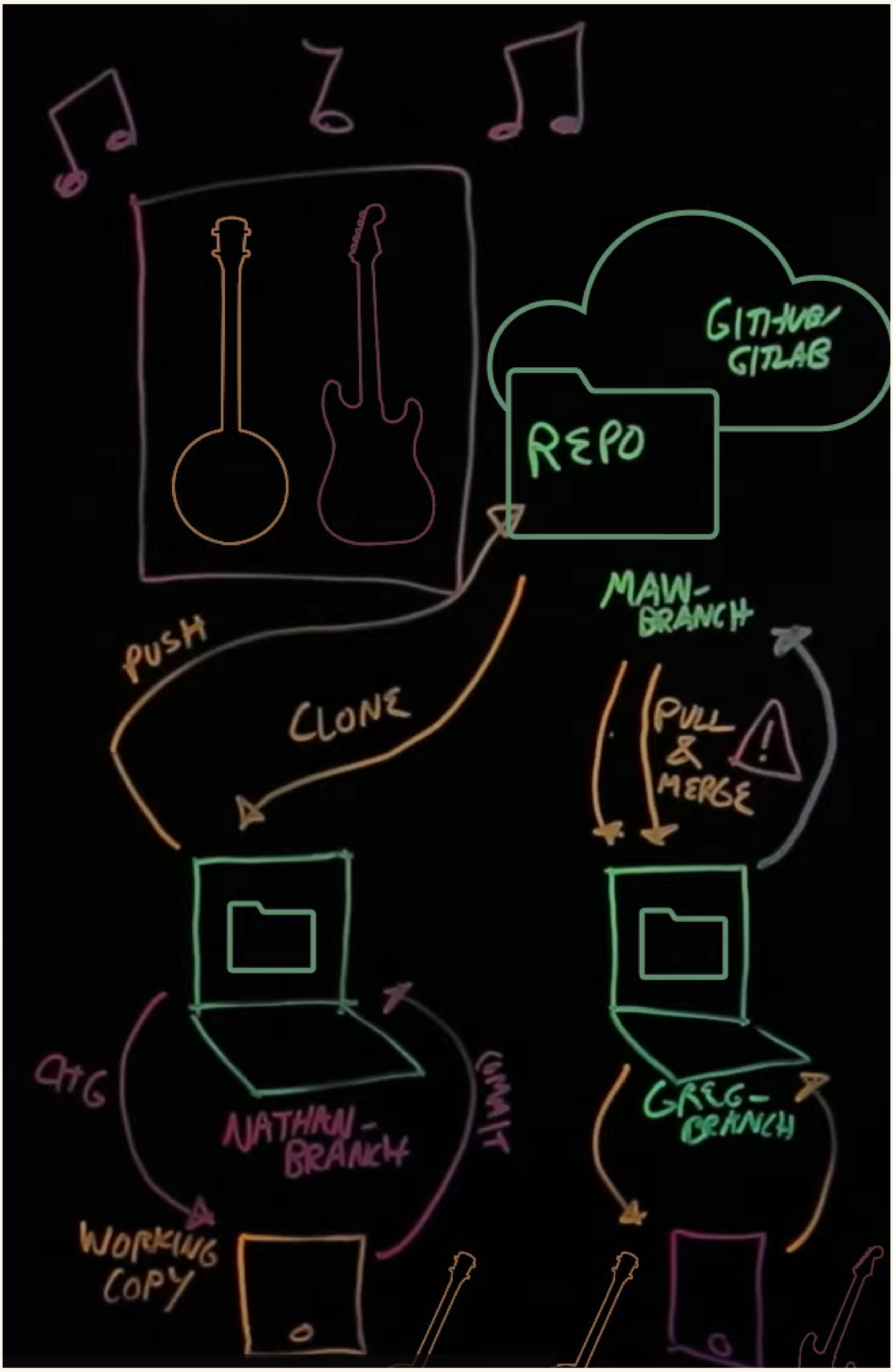
GITHUB

Everything about GitHub and Git

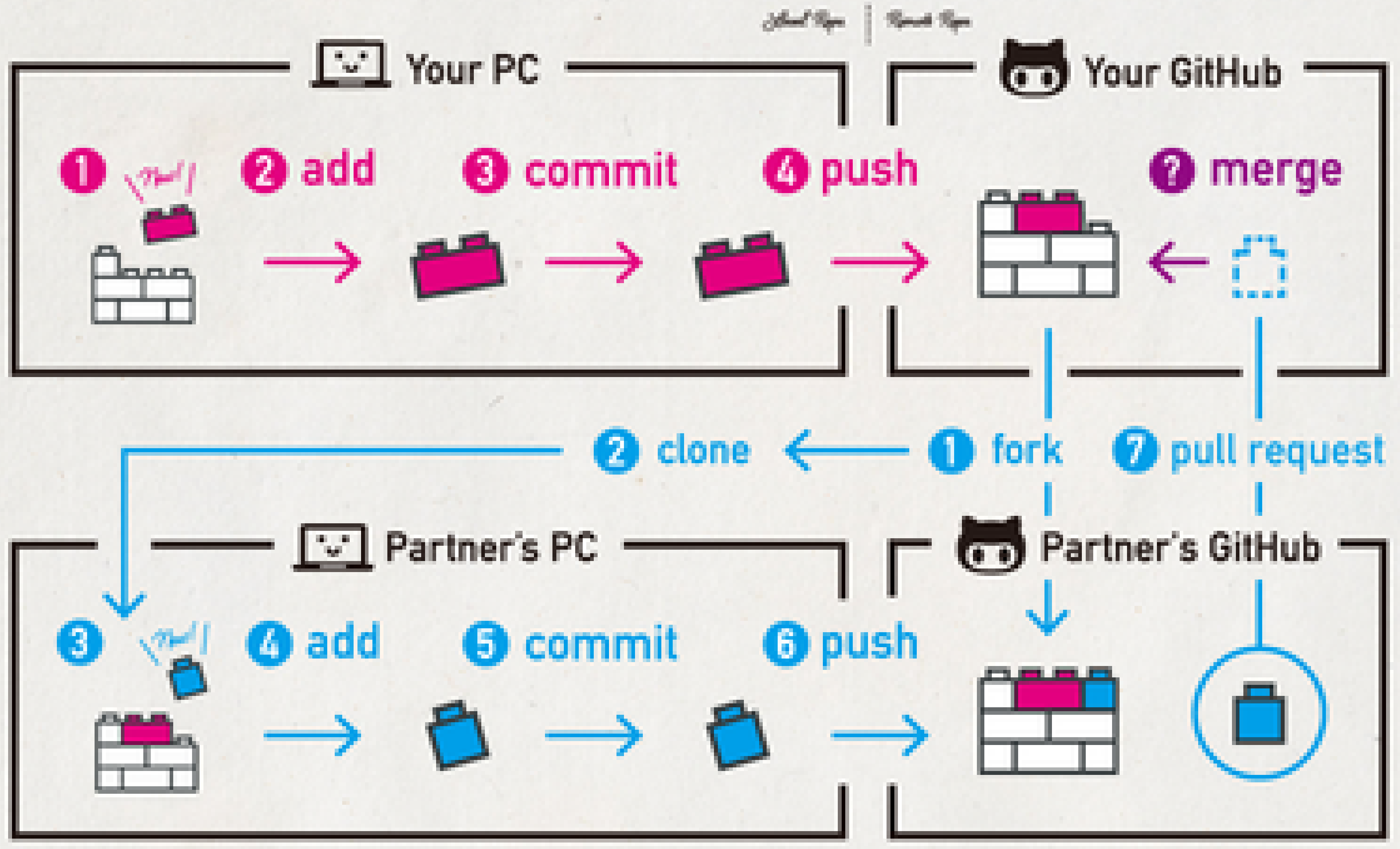
Introduction to Git

Let's say, you are working in a media company, and your team has been assigned the task of designing and developing a music player app. So, in such a situation many developers have to work on the same project.
Git, in short, provides the following benefits:

- Work collaboration + team development
- Version control system
- Track changes - who changed what, added what, etc.
- Historical backup - get different versions, revert to previous versions, etc.
- Flexible - locally or GitHub cloud - DevOps CI/CD
- Interaction using CLI
- Trunk-based development - tree trunk - main -- other branches


GITHUB/
GITLAB
REPO
MAIN-
BRANCH
PUSH
CLONE
PULL
&
MERGE
CHG
NATHAN-
BRANCH
COMMIT
GREG-
BRANCH
WORKING
COPY

Git / GitHub How to "Pull Request"



designless.net

Differences between Git and GitHub

• Git is a software, and a Command Line Tool • Git is maintained by Linux • Git is installed locally on the system	• GitHub is an online hosting service, and GitHub is a graphical user interface • GitHub is maintained by Microsoft • GitHub is hosted on the web
• Git provides a Desktop interface named Git Gui and Git Bash • Git is focused on version control and code sharing • Git is open-source licensed • Git does not have much tool integration.	• GitHub provides a Desktop interface named GitHub Desktop. • GitHub is focused on centralized source code hosting. • GitHub is a hosting service for Git repositories. • GitHub includes a free-tier and pay-for-use tier. • GitHub has an active marketplace for tool integration.

Git - in depth

Git and GitHub are two technologies that every developer should learn, irrespective of their field. If you're a beginner developer, you might think that these two terms mean the same thing – but they're different!

