

# git

Dev Ops

## Git – Free and Open Source

Git is released under the [GNU General Public License version 2.0](#), which is an [open source license](#).

The Git project chose to use GPLv2 to guarantee your freedom to share and change free software---to make sure the software is free for all its users.

# Git

- Git is version-control system for tracking changes in source code during software development.
- It is designed for coordinating work among programmers, but it can be used to track changes in any set of files.
- Git helps them collaborate on code changes across multiple teams, manage different versions of code, and track code changes over time.
- Its goal is to increase efficiency, speed and easily manage large projects through version controlling.
- Every git working directory is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server.
- Git helps the team cope up with the confusion that tends to happen when multiple people are editing the same files
- Git is a distributed version control system that allows developers to manage source code changes and collaborate on projects.
- Microsoft uses Git to manage the source code for Windows and Office, two of their most important products.

# Git Bash

- Git Bash is an application that provides Git command line experience on the Operating System.
- It is a command-line shell for enabling git with the command line in the system.
- A shell is a terminal application used to interface with an operating system through written commands.
- Git Bash is a package that installs Bash, some common bash utilities, and Git on a Windows operating system.
- In Git Bash the user interacts with the repository and git elements through the commands.

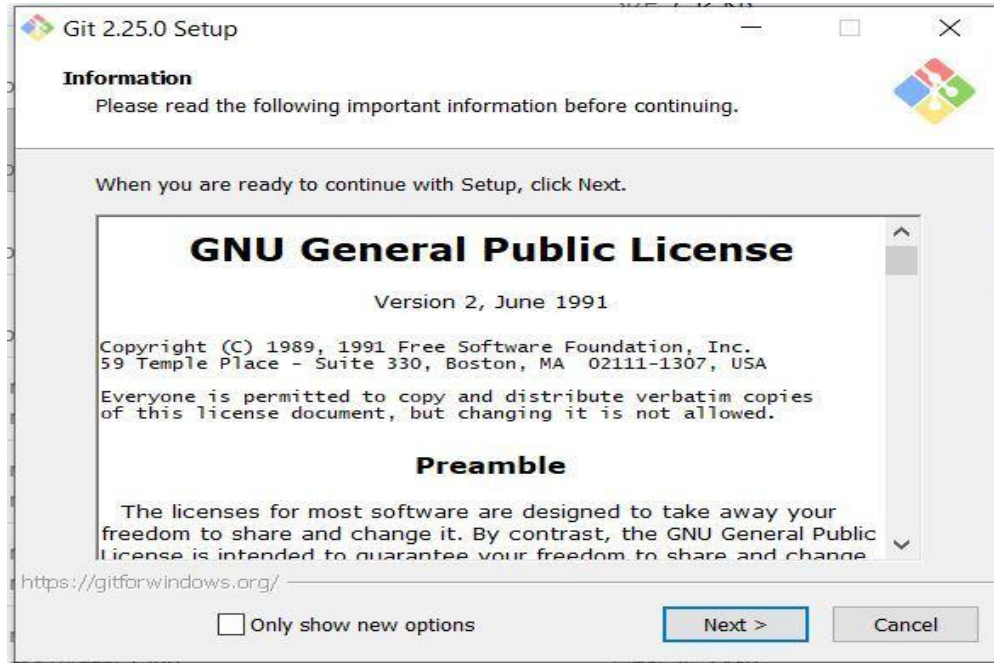
# Installing Git Bash

Steps to install Git Bash on Windows:

## Step 1:

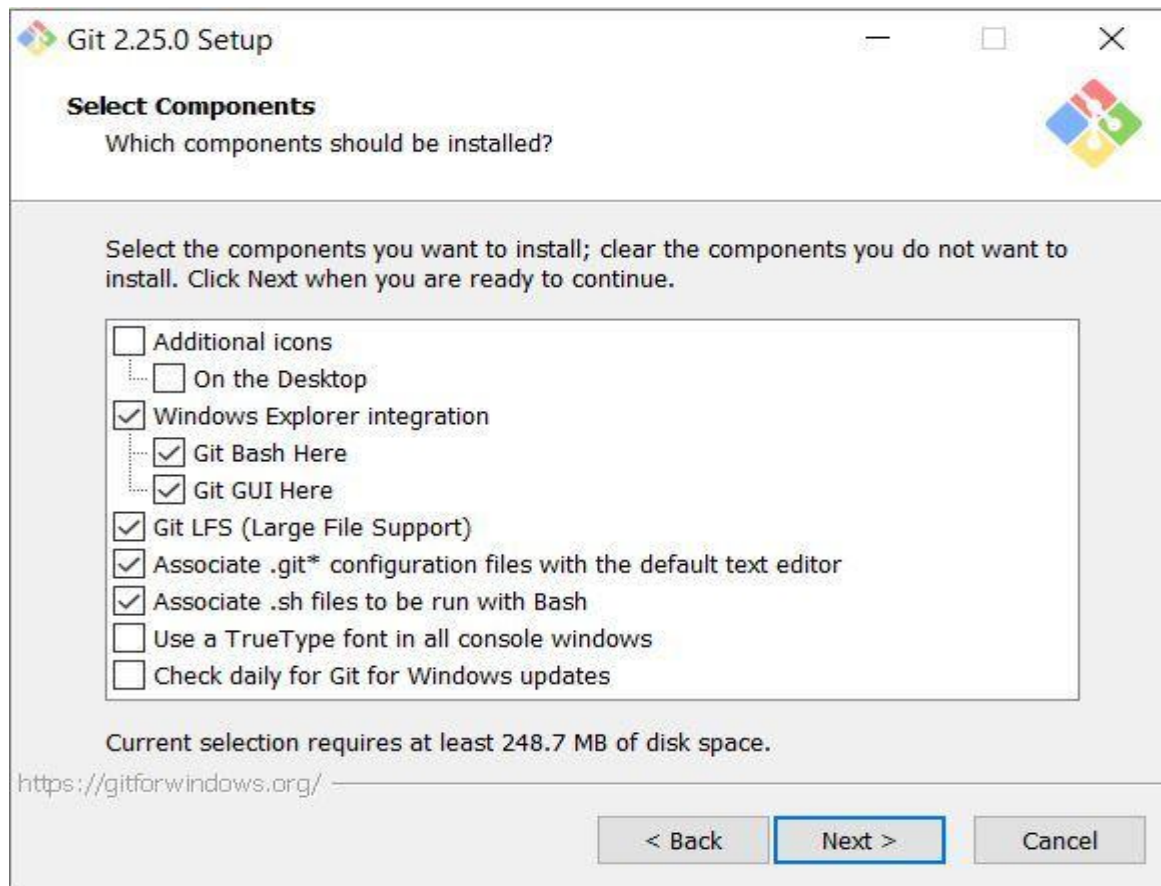
The .exe file installer for Git Bash can be downloaded from <https://gitforwindows.org/>

