

## Suraj Tiwari

MGSA\_088 HDFCLIFE FullStack Java Batch 01.

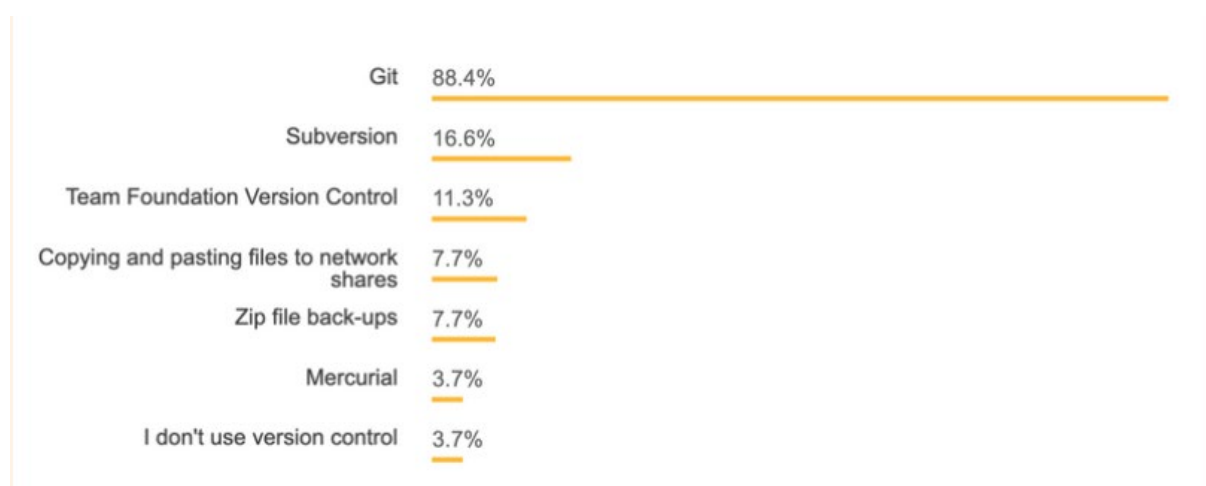
### [Git and GitHub using GitBash & Command Prompt.](#)

#### **Git & Version Control System**

The world's most popular version control system. Version Control, Version control is software that tracks and manages changes to files over time. Version control systems generally allow users to revisit earlier versions of the files, compare changes between versions, undo changes, and a whole lot more.

Git is Just one Version control system Git is just one of the many version control systems available today. Other well-known ones include Subversion, CVS, and Mercurial. They all have similar goals, but vary significantly in how they achieve those goals. Fortunately, we only need to care about Git because, Git is a clear winner

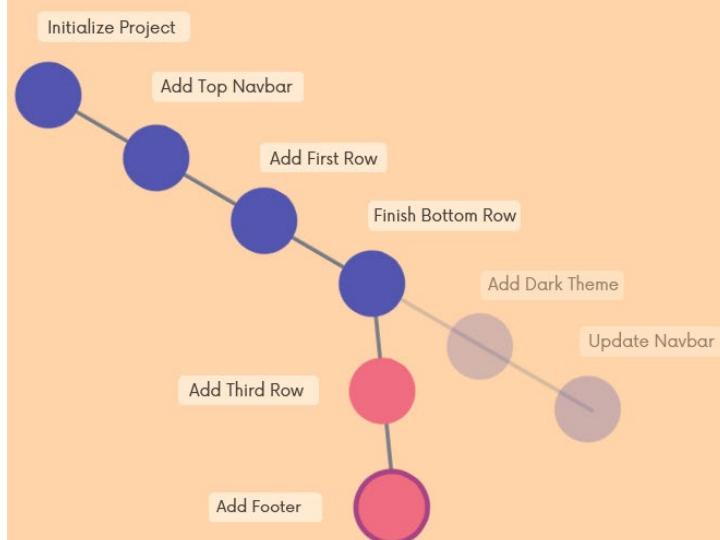
Stack Overflow's 2018 Developer Survey, nearly 90% of respondents reported Git as their version control system of choice. Over the last few years, the survey hasn't even bothered to ask about version control because Git is so widely used.