- Git is a version control system which lets you **track changes** you make to your files over time. With Git, you can **revert to various states of your files** (like a time traveling machine). You can also **make a copy of your file**, **make changes** to that copy, and then **merge these changes** to the original copy.
- For example, you could be working on a website's landing page and discover that you do not like the navigation bar. But at the same time, you might not want to start altering its components because it might get worse. With Git, you can create an identical copy of that file and play around with the navigation bar. Then, when you are satisfied with your changes, you can merge the copy to the original file.
- You are not limited to using Git just for **source code files** – you can also use it to keep track of text **files or even images**. This means that Git is not just for developers – anyone can find it helpful.
- In order to use Git, you have to install it on your computer. https://git-scm.com/downloads. To verify that the Git is installed properly, you can run this command on the command line:

  git --version

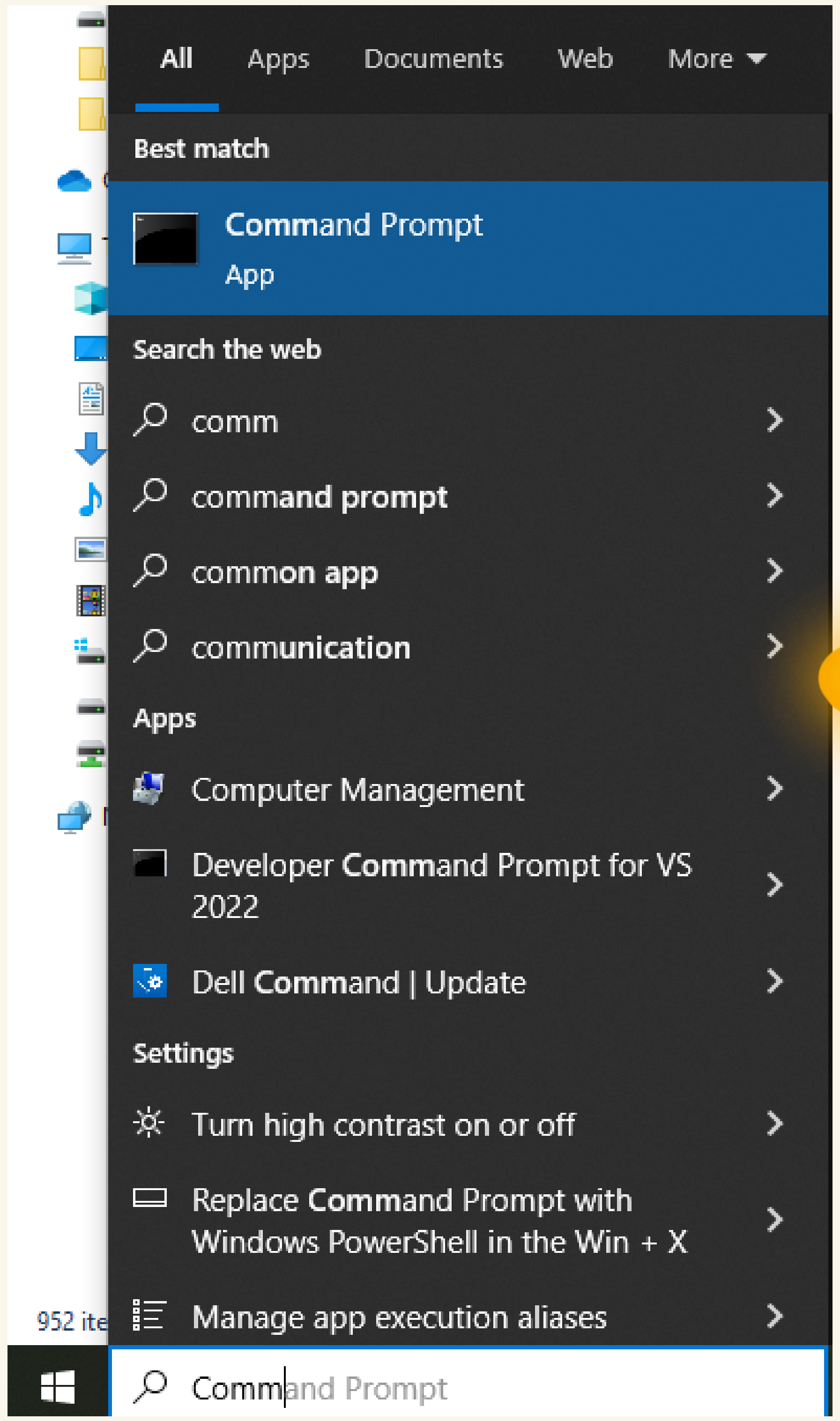
  This shows you the current version installed on you PC.

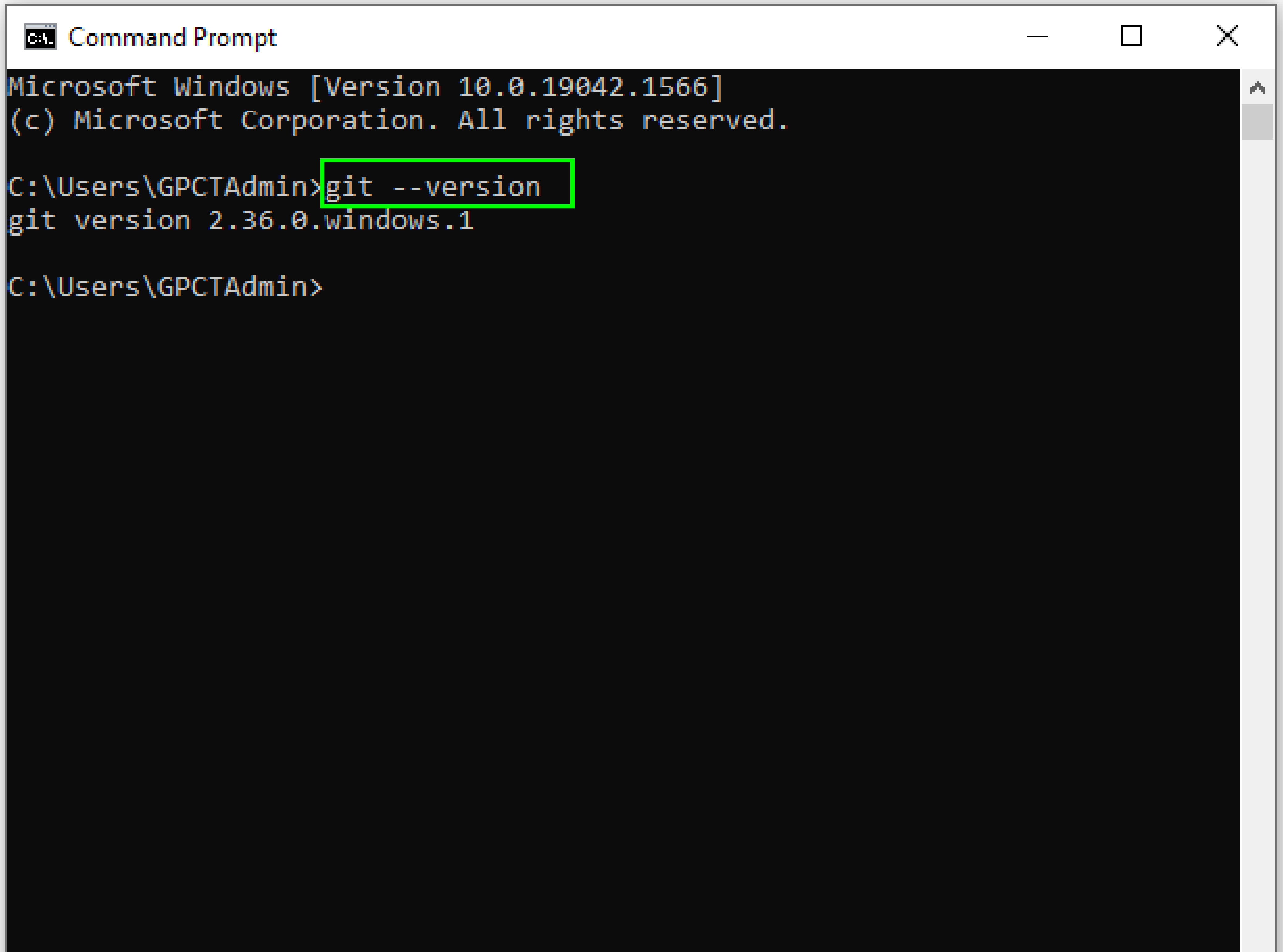
We are finally done with installing and setting up Git.

How to invoke the Git software

1. You can use any terminal (CLI. Command Prompt, Git Bash, etc.)
2. Check the current installed version of Git:

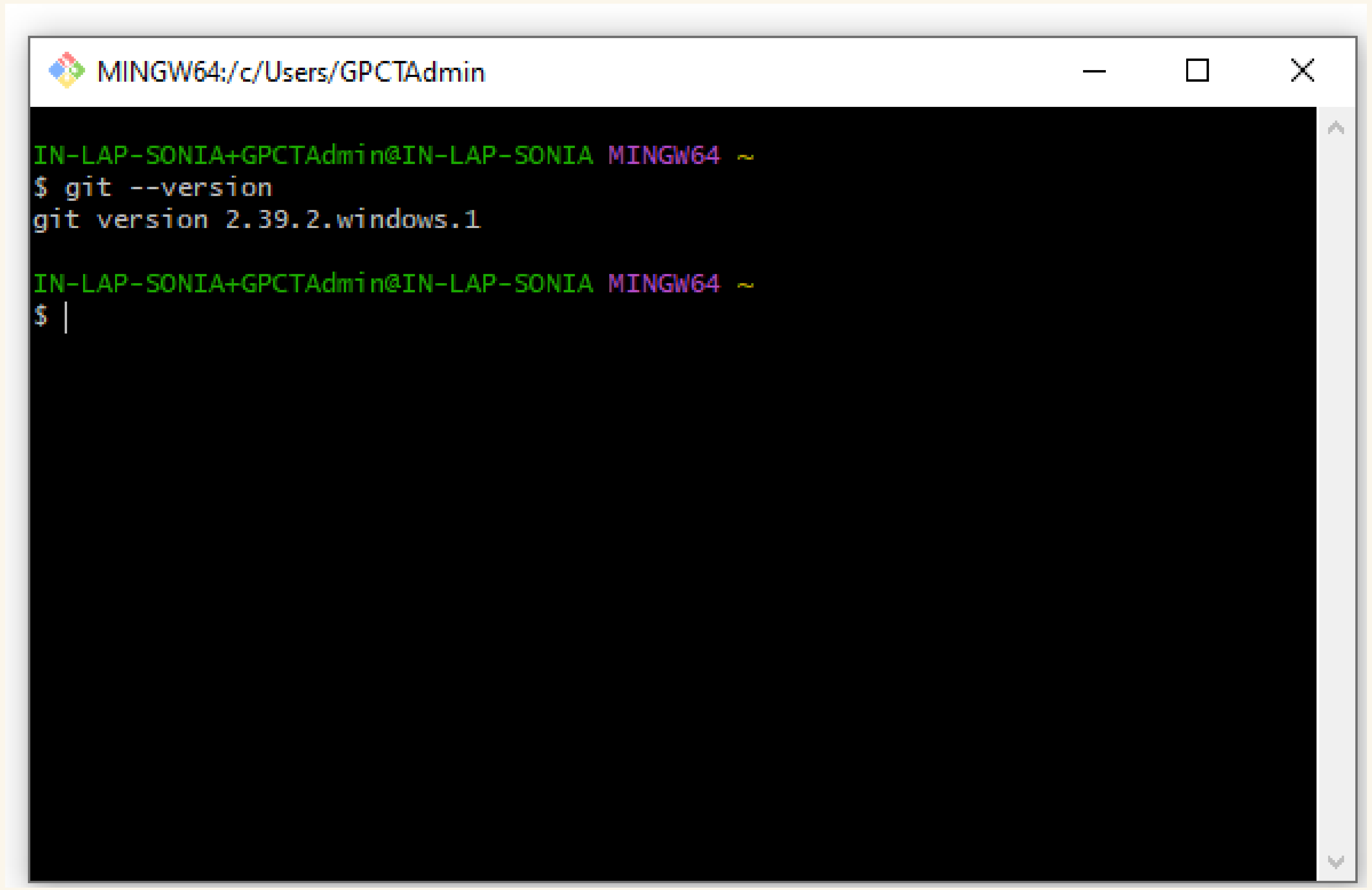
Run the following command on the command line/command prompt:
git --version. (Type command prompt in the start menu)





```
Microsoft Windows [Version 10.0.19042.1566]
(c) Microsoft Corporation. All rights reserved.

C:\Users\GPCTAdmin>git --version
git version 2.36.0.windows.1

C:\Users\GPCTAdmin>
```