Once downloaded execute that installer, following window will occur:-



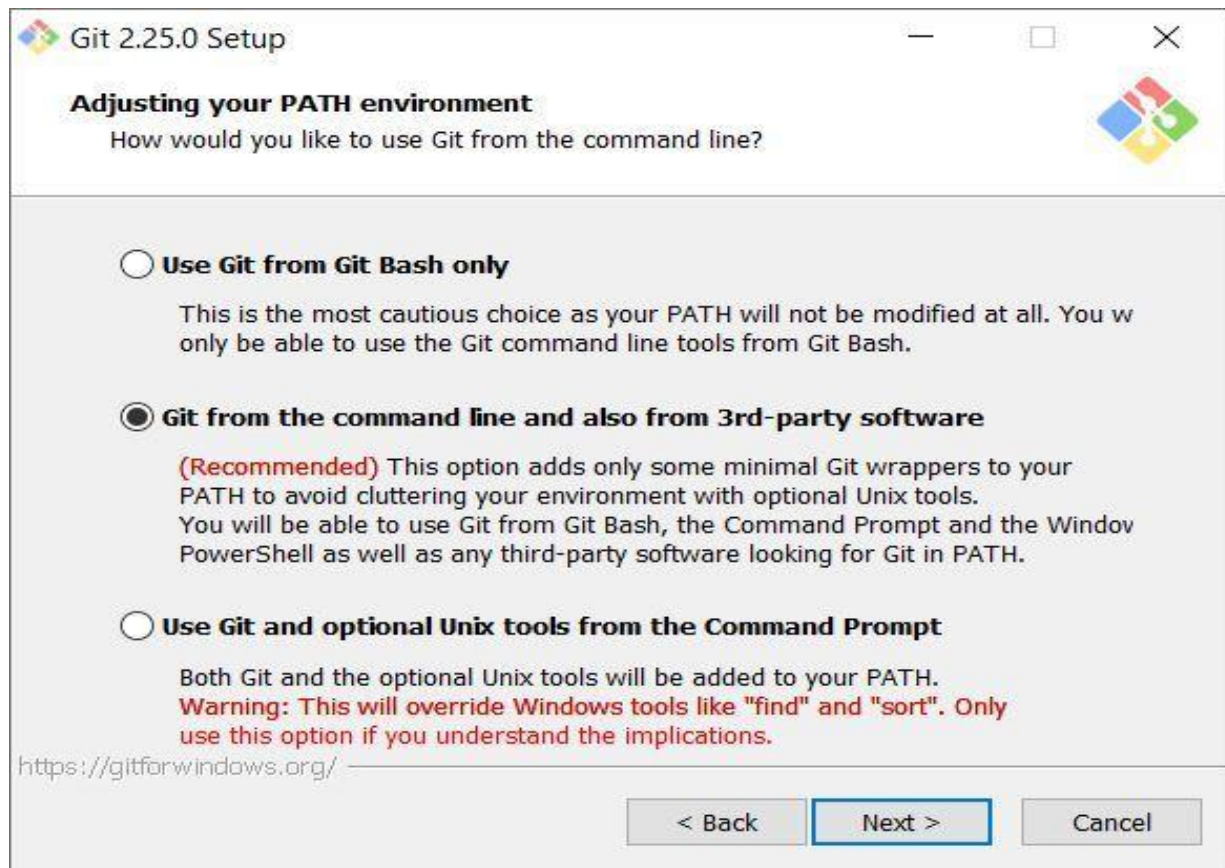
## Step 2:

Select the components that need to install and click on the Next button.

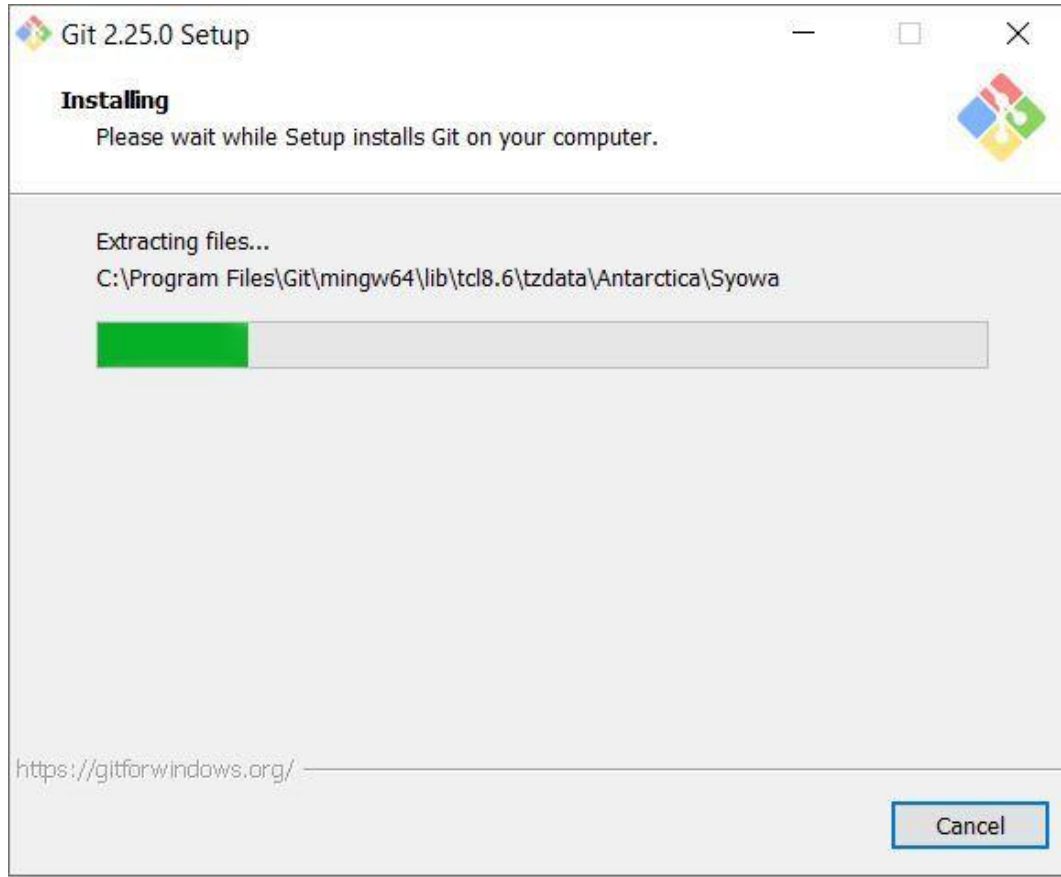


### Step 3:

Select how to use the Git from command-line and click on Next to begin the installation process.

