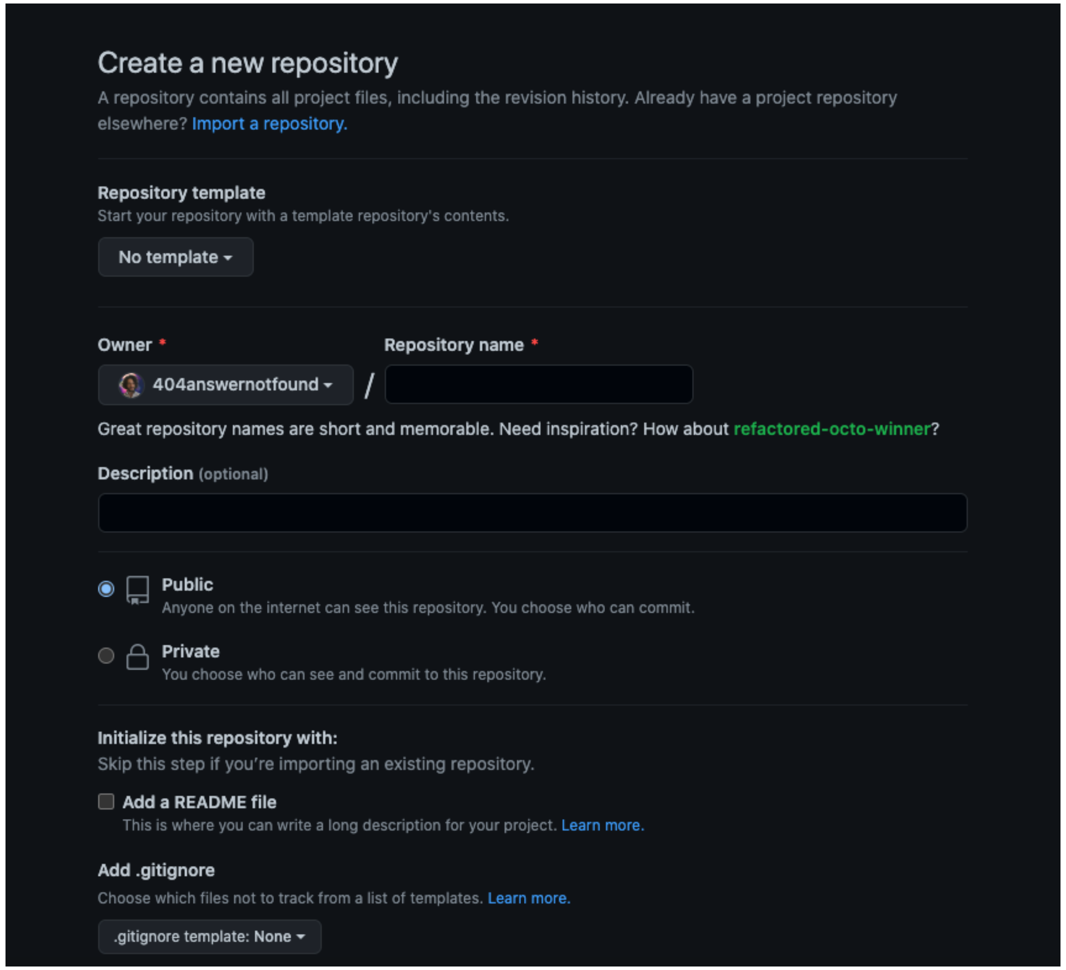
Learning git, pushing our changes

Changes were committed, points in time were seen so all that's left is to learn where's the distributed part in git!

To do that, we need to push our code, but push where?!

Well, we need to create a remote repository which will function as our remote origin. Lots of words, let's just create it!

Go to https://github.com/new



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Repository template

Start your repository with a template repository's contents.

No template ▾

Owner * **Repository name** *

404answernotfound ▾ /

Great repository names are short and memorable. Need inspiration? How about refactored-octo-winner?

Description (optional)

● **Public**
Anyone on the internet can see this repository. You choose who can commit.

○ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. Learn more.

Add .gitignore
Choose which files not to track from a list of templates. Learn more.