Finding Git version using Git Bash



```
IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 ~
$ git --version
git version 2.39.2.windows.1

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 ~
$ |
```

Trivia - just for me

GitBash uses this formula, which is inbuilt:

```
system_name+admin_name@system_name MINGW64
```

That is, for example:

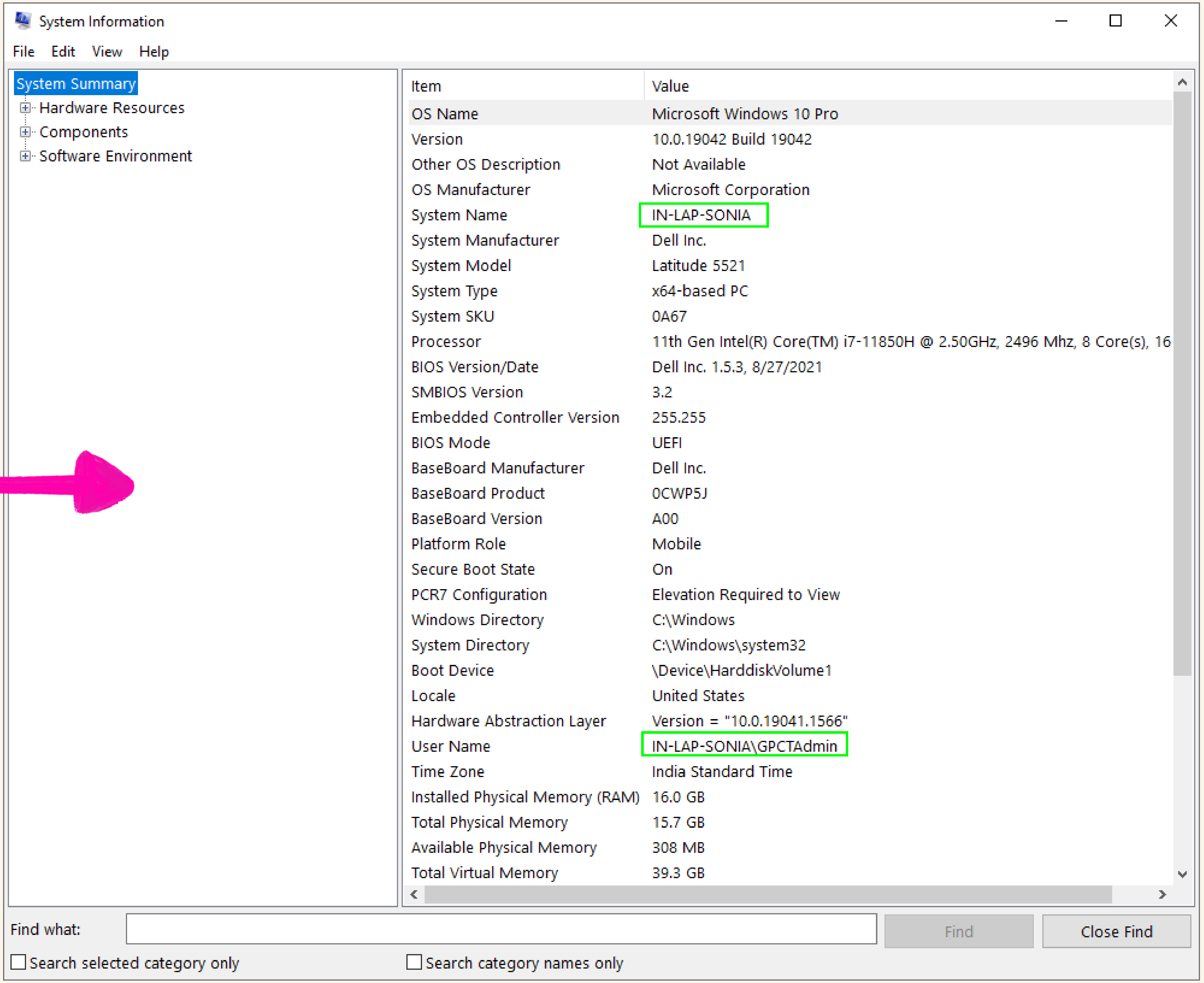
```
IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 ~
```

Every git command is written below this above mentioned line

```
IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 ~
$ git config --global user.name "Sonia Mathew"
```

(Just for me :D)

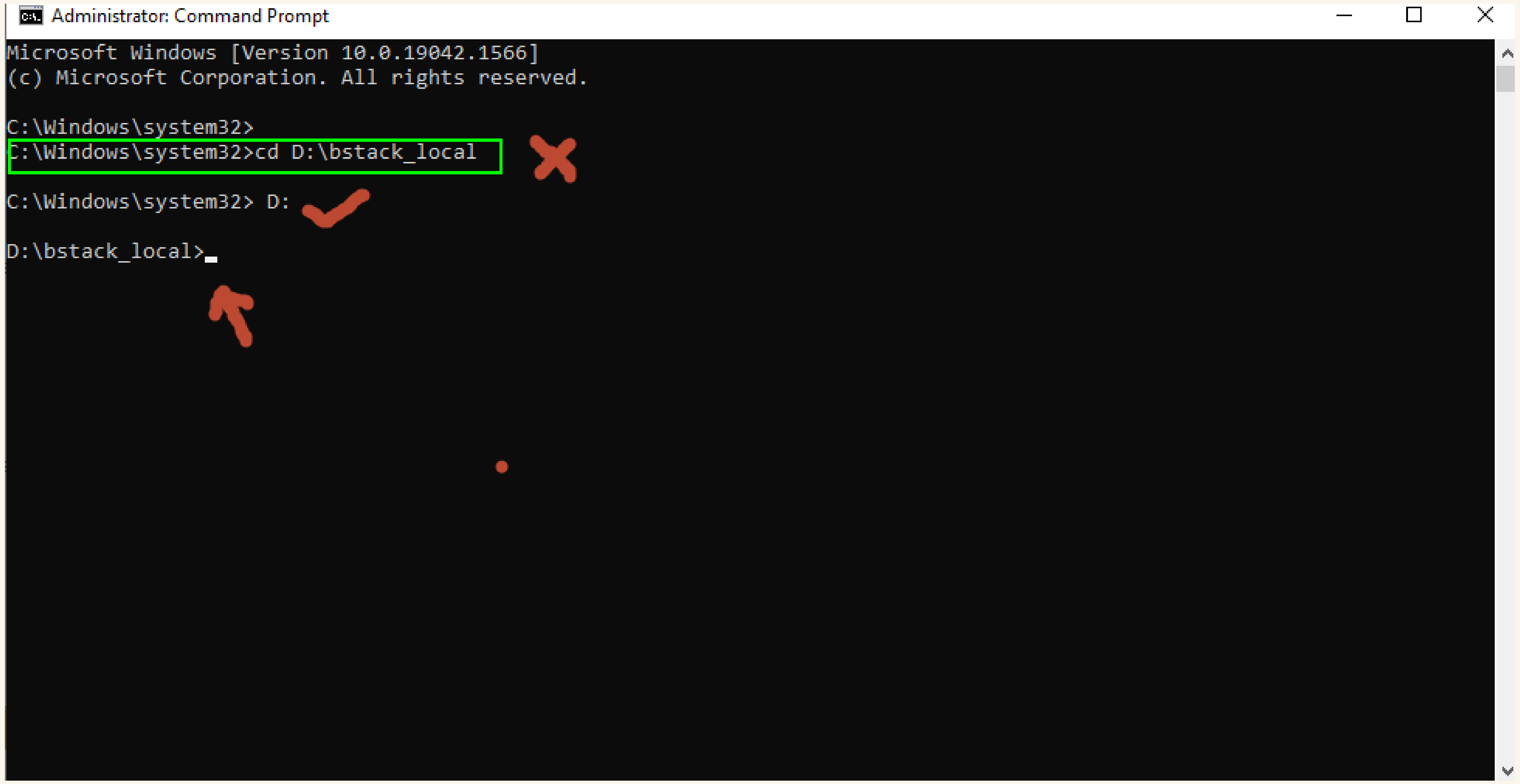
In System Information, we get all info about the pc model and make