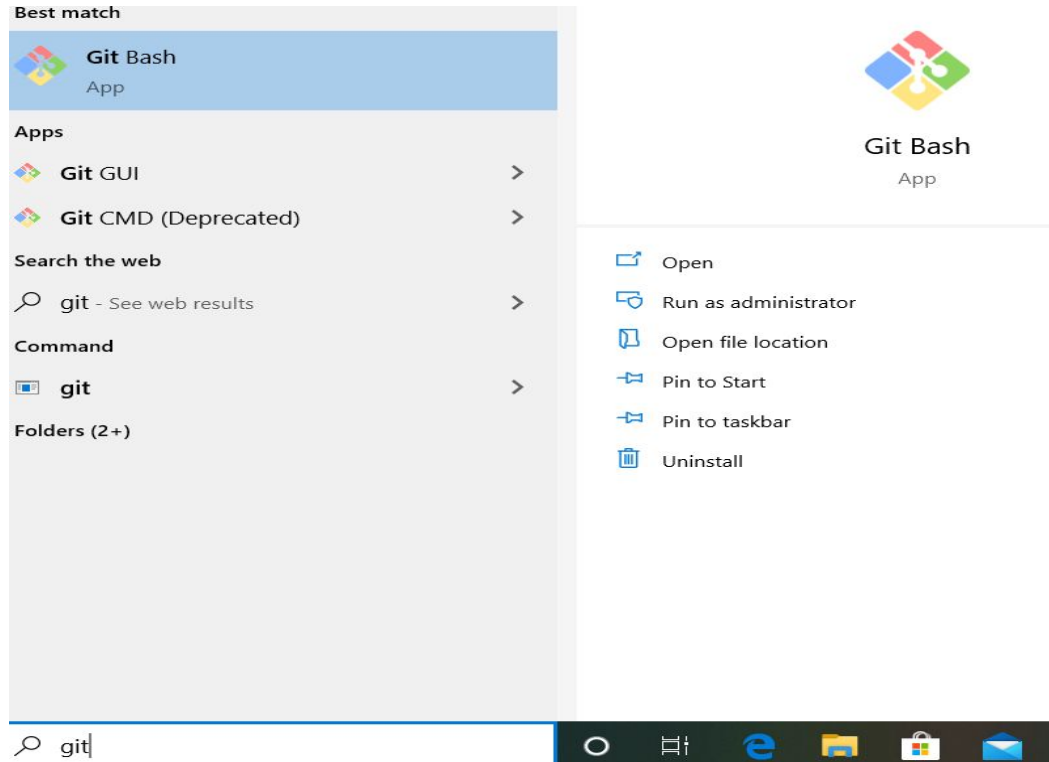


**Step 4:** finish the installation process to begin using Git Bash.





To open Git Bash navigate to the folder where we have installed the git otherwise just simply search in your OS for git bash.



# Navigate in Git Bash

## cd command

cd command refers to **change directory** and is used to get into the desired directory.

To navigate between the folders the **cd** command is used

### Syntax:

```
cd folder_name
```

## ls command

ls command is used to list all the files and folders in the current directory.

### Syntax:

```
ls
```

## Set global username/email configuration

Open Git Bash and begin creating a username and email for working on Git Bash.

### Set username:

```
git config --global user.name "FIRST_NAME LAST_NAME"
```

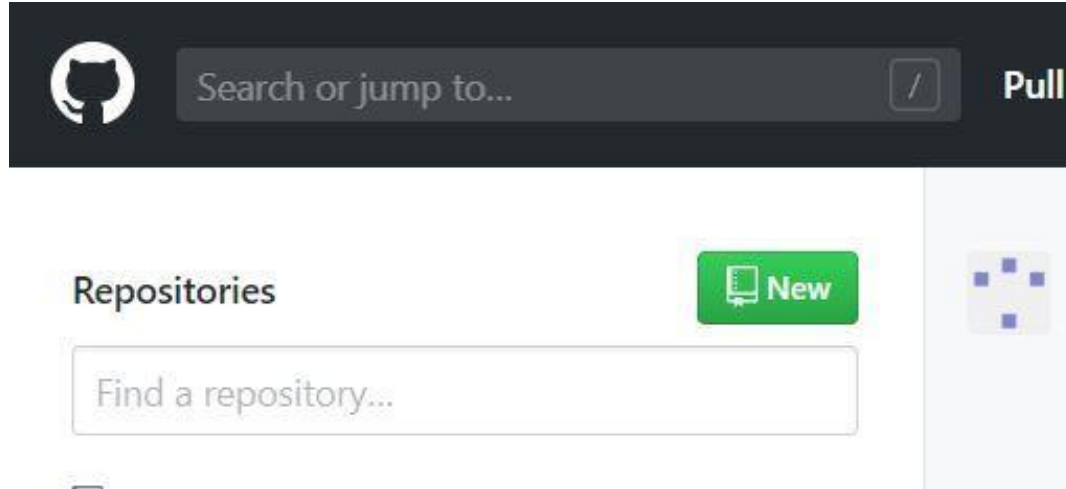
### Set email address:

```
git config --global user.email "MY_NAME@example.com"
```

# Initializing a Local repository

Steps to initialize our Local Repository with Git:

**Step 1:** Make a repository on [Github](#)



## Step 2: Give a suitable name of your repository and create the repository

### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner

 taran910 ▾

Repository name \*

NewRepo



Great repository names are short, simple, and memorable. Need inspiration? How about **curly-lamp**?

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

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

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾

Add a license: **None** ▾



Create repository

**Note:**

ou can choose to initialize your git repository with a README file, and further, you can mention your project details in it. It helps people know what this repository is about.

However, it's absolutely not necessary. But if you do initialize your repo with a README file using interface provided by GitHub, then your local repository won't have this README file.

So to avoid running into a snag while trying to push your files (as in step 3 of next section), after step 5 (where you initialize your local folder as your git repository), do following to pull that file to your local folder:

```
git pull
```

## Step 3: The following will appear after creating the repository

taran910 / NewRepo

Unwatch 1 Star 0 Fork 0

<> Code

Issues 0

Pull requests 0

Actions

Projects 0

Wiki

Security 0

Insights

Settings

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

Set up in Desktop

 or 

HTTPS

SSH

https://github.com/taran910/NewRepo.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