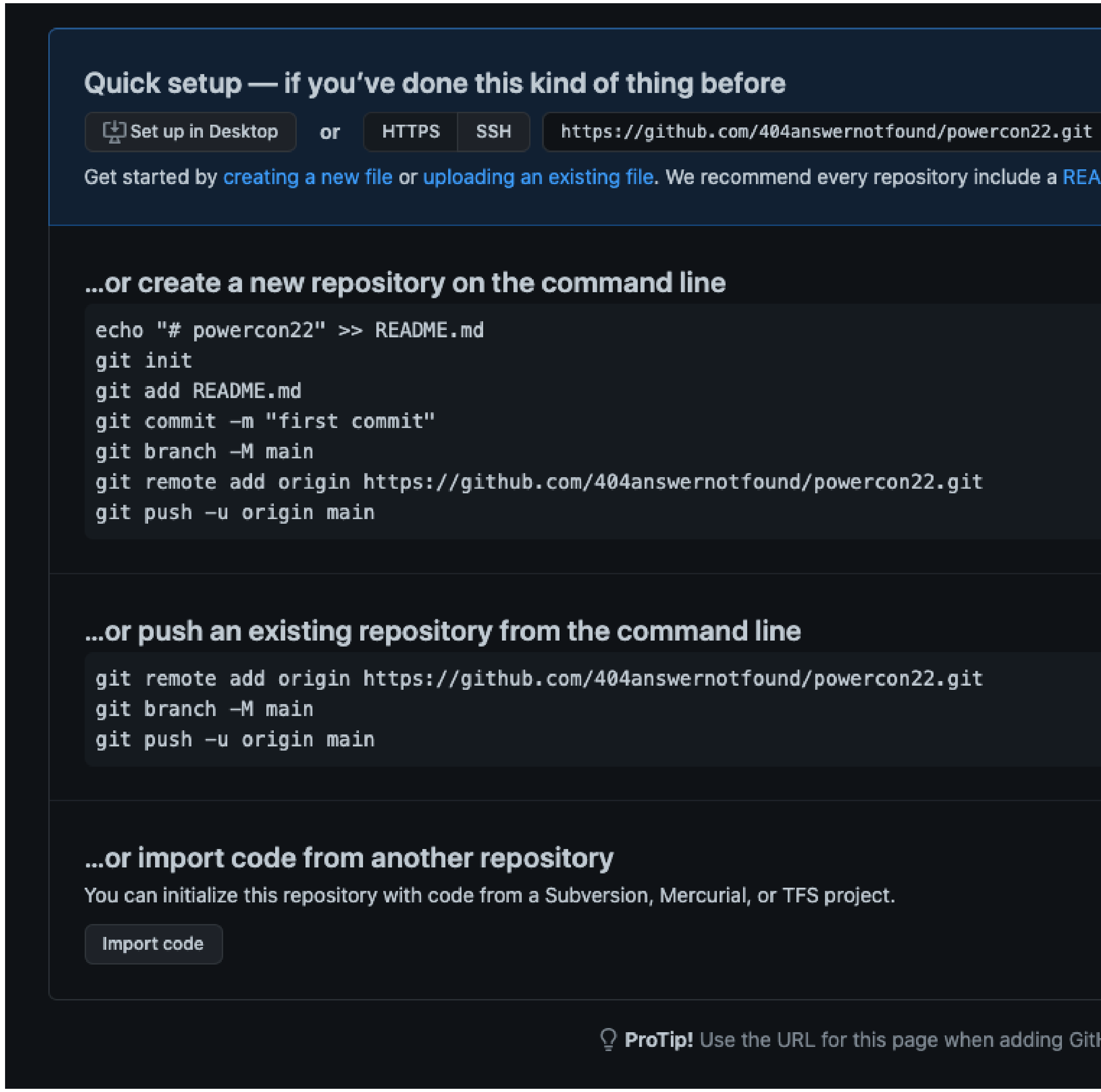
.gitignore template: None ▾

Learning git, remote repo

Add the necessary data, make it public or private (which is the access that your repository will have) and follow the instructions given on the next page

As you can see, we are also invited to change our primary branch to main

The important part there is also the remote add origin



Quick setup — if you've done this kind of thing before

```
https://github.com/404answernotfound/powercon22.git
```

Get started by creating a new file or uploading an existing file. We recommend every repository include a REA