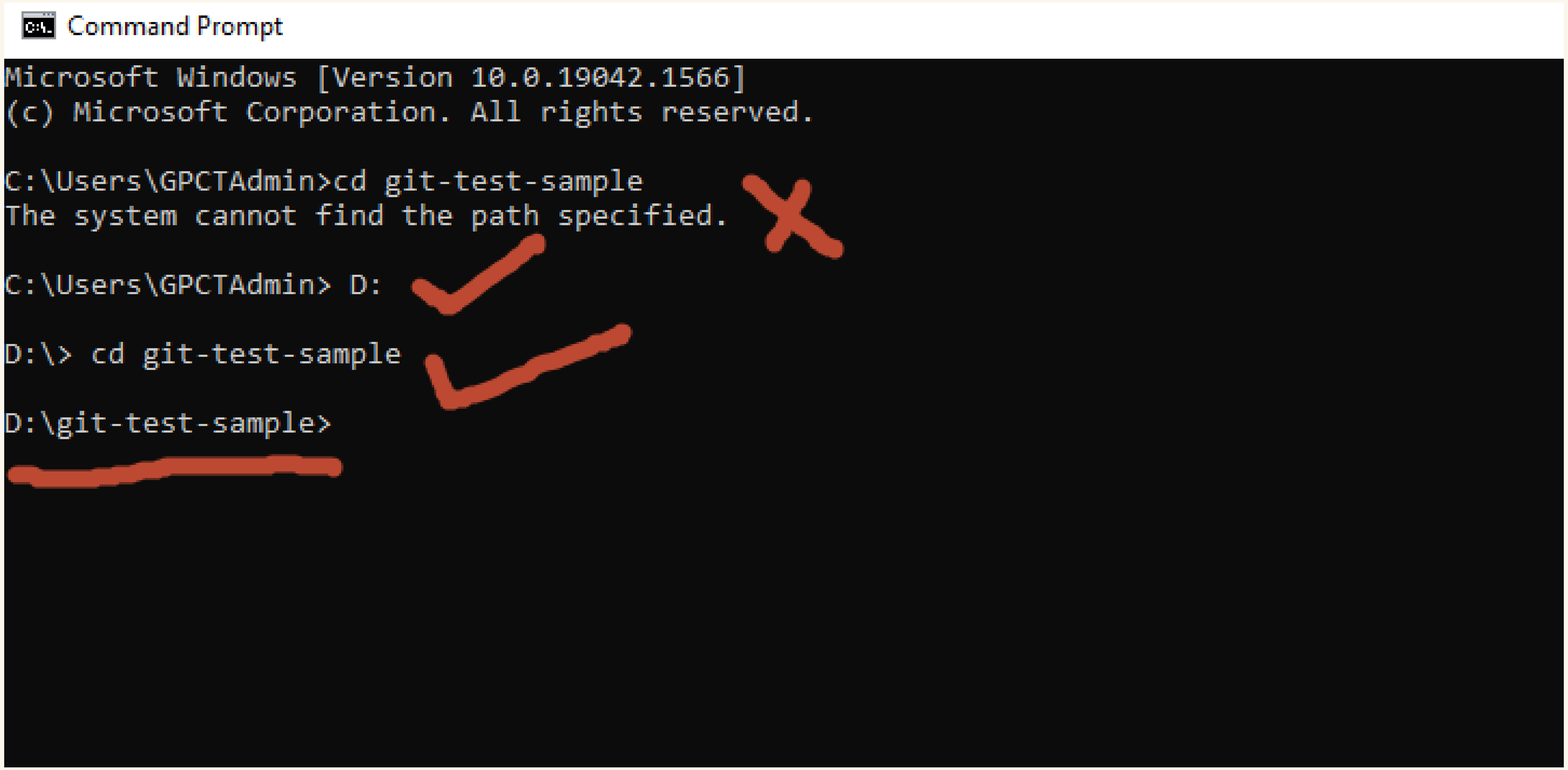
Item	Value
OS Name	Microsoft Windows 10 Pro
Version	10.0.19042 Build 19042
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	IN-LAP-SONIA
System Manufacturer	Dell Inc.
System Model	Latitude 5521
System Type	x64-based PC
System SKU	0A67
Processor	11th Gen Intel(R) Core(TM) i7-11850H @ 2.50GHz, 2496 Mhz, 8 Core(s), 16
BIOS Version/Date	Dell Inc. 1.5.3, 8/27/2021
SMBIOS Version	3.2
Embedded Controller Version	255.255
BIOS Mode	UEFI
BaseBoard Manufacturer	Dell Inc.
BaseBoard Product	0CWP5J
BaseBoard Version	A00
Platform Role	Mobile
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\Windows
System Directory	C:\Windows\system32
Boot Device	\Device\HarddiskVolume1
Locale	United States
Hardware Abstraction Layer	Version = "10.0.19041.1566"
User Name	IN-LAP-SONIA\GPCTAdmin
Time Zone	India Standard Time
Installed Physical Memory (RAM)	16.0 GB
Total Physical Memory	15.7 GB
Available Physical Memory	308 MB
Total Virtual Memory	39.3 GB

Note: Type System Information in the Start menu.

Successfully changed the directory using CLI



```
Microsoft Windows [Version 10.0.19042.1566]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>
C:\Windows\system32>cd D:\bstack_local

C:\Windows\system32> D:

D:\bstack_local>_
```



```
Microsoft Windows [Version 10.0.19042.1566]
(c) Microsoft Corporation. All rights reserved.

C:\Users\GPCTAdmin>cd git-test-sample
The system cannot find the path specified.

C:\Users\GPCTAdmin> D:

D:\> cd git-test-sample

D:\git-test-sample>
```

Git Username and Mail ID

1. How to setup your Git username?

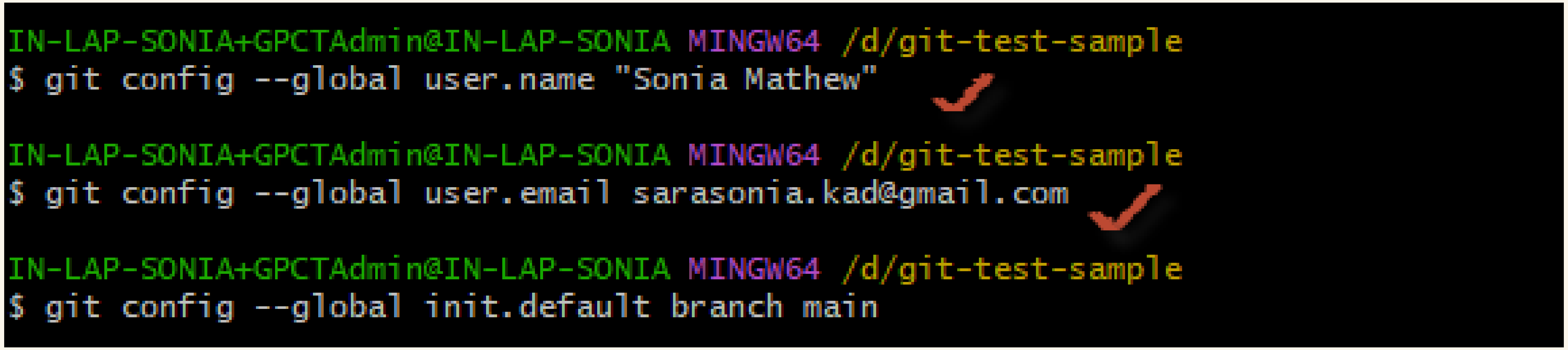
With the command below you can configure your user name:

```
git config --global user.name "Fabio"
```

2. How to setup your Git user email?

This command lets you setup the user email address you'll use in your commits.

```
git config --global user.email "signups@fabiopacifici.com"
```



```
IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample
$ git config --global user.name "Sonia Mathew"

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample
$ git config --global user.email sarasonia.kad@gmail.com

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample
$ git config --global init.default branch main
```

Initialize projects using Git

Now to initialize your project, simply run git init. This will tell Git to get ready to start watching your files for every change that occurs. It looks like this:

```
git init
```

The first line has information about my PC and the path to where the folder exists. The second line is the command git init, and the third line is the response sent back telling me that my repository (repo) has been initialized. It is considered empty because we have not told Git what files to track.

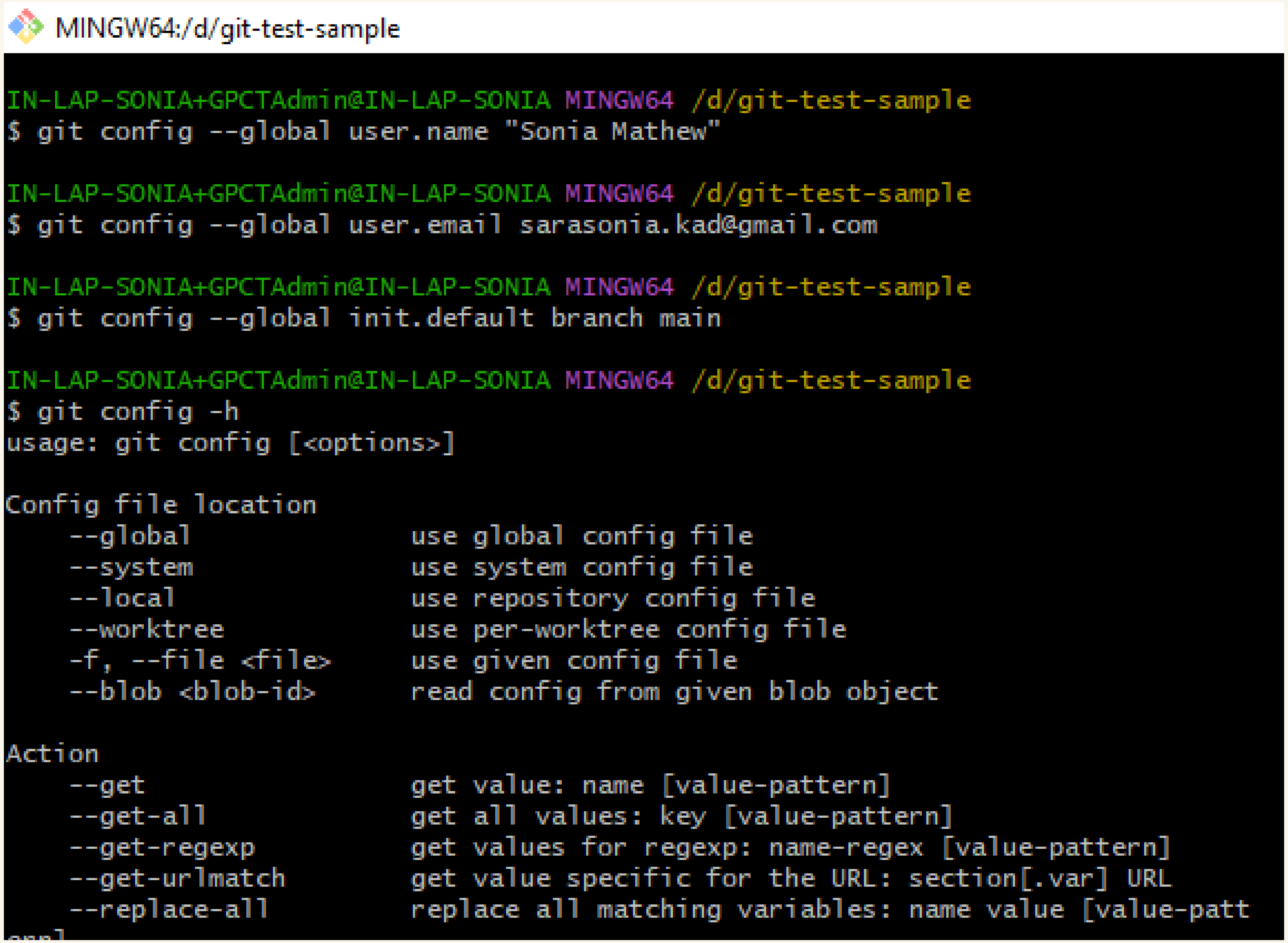
A repository is just another way to define a project being watched/tracked by Git.