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

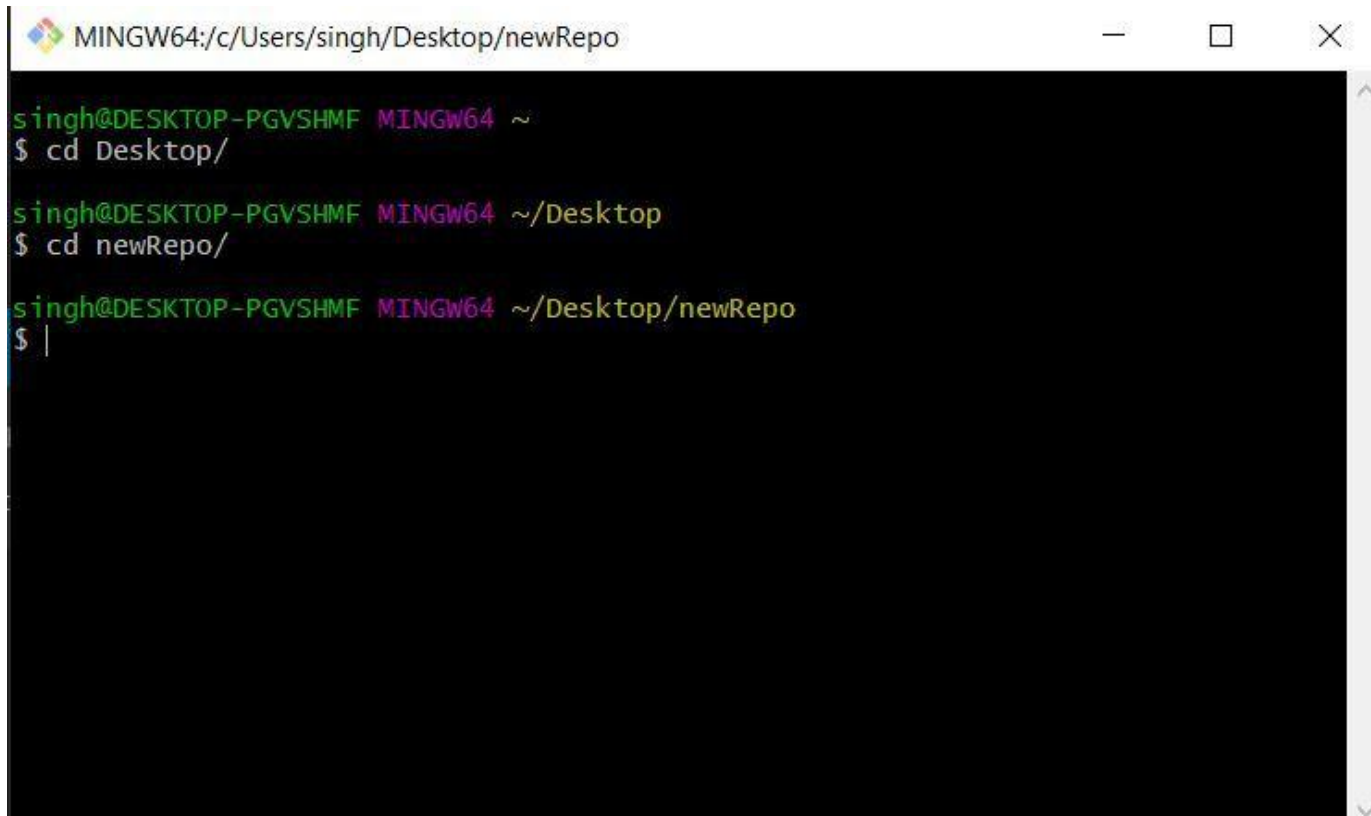
```
echo "# NewRepo" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/taran910/NewRepo.git
git push -u origin master
```

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

```
git remote add origin https://github.com/taran910/NewRepo.git
git push -u origin master
```

#### Step 4:

Open Git Bash and change the current working directory to your local project by use of cd command.



```
MINGW64:/c/Users/singh/Desktop/newRepo
singh@DESKTOP-PGVSHMF MINGW64 ~
$ cd Desktop/

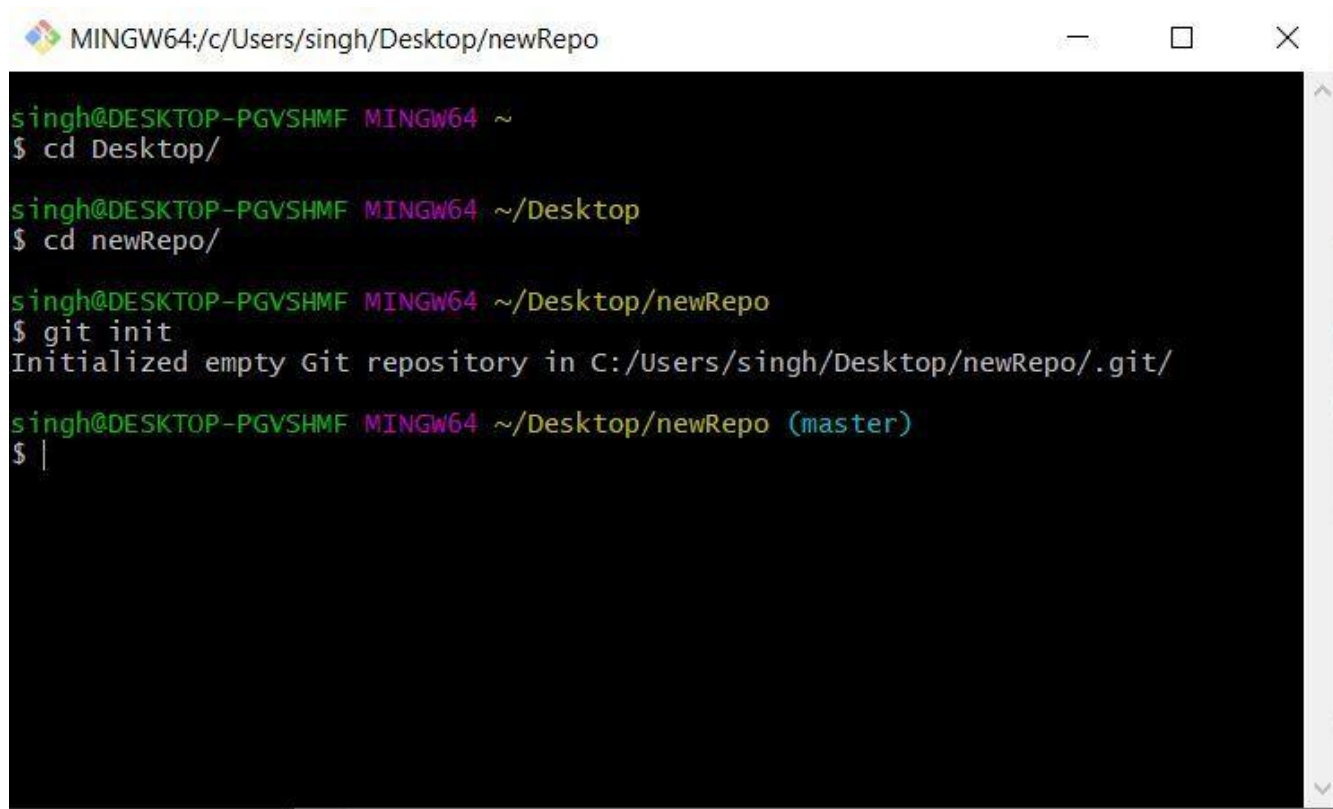
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop
$ cd newRepo/

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo
$ |
```