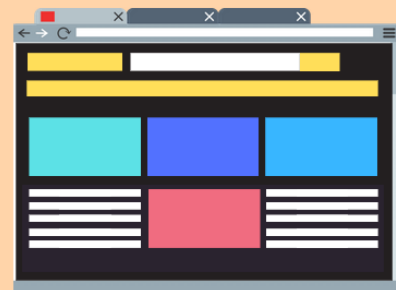
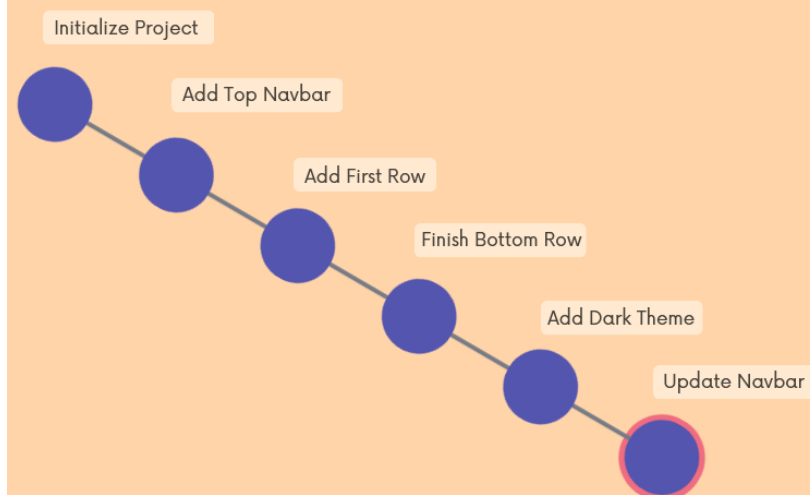
#### **Advantages of Git:**

- track changes across multiple files
- Compare versions of a project
- "Time travel" back to old versions
- Collaborate and share changes
- Combine changes.
- Revert to a previous version

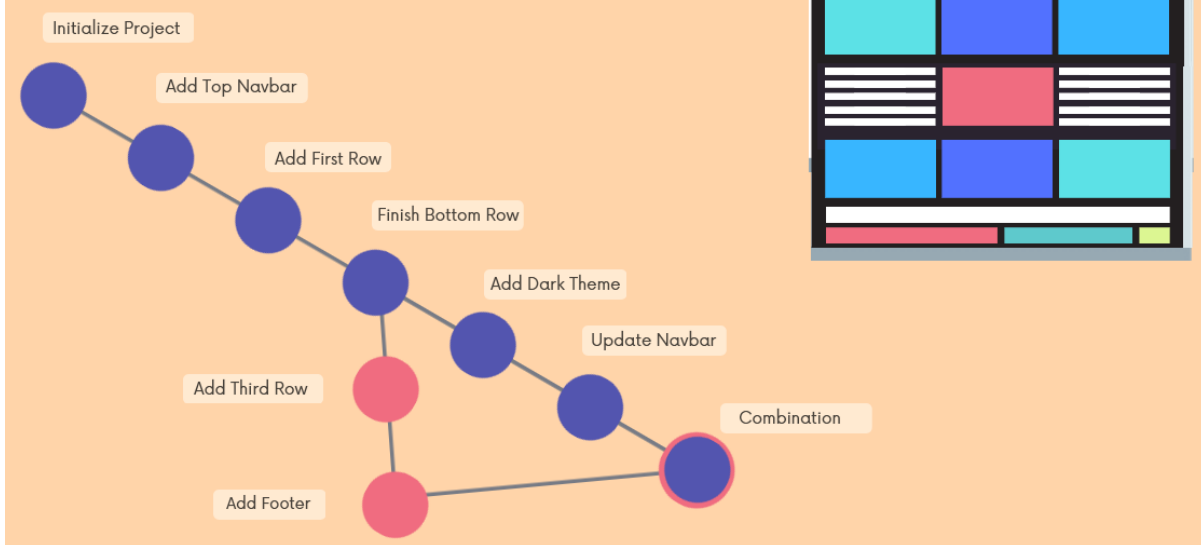
## Another checkpoint!