OR

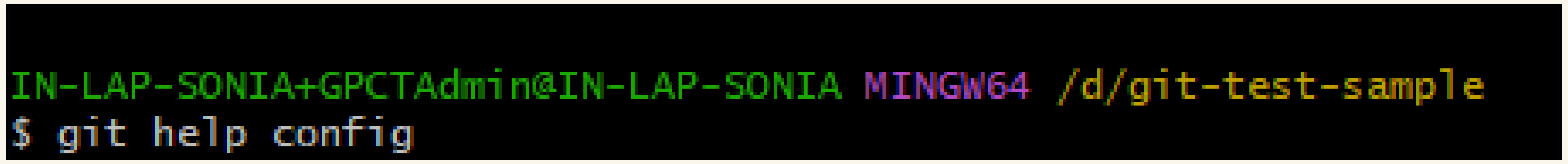
Note: You can also use:

```
git config --global init.default branch
```

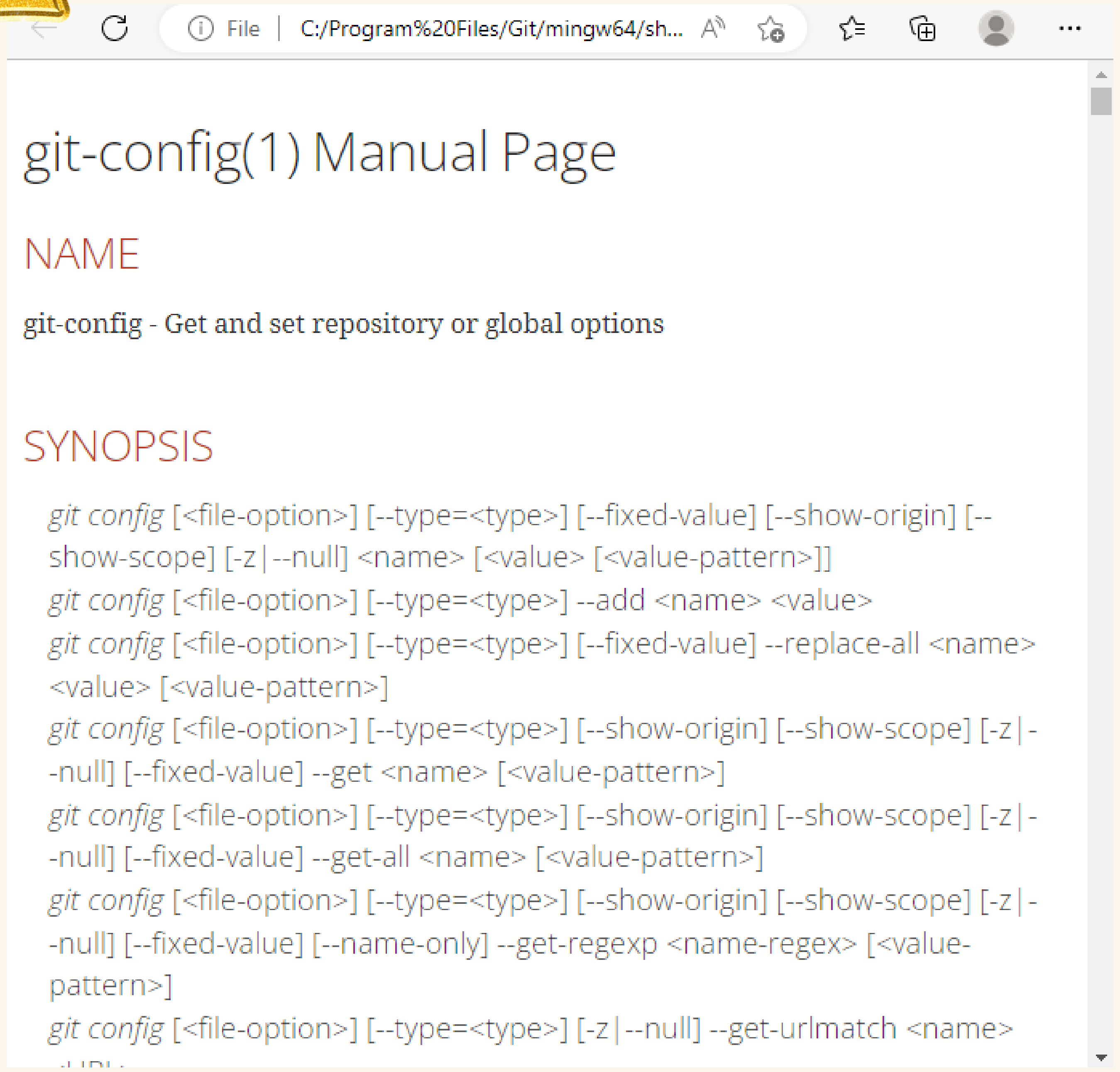
By default, Git will create a branch called master when you create a new repository with git init.


MINGW64:/d/git-test-sample

```
IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample
$ git config --global user.name "Sonia Mathew"

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample
$ git config --global user.email sarasonia.kad@gmail.com

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample
$ git config --global init.default branch main

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample
$ git config -h
usage: git config [<options>]

Config file location
    --global              use global config file
    --system              use system config file
    --local               use repository config file
    --worktree            use per-worktree config file
    -f, --file <file>     use given config file
    --blob <blob-id>      read config from given blob object

Action
    --get                 get value: name [value-pattern]
    --get-all             get all values: key [value-pattern]
    --get-regexp          get values for regexp: name-regex [value-pattern]
    --get-urlmatch        get value specific for the URL: section[.var] URL
    --replace-all         replace all matching variables: name value [value-patt
ern]
```



```
IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample
$ git help config
```