## Step 5: Initialize the local directory as a Git repository.

```
git init
```



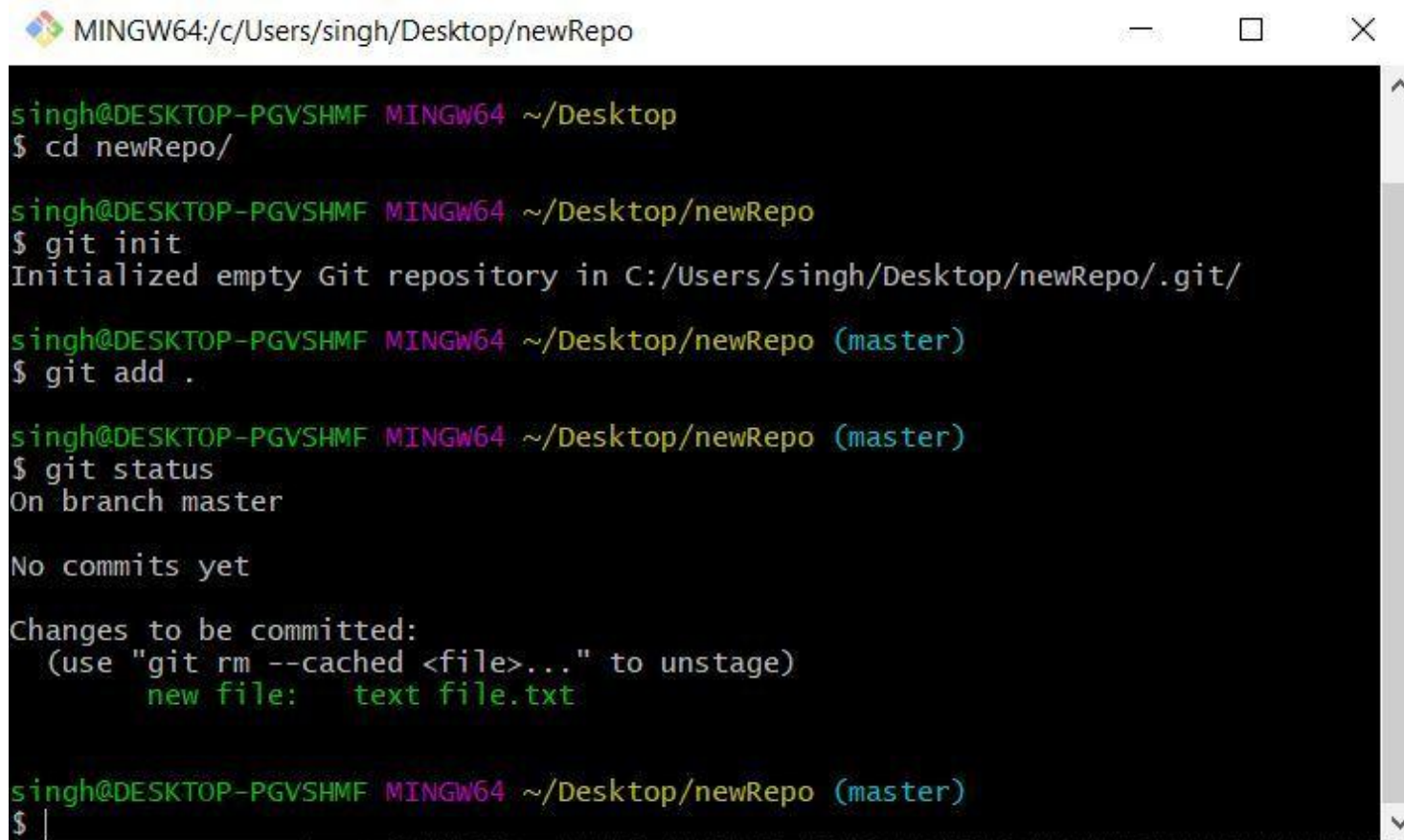
A screenshot of a Windows command prompt window titled "MINGW64:/c/Users/singh/Desktop/newRepo". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The command prompt shows the following sequence of commands and output:

```
singh@DESKTOP-PGVSHMF MINGW64 ~  
$ cd Desktop/  
  
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop  
$ cd newRepo/  
  
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo  
$ git init  
Initialized empty Git repository in C:/Users/singh/Desktop/newRepo/.git/  
  
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)  
$ |
```

## Step 6: Stage the files for the first commit by adding them to the local repository

```
git add .
```

## Step 7: By “git status” you can see the staged files



A screenshot of a Windows command prompt window titled "MINGW64:/c/Users/singh/Desktop/newRepo". The window shows the following commands and output:

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop
$ cd newRepo/

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo
$ git init
Initialized empty Git repository in C:/Users/singh/Desktop/newRepo/.git/

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git add .

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git status
On branch master

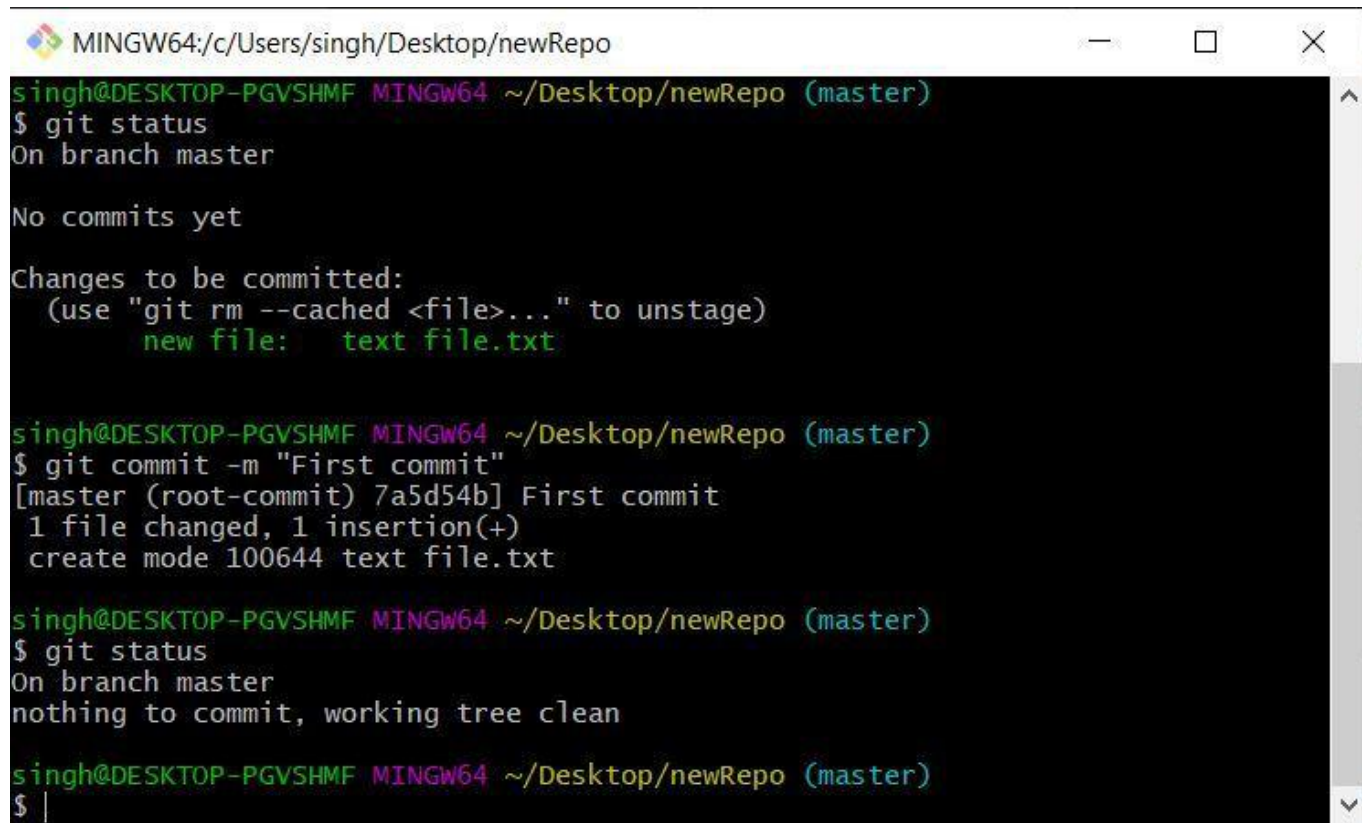
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   text file.txt

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ |
```

**Step 8:** Commit the files that we have staged in your local repository.