...or create a new repository on the command line

```
echo "# powercon22" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/404answernotfound/powercon22.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/404answernotfound/powercon22.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

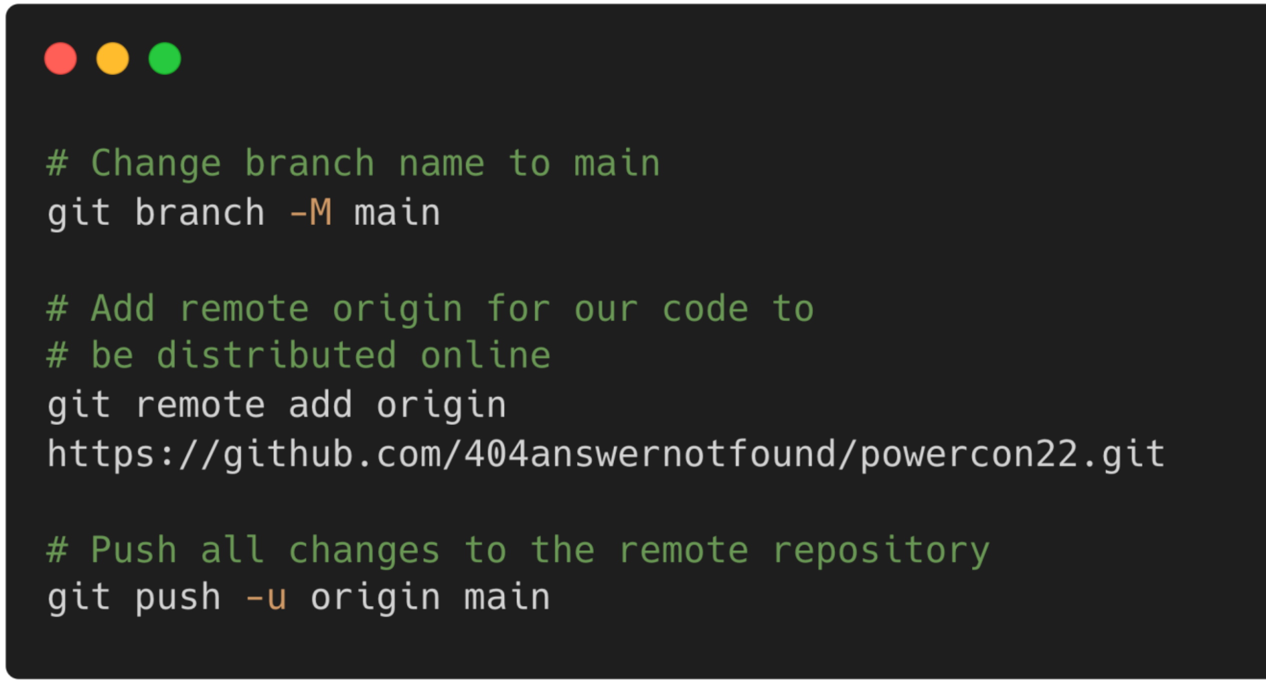
ProTip! Use the URL for this page when adding Gitl

Learning git, remote repo

Great job there, creating your remote origin to push your code online!

If you were to go to the url you just pushed your code to, you will find all your files and folders that were added, staged, commited and lastly pushed to your now remote repository on **Github**!

Congratulations! Your code is fully distributed!

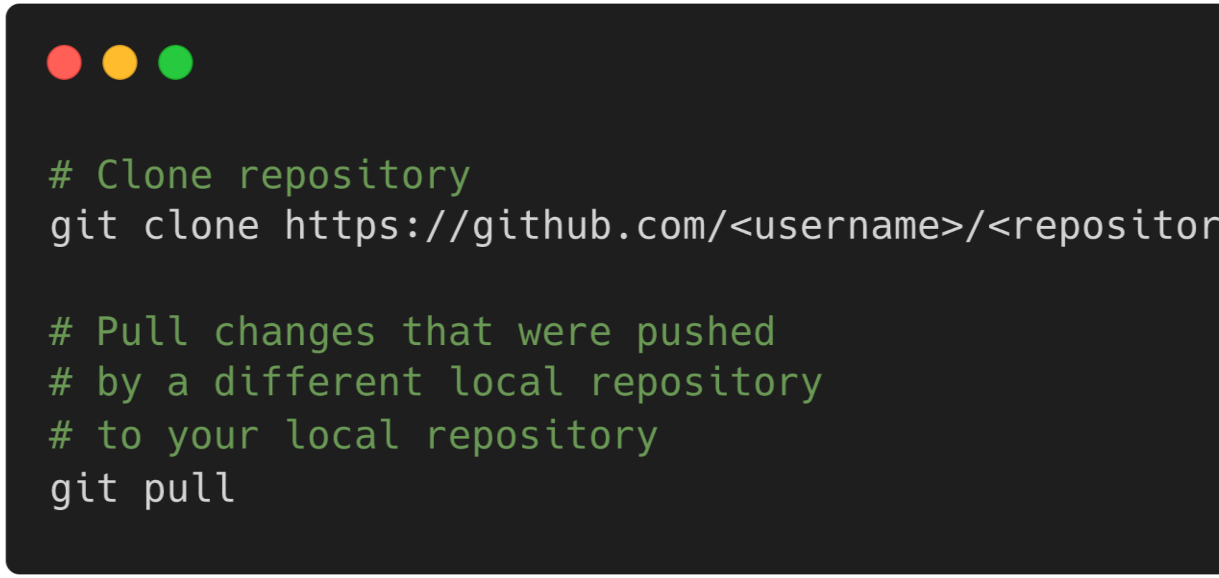