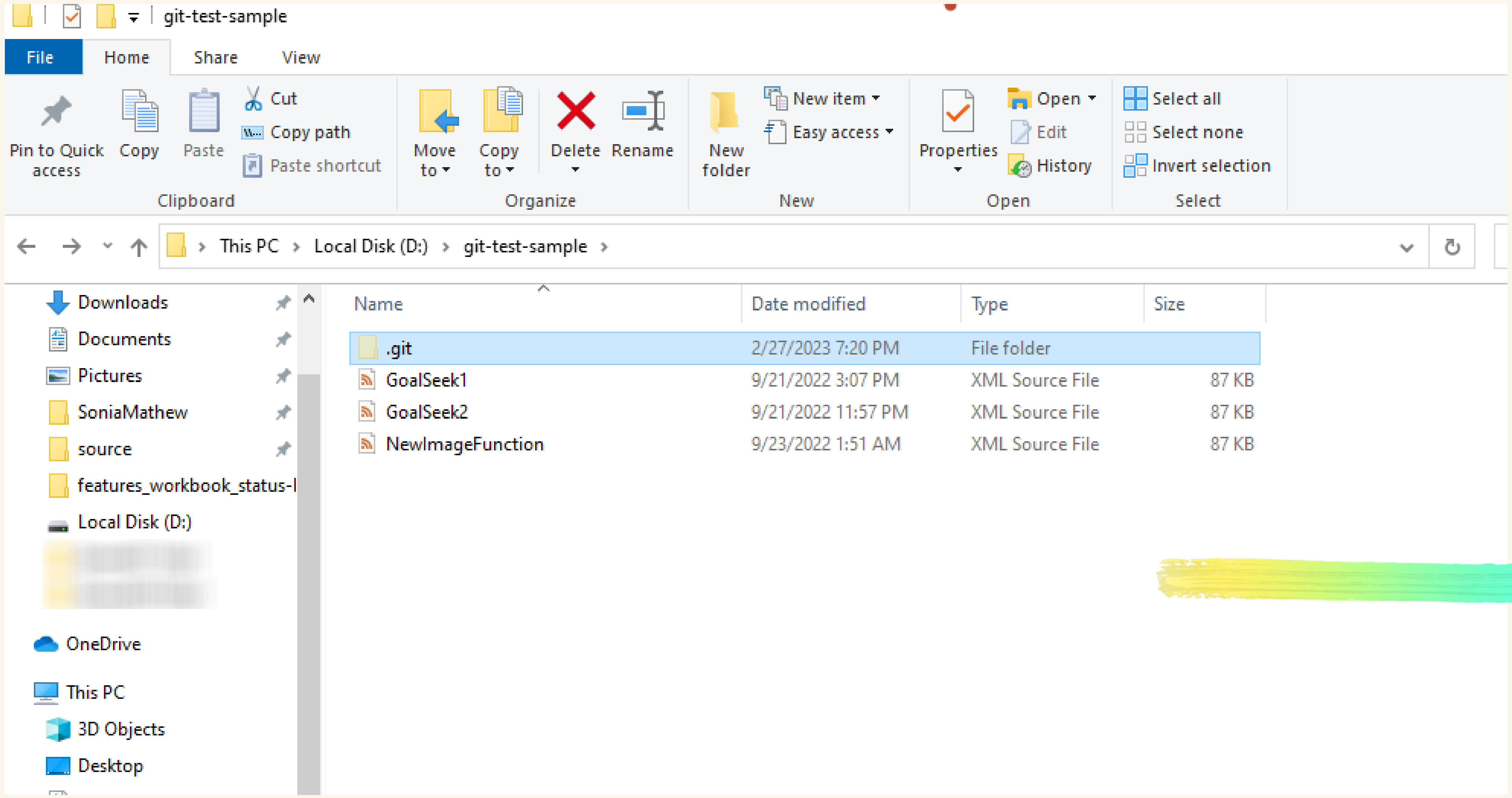
git-config(1) Manual Page

NAME

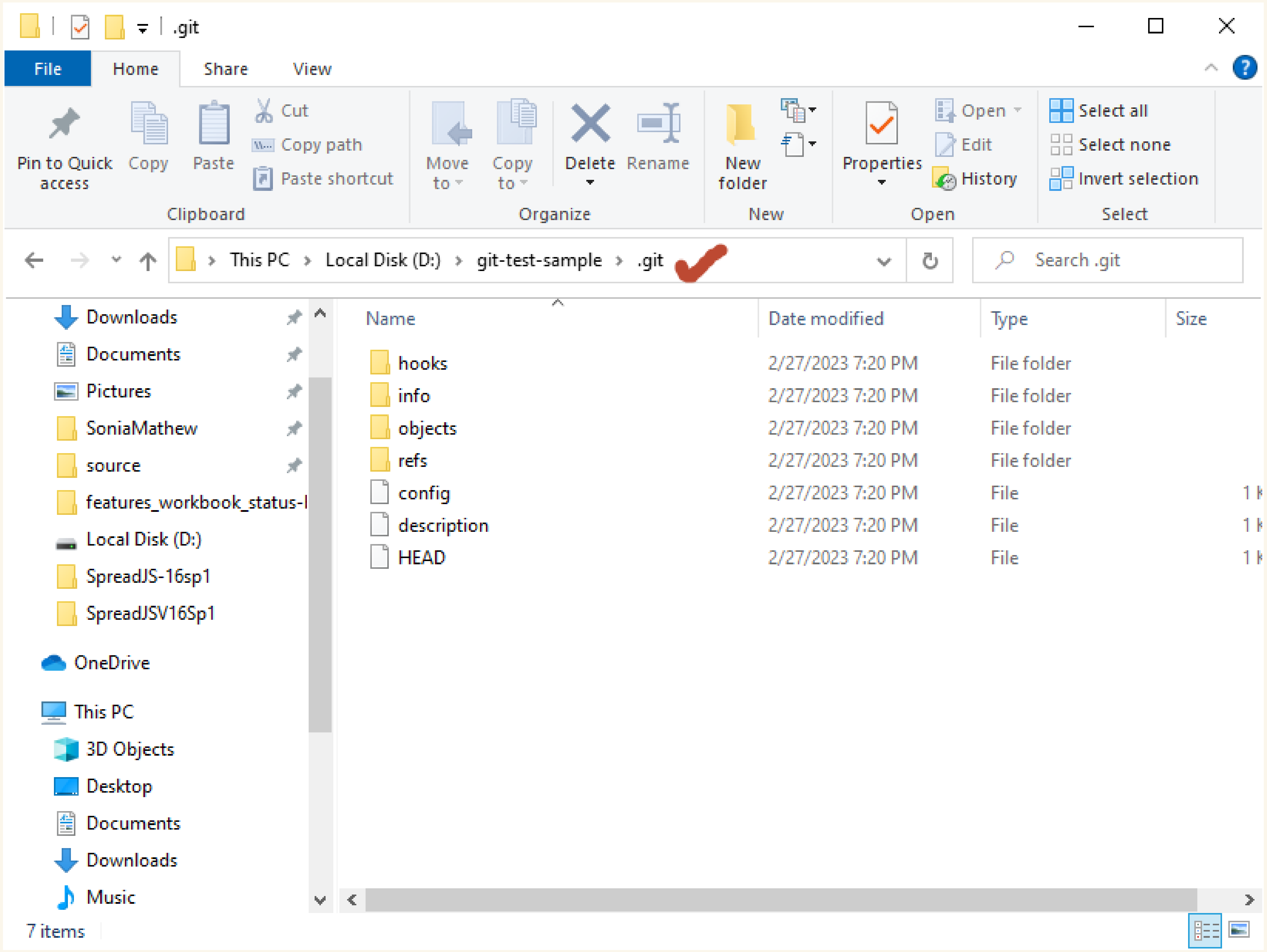
git-config - Get and set repository or global options

SYNOPSIS

```
git config [<file-option>] [--type=<type>] [--fixed-value] [--show-origin] [--show-scope] [-z|--null] <name> [<value> [<value-pattern>]]
git config [<file-option>] [--type=<type>] --add <name> <value>
git config [<file-option>] [--type=<type>] [--fixed-value] --replace-all <name> <value> [<value-pattern>]
git config [<file-option>] [--type=<type>] [--show-origin] [--show-scope] [-z|--null] [--fixed-value] --get <name> [<value-pattern>]
git config [<file-option>] [--type=<type>] [--show-origin] [--show-scope] [-z|--null] [--fixed-value] --get-all <name> [<value-pattern>]
git config [<file-option>] [--type=<type>] [--show-origin] [--show-scope] [-z|--null] [--fixed-value] [--name-only] --get-regexp <name-regex> [<value-pattern>]
git config [<file-option>] [--type=<type>] [-z|--null] --get-urlmatch <name>
```



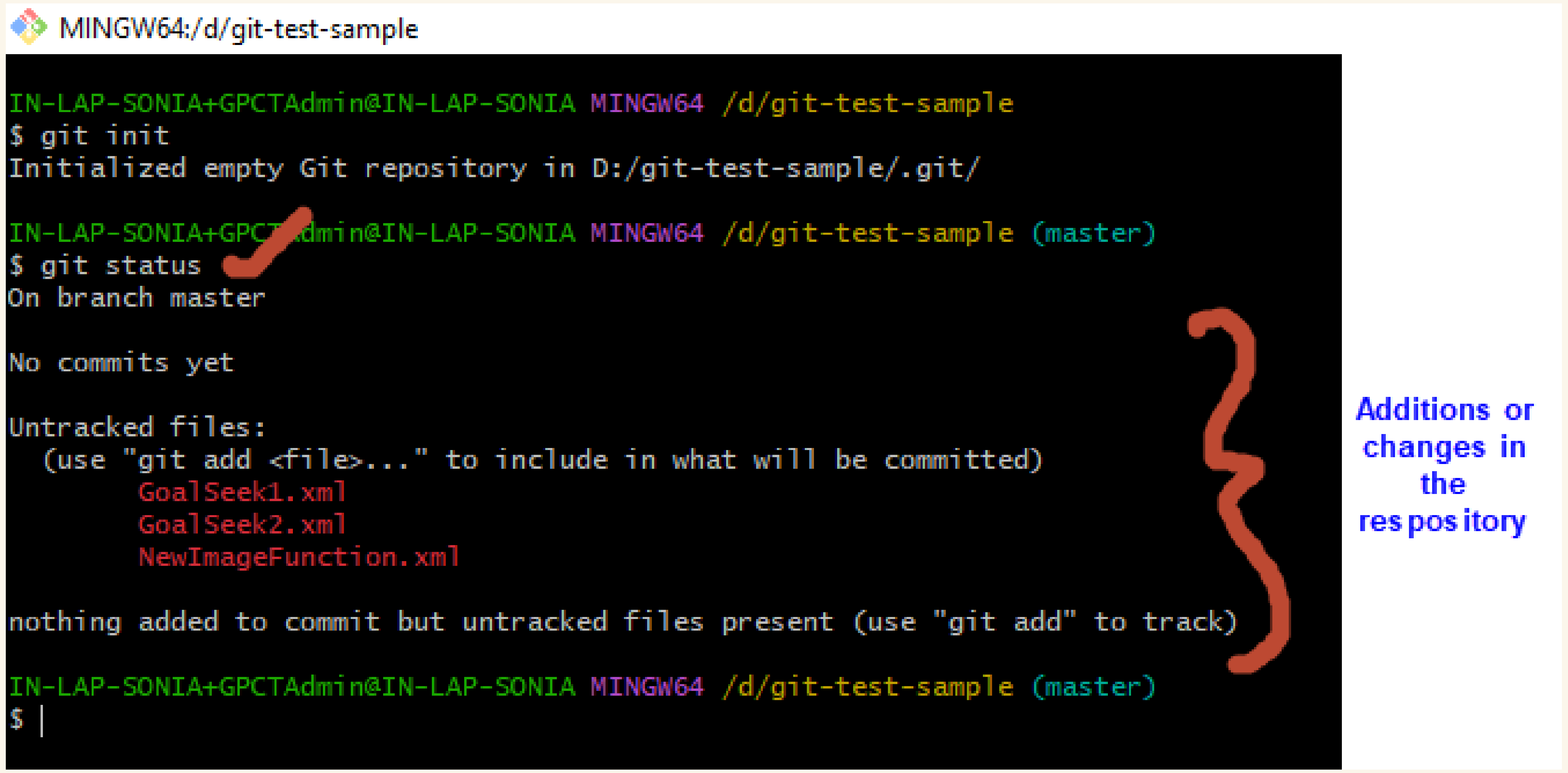
Name	Date modified	Type	Size
.git	2/27/2023 7:20 PM	File folder	
GoalSeek1	9/21/2022 3:07 PM	XML Source File	87 KB
GoalSeek2	9/21/2022 11:57 PM	XML Source File	87 KB
NewImageFunction	9/23/2022 1:51 AM	XML Source File	87 KB



Name	Date modified	Type	Size
hooks	2/27/2023 7:20 PM	File folder	
info	2/27/2023 7:20 PM	File folder	
objects	2/27/2023 7:20 PM	File folder	
refs	2/27/2023 7:20 PM	File folder	
config	2/27/2023 7:20 PM	File	1
description	2/27/2023 7:20 PM	File	1
HEAD	2/27/2023 7:20 PM	File	1

Know the Git Status

Know the status of the Git repository.



```
MINGW64:/d/git-test-sample

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample
$ git init
Initialized empty Git repository in D:/git-test-sample/.git/

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        GoalSeek1.xml
        GoalSeek2.xml
        NewImageFunction.xml

nothing added to commit but untracked files present (use "git add" to track)

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ |
```

After typing the following script:

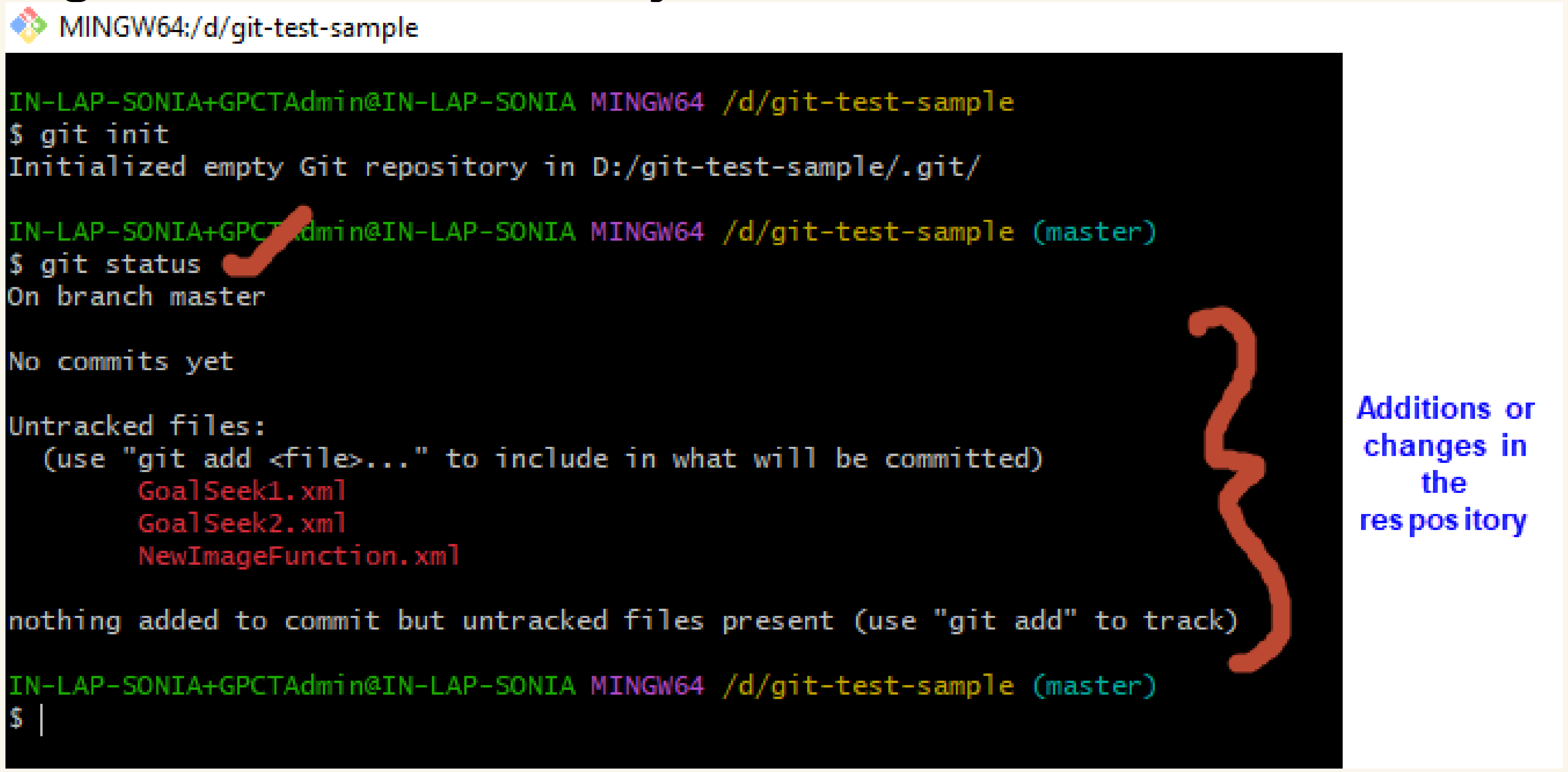
git status

the status of the repository can be viewed in the following scripts (image).

What info one gets after using 'git status'?

- On which branch the repository stays
- Any commits yet in the repository
- How many and what all untracked files are there in the repository

Note: The problem with untracked files is that if I make any changes to any one of those files, those changes won't be tracked by Git.



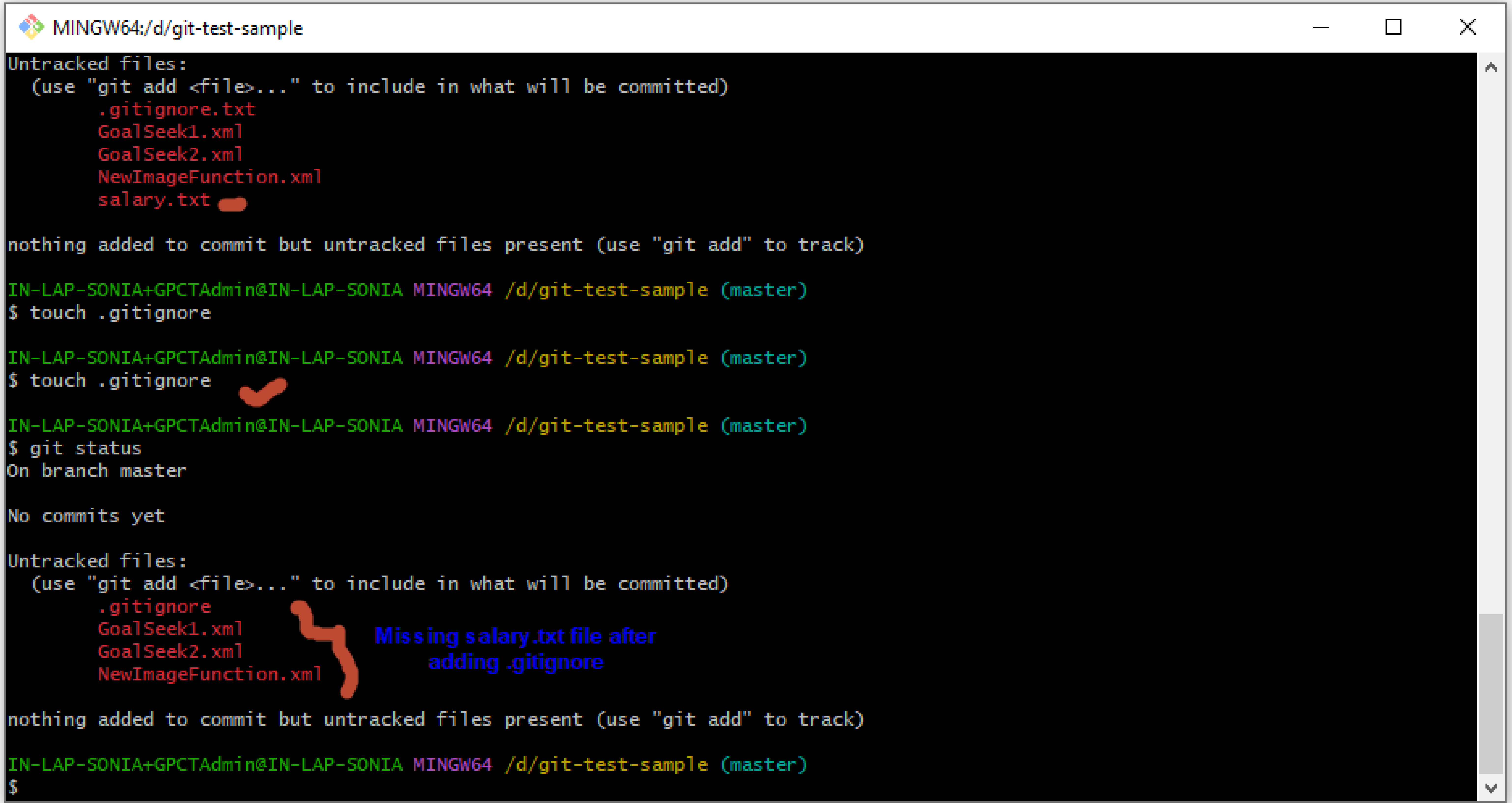
MINGW64:/d/git-test-sample

```
IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample
$ git init
Initialized empty Git repository in D:/git-test-sample/.git/

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        GoalSeek1.xml
        GoalSeek2.xml
        NewImageFunction.xml

nothing added to commit but untracked files present (use "git add" to track)

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ |
```

How **Git ignores** untracked files before adding it to repository?

To create a .gitignore file, go to the root of your local Git, and create it:

$ touch .gitignore



MINGW64:/d/git-test-sample

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore.txt
        GoalSeek1.xml
        GoalSeek2.xml
        NewImageFunction.xml
        salary.txt

nothing added to commit but untracked files present (use "git add" to track)

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ touch .gitignore

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ touch .gitignore

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        GoalSeek1.xml
        GoalSeek2.xml
        NewImageFunction.xml

nothing added to commit but untracked files present (use "git add" to track)

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$
```

Missing salary.txt file after adding .gitignore

Add files via Git (Staged State)

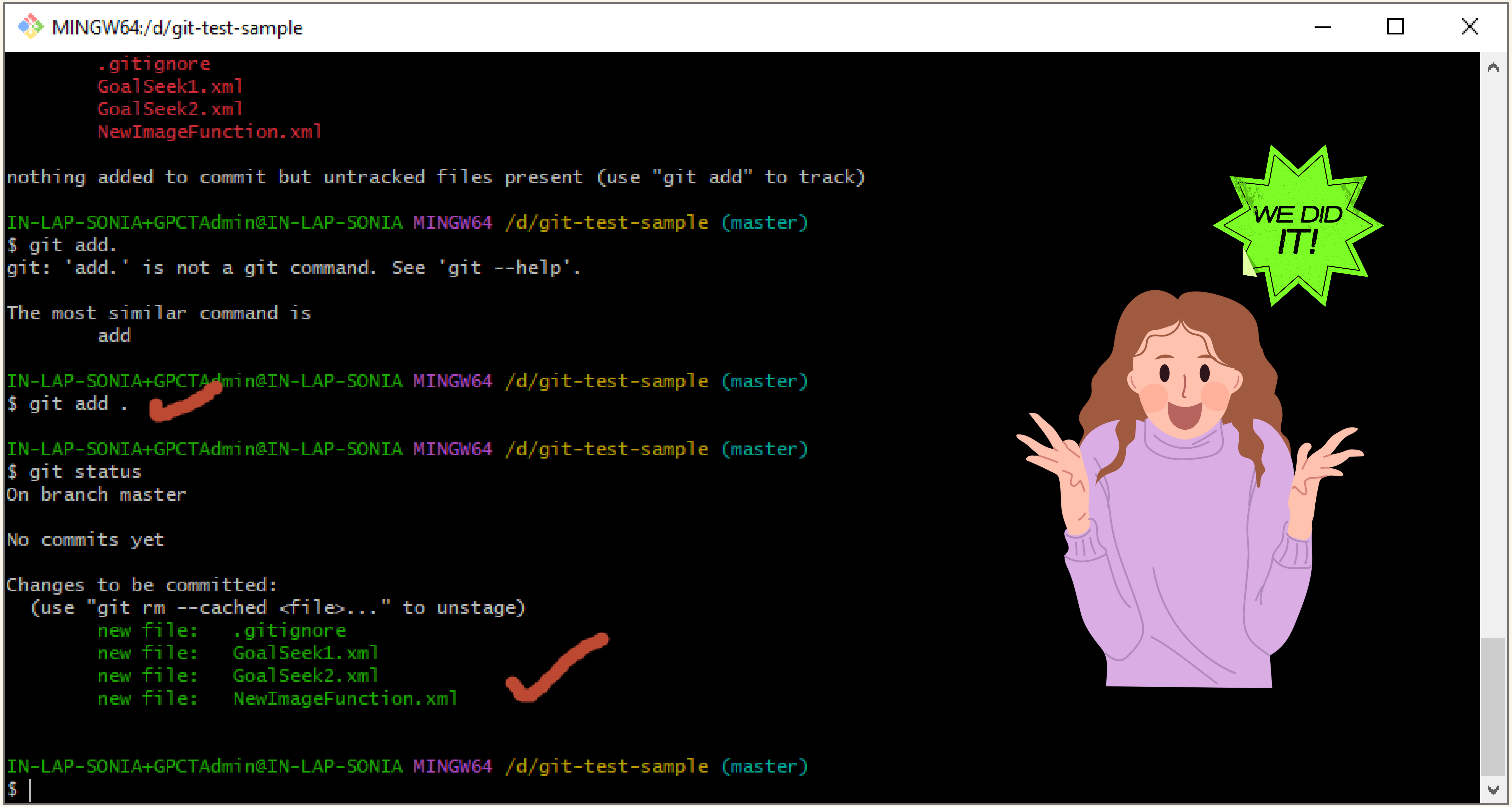
When we first initialized our project, the file was not being tracked by Git. To do that, we use this command:

```
git add .
```

The period or dot that comes after add means all the files that exist in the repository.
If you want to add a specific file, maybe one named about.txt, you use:

```
git add about.txt
```

Note: Now our file is in the **staged state**. You will not get a response after this command, but to know what state your file is in, you can run the git status command.

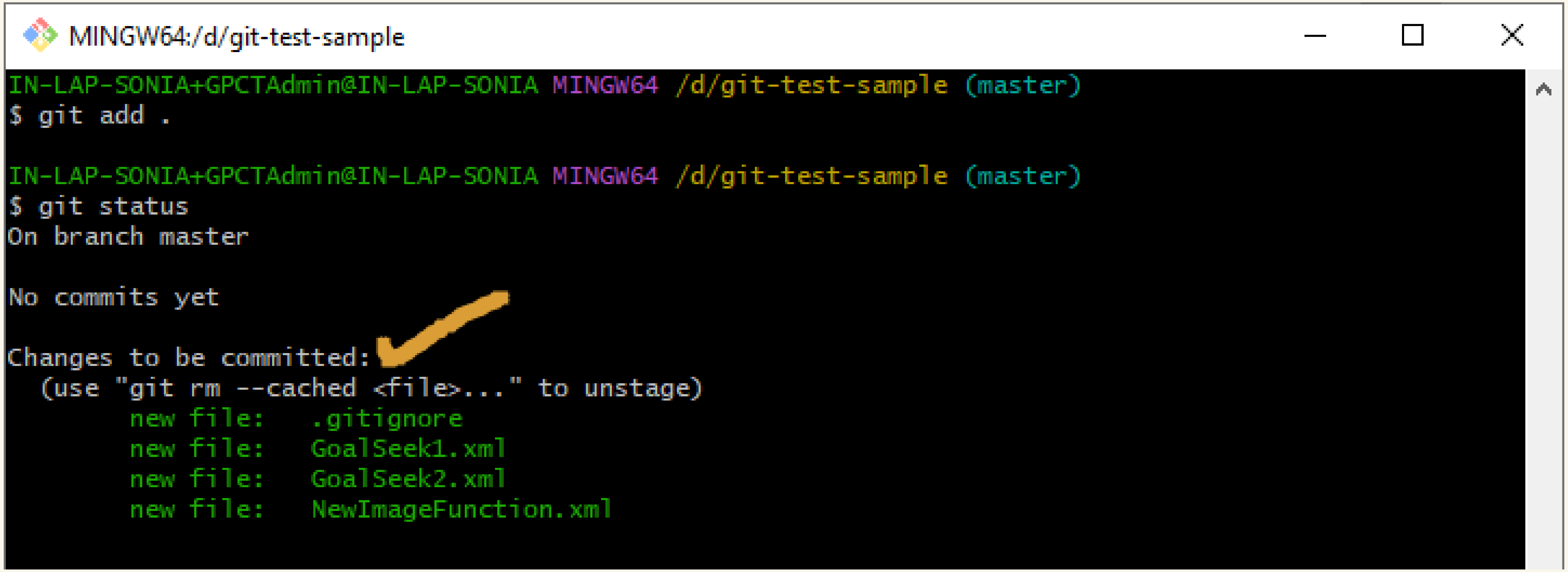


MINGW64:/d/git-test-sample

```
        .gitignore
        GoalSeek1.xml
        GoalSeek2.xml
        NewImageFunction.xml

nothing added to commit but untracked files present (use "git add" to track)

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ git add.
git: 'add.' is not a git command. See 'git --help'.

The most similar command is
        add

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ git add .

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   GoalSeek1.xml
        new file:   GoalSeek2.xml
        new file:   NewImageFunction.xml


IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ |
```

Commit files via Git (Committed State)

Post the addition of all the files, and after doing a git status, we get the following message:



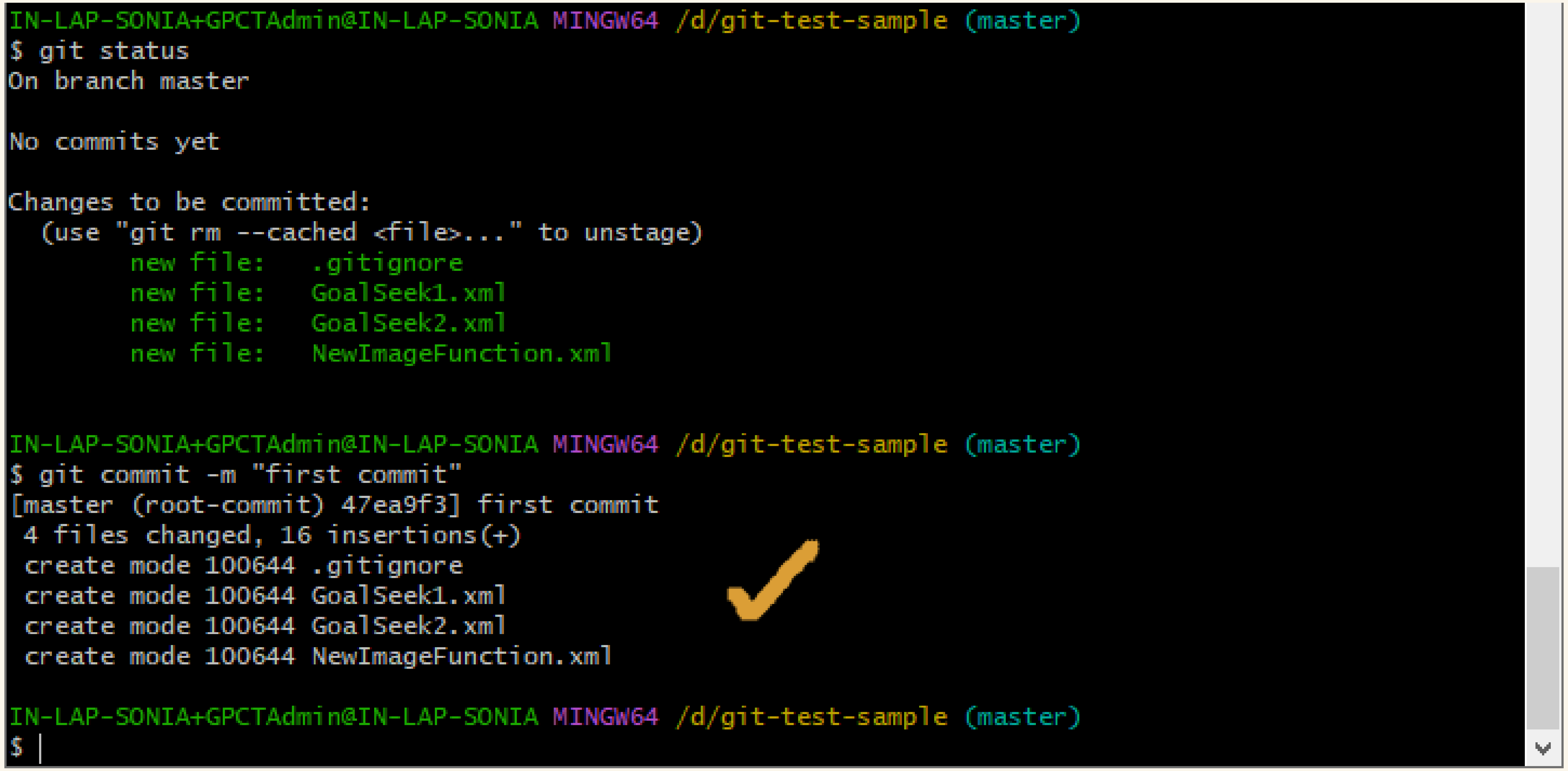
```
IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ git add .

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   GoalSeek1.xml
        new file:   GoalSeek2.xml
        new file:   NewImageFunction.xml
```

This signifies that these files are waiting for commit. Let's see how to commit files via Git.

The next state for a file after the **staged state** is the committed state. To commit our file, we use:
git commit -m "first commit"

The different parts of the command:
git commit : tells Git that all the files staged are ready to be committed so it is time to take a snapshot.
-m "first commit" : -m is shorthand for message while the text inside the parenthesis is the commit message.

After executing this command, you should get a response similar to this:



```
IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   GoalSeek1.xml
        new file:   GoalSeek2.xml
        new file:   NewImageFunction.xml


IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ git commit -m "first commit"
[master (root-commit) 47ea9f3] first commit
 4 files changed, 16 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 GoalSeek1.xml
 create mode 100644 GoalSeek2.xml
 create mode 100644 NewImageFunction.xml

IN-LAP-SONIA+GPCTAdmin@IN-LAP-SONIA MINGW64 /d/git-test-sample (master)
$ |
```

Committed State

A file is in the committed state when all the changes made to the file have been saved in the local repo.

Files in the **committed stage** are files **ready to be pushed** to the **remote repo** (on GitHub).

Success! Our files are in Committed state.

Ready for Git push to GitHub repository

Push the repository to GitHub