```
git commit -m "First commit"
```



A screenshot of a Windows terminal window titled "MINGW64:/c/Users/singh/Desktop/newRepo". The terminal shows the following sequence of commands and output:

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   text file.txt

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git commit -m "First commit"
[master (root-commit) 7a5d54b] First commit
1 file changed, 1 insertion(+)
create mode 100644 text file.txt


singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git status
On branch master
nothing to commit, working tree clean


singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ |
```


Now After “**git status**” command it can be seen that nothing to commit is left,  
Therefore We can see that, all files have been committed.


# Push files to your Git repository


**Step 1:** Go to Github repository and in the **code** section **copy the URL**.


 taran910 / NewRepo


 Unwatch ▾ 1


 Star 0


 Fork 0


 Code


 Issues 0


 Pull requests 0


 Actions

 Projects 0


 Wiki

 Security 0

 Insights

 Settings

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


 Set up in Desktop

 or 

HTTPS

SSH

`https://github.com/taran910/NewRepo.git`



Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

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

```
echo "# NewRepo" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/taran910/NewRepo.git
git push -u origin master
```



**Step 2:** In the Command prompt, add the URL for your repository where your local repository will be pushed.

```
git remote add origin repository_URL
```

**Step 3:** Push the changes in your local repository to GitHub.

```
git push origin master
```

Here the files have been pushed to the master branch of your repository.

MINGW64:/c/Users/singh/Desktop/newRepo



```
$ git commit -m "First commit"
[master (root-commit) 7a5d54b] First commit
1 file changed, 1 insertion(+)
create mode 100644 text file.txt
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
```

```
$ git status
```

```
On branch master
```

```
nothing to commit, working tree clean
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
```

```
$ git remote add origin https://github.com/taran910/NewRepo.git
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
```

```
$ git push origin master
```

```
Enumerating objects: 3, done.
```

```
Counting objects: 100% (3/3), done.
```

```
Writing objects: 100% (3/3), 227 bytes | 75.00 KiB/s, done.
```

```
Total 3 (delta 0), reused 0 (delta 0)
```

```
To https://github.com/taran910/NewRepo.git
```

```
* [new branch]      master -> master
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
```

```
$ |
```



Now in the GitHub repository, the pushed files can be seen.

taran910 / NewRepo

Unwatch 1

Star 0

Fork 0

<> Code

Issues 0

Pull requests 0

Actions

Projects 0

Wiki

Security 0

Insights

Settings

No description, website, or topics provided.

Edit

Manage topics

1 commit

1 branch

0 packages

0 releases

1 contributor

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

taran910

First commit

Latest commit 7a5d54b 20 minutes ago

text file.txt

First commit

20 minutes ago

Help people interested in this repository understand your project by adding a README.

Add a README

# Saving changes to local repository

Suppose the files are being changed and new files are added to local repository. To save the changes in the git repository:

**Step 1:** Changes have to be staged for the commit.

```
git add .
```

or

```
git add file_name
```

**Step 2:** Now commit the staged files.

```
git commit -m "commit_name"
```

**Step 3:** Push the changes.

```
git push origin master
```

## Saving changes to local repository

MINGW64:/c/Users/singh/Desktop/newRepo

nothing to commit, working tree clean

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)

\$ git add .

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)

\$ git commit -m "second commit"

[master 9e7f7d0] second commit

1 file changed, 1 insertion(+)

create mode 100644 abc/jhvjhb.txt

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)

\$ git push origin master

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Delta compression using up to 8 threads

Compressing objects: 100% (2/2), done.

Writing objects: 100% (4/4), 326 bytes | 65.00 KiB/s, done.

Total 4 (delta 0), reused 0 (delta 0)

To https://github.com/taran910/NewRepo.git

7a5d54b..9e7f7d0 master -> master

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)

\$ |


# New changes can be seen


 [taran910](#) / [NewRepo](#)

 Unwatch ▾ 1


 Star 0


 Fork 0


 Code


 Issues 0


 Pull requests 0


 Actions

 Projects 0

 Wiki

 Security 0

 Insights

 Settings

No description, website, or topics provided. [Edit](#)

[Manage topics](#)

 2 commits

 1 branch

 0 packages

 0 releases

 1 contributor

Branch: master ▾




New pull request

Create new file

Upload files

Find file

Clone or download ▾

 <a href="#">taran910</a> second commit		Latest commit 9e7f7d0 6 minutes ago
 <a href="#">abc</a>	second commit	6 minutes ago
 <a href="#">text file.txt</a>	First commit	1 hour ago

Help people interested in this repository understand your project by adding a README. [Add a README](#)

# Branching through Git Bash

## Branching in Github

- Suppose if a team is working on a project and a branch is created for every member working on the project.
- Each member will work on their branches hence every time the best branch is merged to the master branch of the project.
- The branches make it a version controlling system and makes it very easy to maintain a project source code.

## Syntax:

**List** all of the branches in your repository.

```
git branch
```

**Create** a new branch

```
git branch branch_name
```

**Safe Delete** the specified branch

```
git branch -d branch_name
```

**Force delete** the specified branch

```
git branch -D branch_name
```

## Navigating between Branches

- To navigate between the branches **git checkout** is used.
- To create create a new branch and switch on it:
  - `git checkout -b new_branch_name`
- To simply switch to a branch
  - `git checkout branch_name`
- After checkout to branch you can see a \* on the current branch

MINGW64:/c/Users/singh/Desktop/newRepo

```
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 326 bytes | 65.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/taran910/NewRepo.git
    7a5d54b..9e7f7d0  master -> master
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git branch
* master
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git checkout -b newbranch
Switched to a new branch 'newbranch'
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (newbranch)
$ git branch
master
* newbranch
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (newbranch)
$ |
```



Now the same **commit add** and **commit actions** can be performed on this branch also.