```
# Change branch name to main
git branch -M main

# Add remote origin for our code to
# be distributed online
git remote add origin
https://github.com/404answernotfound/powercon22.git

# Push all changes to the remote repository
git push -u origin main
```

Learning git, collaborations

Now that your code is distributed your friends can finally help you out with your scripts and configurations.

If any of them wanted to get your repository, they would have to clone it and if they pushed changes to that repository, you would have to pull the changes and merge them into your local repository. There might be a bit more to it, but let's leave it at this for now!

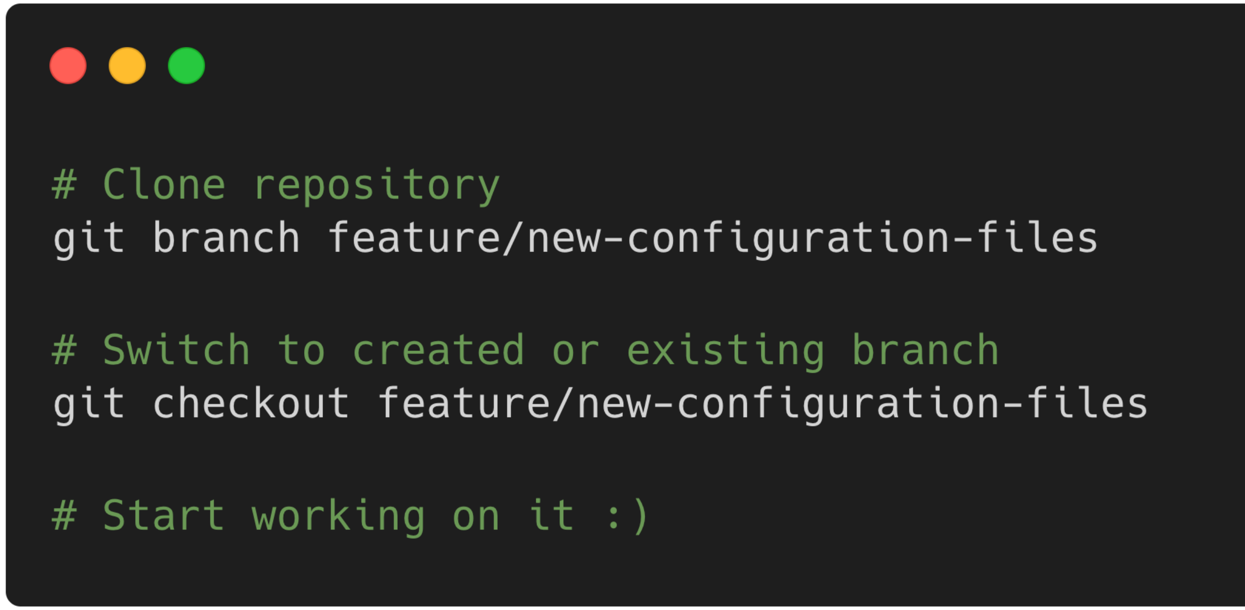


```
# Clone repository
git clone https://github.com/<username>/<repositor

# Pull changes that were pushed
# by a different local repository
# to your local repository
git pull
```

Learning git, branching

Collaborations can become messy, everyone should work on their own feature branch and that's exactly what we should do!

To keep our code clean and less error prone, especially when more people are working on it, it's good habit to create branches based on utility (feature, bugfix, hotfix, release)



```
# Clone repository
git branch feature/new-configuration-files

# Switch to created or existing branch
git checkout feature/new-configuration-files

# Start working on it :)
```

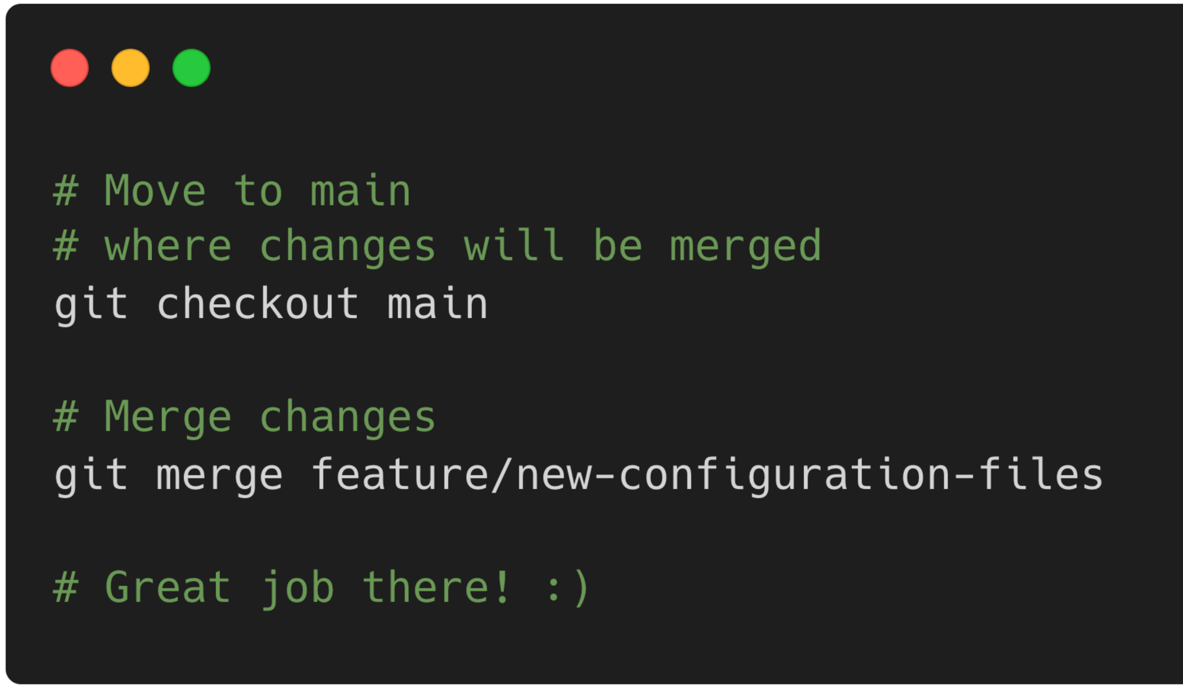
Learning git, merging

Sometimes you want to bring changes from one branch to another, which could be the primary branch (remember main?)

In that case, you want to merge branches

git checkout main
git merge <name of branch>

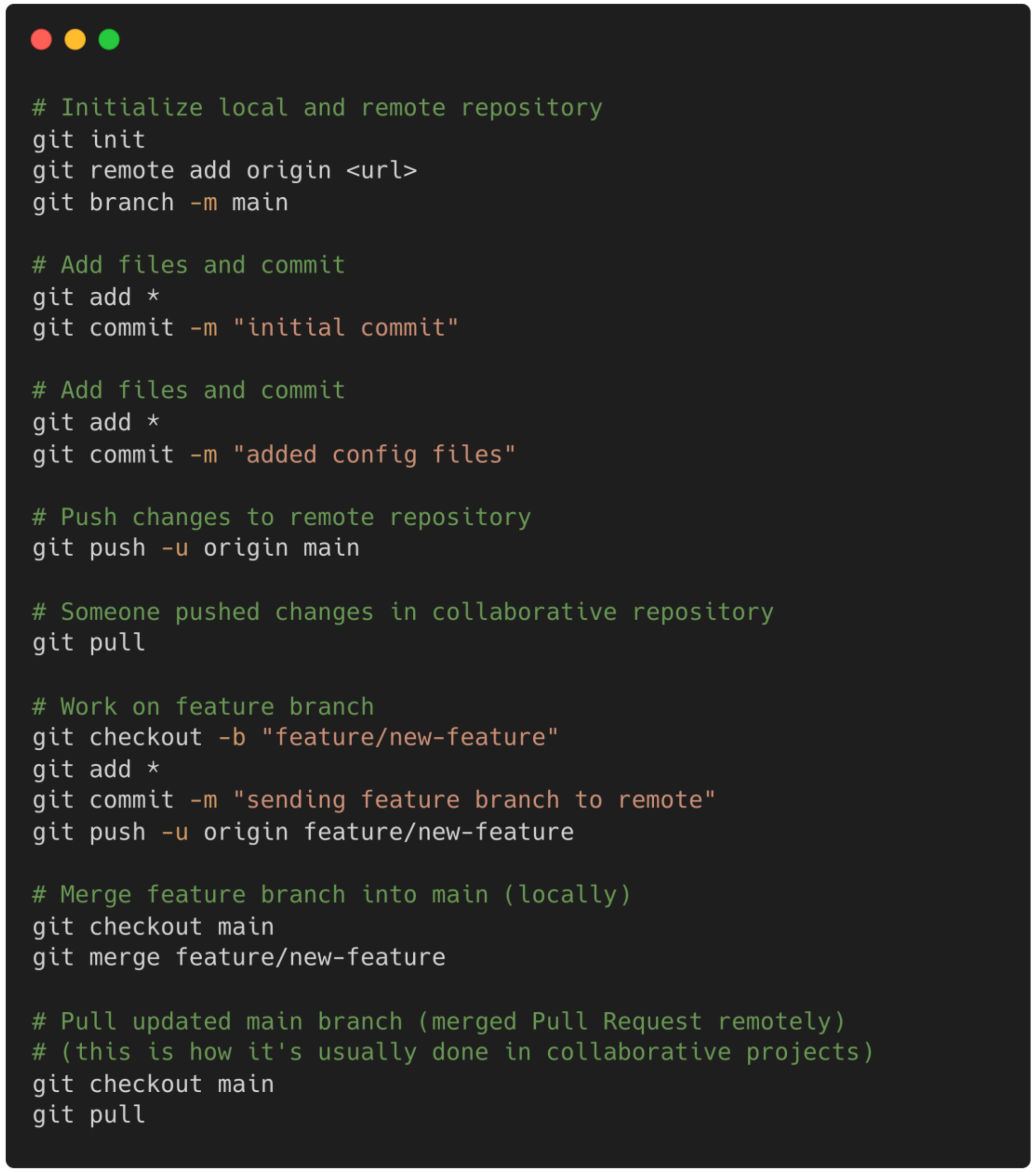
If there are no conflicts between files and folders, merging will be automatic