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (newbranch)
$ git add .
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (newbranch)
$ git commit -m "branch commit"
[newbranch 3eb93e9] branch commit
1 file changed, 1 insertion(+)
create mode 100644 branch file.txt
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (newbranch)
$ git push origin newbranch
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 312 bytes | 104.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'newbranch' on GitHub by visiting:
remote:   https://github.com/taran910/NewRepo/pull/new/newbranch
remote:
To https://github.com/taran910/NewRepo.git
 * [new branch]      newbranch -> newbranch
```

taran910 / NewRepo

Unwatch 1

Star 0

Fork 0

Code

Issues 0

Pull requests 0

Actions

Projects 0

Wiki

Security 0

Insights

Settings

No description, website, or topics provided.

Edit

Manage topics

2 commits

2 branches

0 packages

0 releases

1 contributor

Your recently pushed branches:

newbranch (less than a minute ago)

Compare & pull request

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

taran910 second commit

Latest commit 9e7f7d0 2 hours ago

abc

second commit

2 hours ago

text file.txt

First commit

3 hours ago

Help people interested in this repository understand your project by adding a README.

Add a README

## Merge any two branches

To merge a branch in any branch:

First reach to the target branch

```
git checkout branch_name
```

Merge the branch to target branch

```
git merge new_branch
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (newbranch)
```

```
$ git checkout master  
Switched to branch 'master'
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
```

```
$ git merge newbranch  
Updating 9e7f7d0..3eb93e9  
Fast-forward  
  branch file.txt | 1 +  
  1 file changed, 1 insertion(+)  
  create mode 100644 branch file.txt
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
```

```
$ |
```

## Cloning Repository to system

- Cloning is used to get a **copy** of the existing **git repository**.
- When we run the **git clone** command it makes the zip folder saved in our default location

```
git clone url
```

- This command saves the directory as the default directory name of the git repository
- To save directory name as our custom name an additional argument is to be passed for our custom name of directory

```
git clone url custom_name
```

```
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git clone https://github.com/taran910/NewRepo.git
Cloning into 'NewRepo'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 10 (delta 0), reused 10 (delta 0), pack-reused 0
Unpacking objects: 100% (10/10), 781 bytes | 2.00 KiB/s, done.

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ |
```

## Undoing commits

When there is a situation when we forget to add some files to commit and want to undo any commit, it can be commit again using **–amend Syntax**:

```
git commit --amend
```













# Branching and Merging

- The Git feature that really makes it stand apart from nearly every other SCM out there is its branching model.
- Git allows and encourages you to have multiple local branches that can be entirely independent of each other.
- The creation, merging, and deletion of those lines of development takes seconds.
- **Frictionless Context Switching.**
  - Create a branch to try out an idea, commit a few times, switch back to where you branched from, apply a patch, switch back to where you are experimenting, and merge it in.
- **Role-Based Codelines.**
  - Have a branch that always contains only what goes to production, another that you merge work into for testing, and several smaller ones for day to day work.
- **Feature Based Workflow.**
  - Create new branches for each new feature you're working on so you can seamlessly switch back and forth between them, then delete each branch when that feature gets merged into your main line.
- **Disposable Experimentation.**
  - Create a branch to experiment in, realize it's not going to work, and just delete it - abandoning the work—with nobody else ever seeing it (even if you've pushed other branches in the meantime).

## Branching and Merging



## Distributed

One of the nicest features of any Distributed SCM, Git included, is that it's distributed.

This means that instead of doing a "checkout" of the current tip of the source code, you do a "clone" of the entire repository.

## Multiple Backups

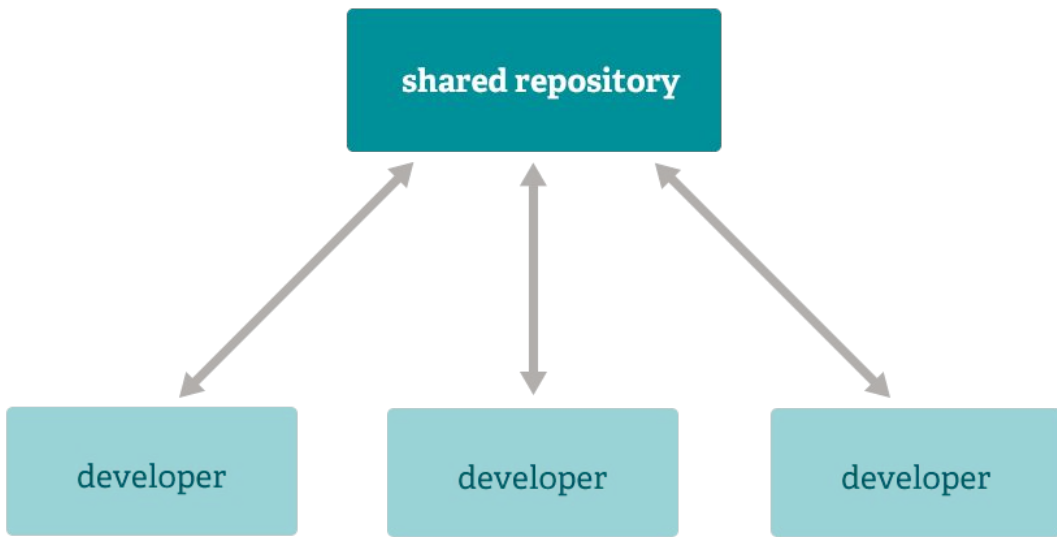
- even if you're using a centralized workflow, every user essentially has a full backup of the main server.
- each of these copies could be pushed up to replace the main server in the event of a crash or corruption.
- In effect, there is no single point of failure with Git unless there is only a single copy of the repository.

## Any Workflow

Because of Git's distributed nature and superb branching system, an almost endless number of workflows can be implemented with relative ease.

## Subversion-Style Workflow

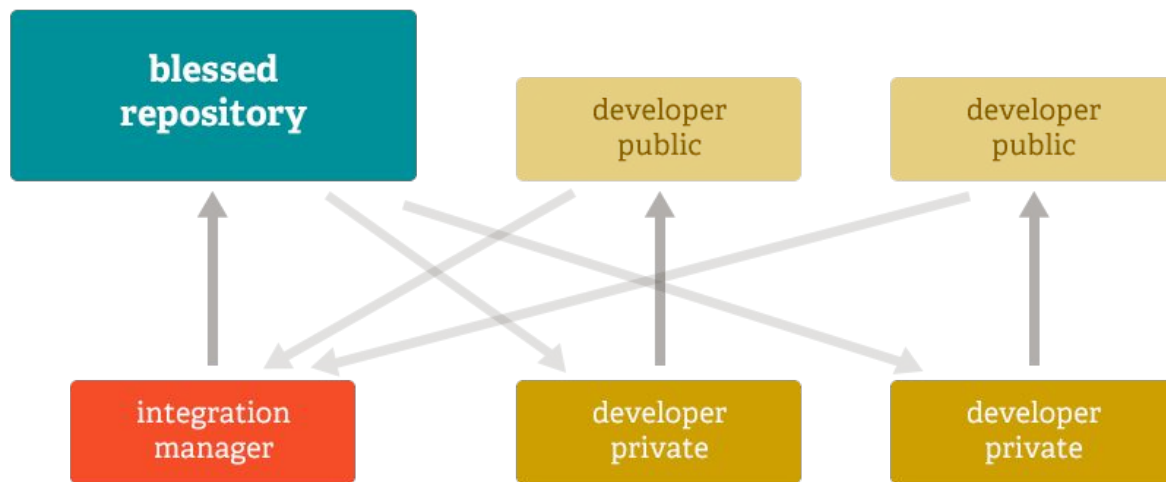
A centralized workflow is very common, especially from people transitioning from a centralized system. Git will not allow you to push if someone has pushed since the last time you fetched, so a centralized model where all developers push to the same server works just fine.





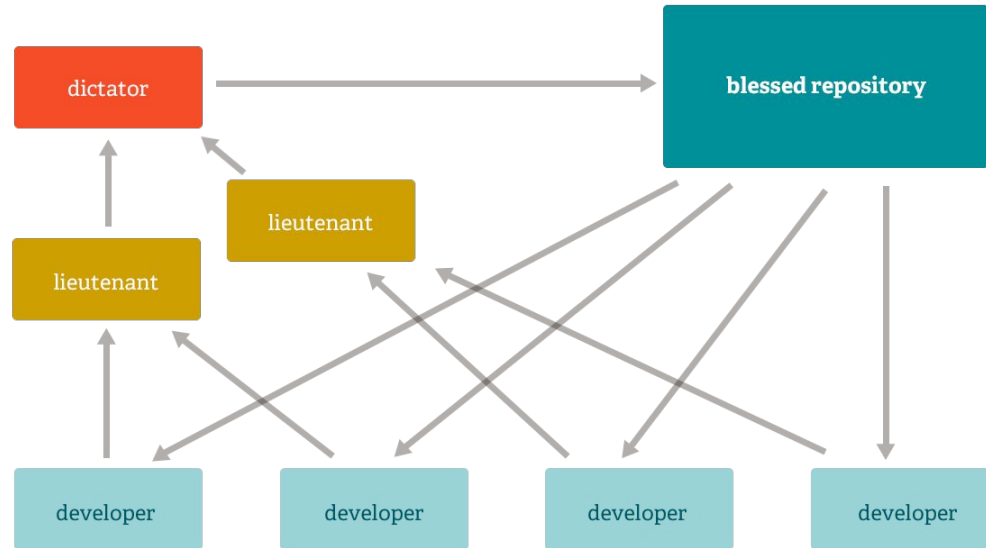
## Integration Manager Workflow

- Git workflow involves an integration manager — a single person who commits to the 'blessed' repository.
- A number of developers then clone from that repository, push to their own independent repositories, and ask the integrator to pull in their changes.
- This type of development model often used with open source or GitHub repositories.



## Dictator and Lieutenants Workflow

- For more massive projects, a development workflow like that of the Linux kernel is often effective.
- In this model, some people ('lieutenants') are in charge of a specific subsystem of the project and they merge in all changes related to that subsystem.
- Another integrator (the 'dictator') can pull changes from only his/her lieutenants and then push to the 'blessed' repository that everyone then clones from again.



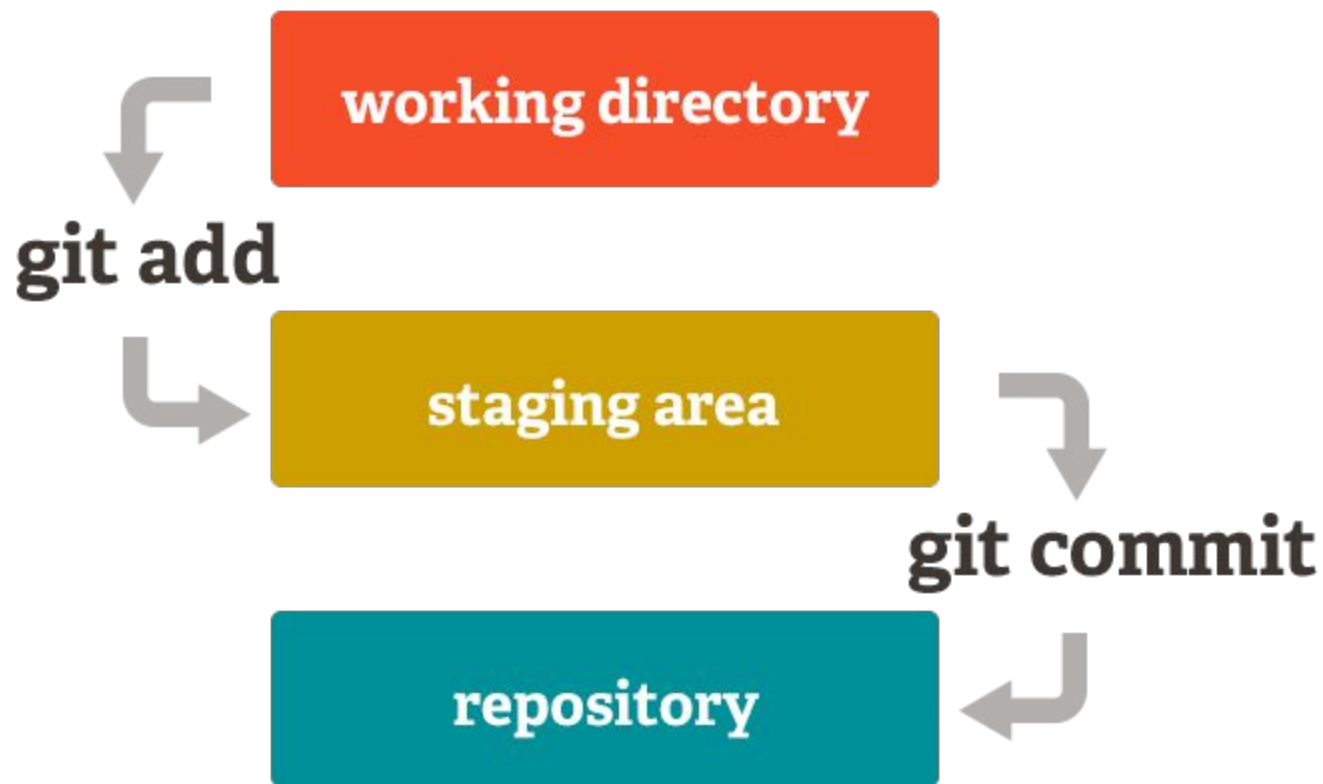
# Data Assurance

- The data model that Git uses ensures the cryptographic integrity of every bit of your project.
- Every file and commit is checksummed and retrieved by its checksum when checked back out. It's impossible to get anything out of Git other than the **exact bits you put in**.
- It is also impossible to change any file, date, commit message, or any other data in a Git repository without changing the IDs of everything after it.
- This means that if you have a commit ID, you can be assured not only that your project is exactly the same as when it was committed, but that nothing in its history was changed.
- Most centralized version control systems provide no such integrity by default.



## Staging Area

- Unlike the other systems, Git has something called the "**staging area**" or "**index**". This is an intermediate area where commits can be formatted and reviewed before completing the commit.
- One thing that sets Git apart from other tools is that it's possible to quickly stage some of your files and commit them without committing all of the other modified files in your working directory or having to list them on the command line during the commit.
- This allows you to stage only portions of a modified file. Gone are the days of making two logically unrelated modifications to a file before you realized that you forgot to commit one of them. Now you can just stage the change you need for the current commit and stage the other change for the next commit. This feature scales up to as many different changes to your file as needed.
- Of course, Git also makes it easy to ignore this feature if you don't want that kind of control — just add a '-a' to your commit command in order to add all changes to all files to the staging area.



**working directory**

```
graph TD; A[working directory] --> B[staging area]; B --> C[repository];
```

**staging area**

**git commit -a**

**repository**







