```
# Move to main
# where changes will be merged
git checkout main

# Merge changes
git merge feature/new-configuration-files

# Great job there! :)
```

Example use case

This workflow highly depends on your choices or your team's choices but it can be made simpler or more detailed! You choose!



```
# Initialize local and remote repository
git init
git remote add origin <url>
git branch -m main

# Add files and commit
git add *
git commit -m "initial commit"

# Add files and commit
git add *
git commit -m "added config files"

# Push changes to remote repository
git push -u origin main

# Someone pushed changes in collaborative repository
git pull

# Work on feature branch
git checkout -b "feature/new-feature"
git add *
git commit -m "sending feature branch to remote"
git push -u origin feature/new-feature

# Merge feature branch into main (locally)
git checkout main
git merge feature/new-feature

# Pull updated main branch (merged Pull Request remotely)
# (this is how it's usually done in collaborative projects)
git checkout main
git pull
```

No more final-final-reallyfinal files

Now that you know quite a bit of git you can finally say goodbye to weird file naming!

Just create a new branch, make a commit, do what you think might be the best way to handle your files!

Where to go from here

There are so many more commands and features of git (and Github) left to learn!

You should check out:

- Github Actions
- Git Workflow
- Git the simple guide
- Oh shit git!

Bing, Google, Ecosia, DuckDuckGo. Pick your favorite Search Engine and look for great learning material!

Where to find me

ICTPower blog https://www.ictpower.it/

My personal blog https://404answernotfound.eu/

Medium @404answernotfound

Spotify 404AnswerNotFound Podcast

@404answnotfound

lorenzopieri

Thank you

Lorenzo Pieri
Senior Software Engineer
404answernotfound.public@gmail.com

@404answnotfound
lorenzopieri