## Add A Checkpoint



And I can even combine checkpoints!



Git is not equal to github

Git  $\neq$  Github

Github is a service that hosts Git repositories in the cloud and makes it easier to collaborate with other people. You do need to sign up for an account to use Github. It's an online place to share work that is done using Git.

Repository

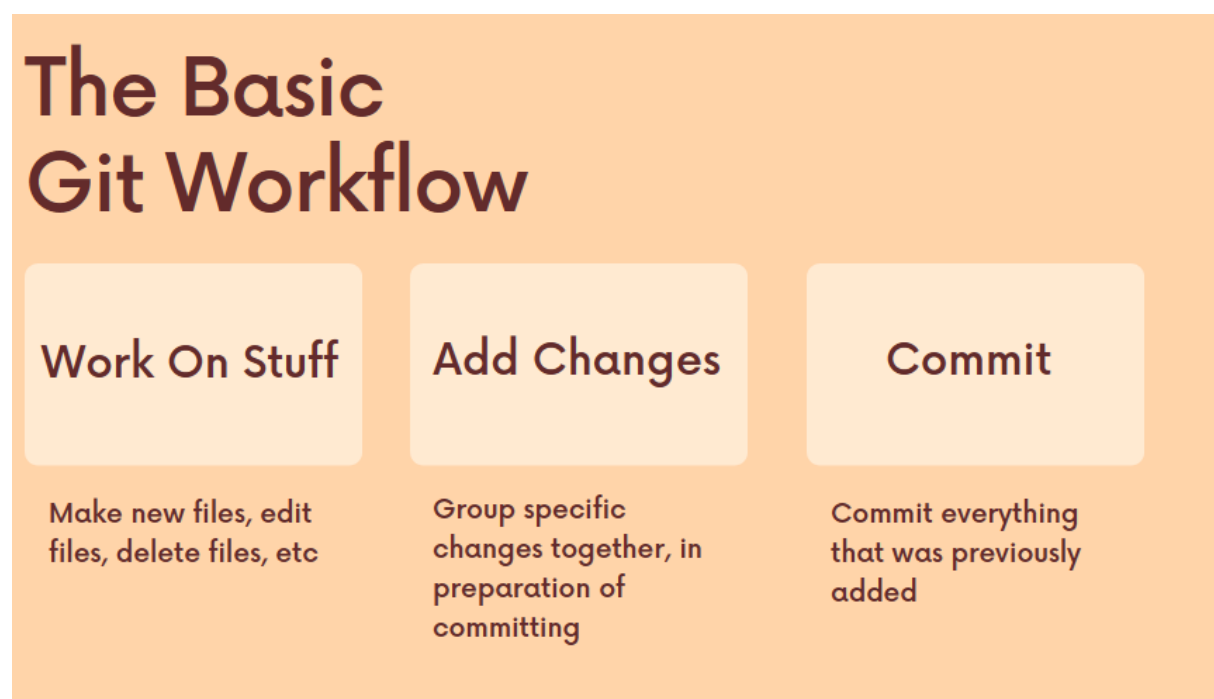
A Git "Repo" is a workspace which tracks and manages files within a folder.

Anytime we want to use Git with a project, app, etc we need to create a new git repository. We can have as many repos on our machine as needed, all with separate histories and contents.

## Committing

Making a commit is similar to making a save in a video game. We're taking a snapshot of a git repository in time.

When saving a file, we are saving the state of a single file. With Git, we can save the state of multiple files and folders together.



Git command

## Git status

git status gives information on the current status of a git repository and its contents It's very useful.

```
PC@SurajTiwari MINGW64 /f (master)
$ git status
warning: could not open directory 'Recovery/': Permission denied
warning: could not open directory 'System Volume Information/': Permission denied
On branch master
```

## Git init:

Use git init to create a new git repository. Before we can do anything git-related, we must initialize a repo first! This is something you do once per project. Initialize the repo in the top-level folder containing your project.

```
PC@SurajTiwari MINGW64 /f
$ git init
Initialized empty Git repository in F:/./.git/
```

## Git Add:

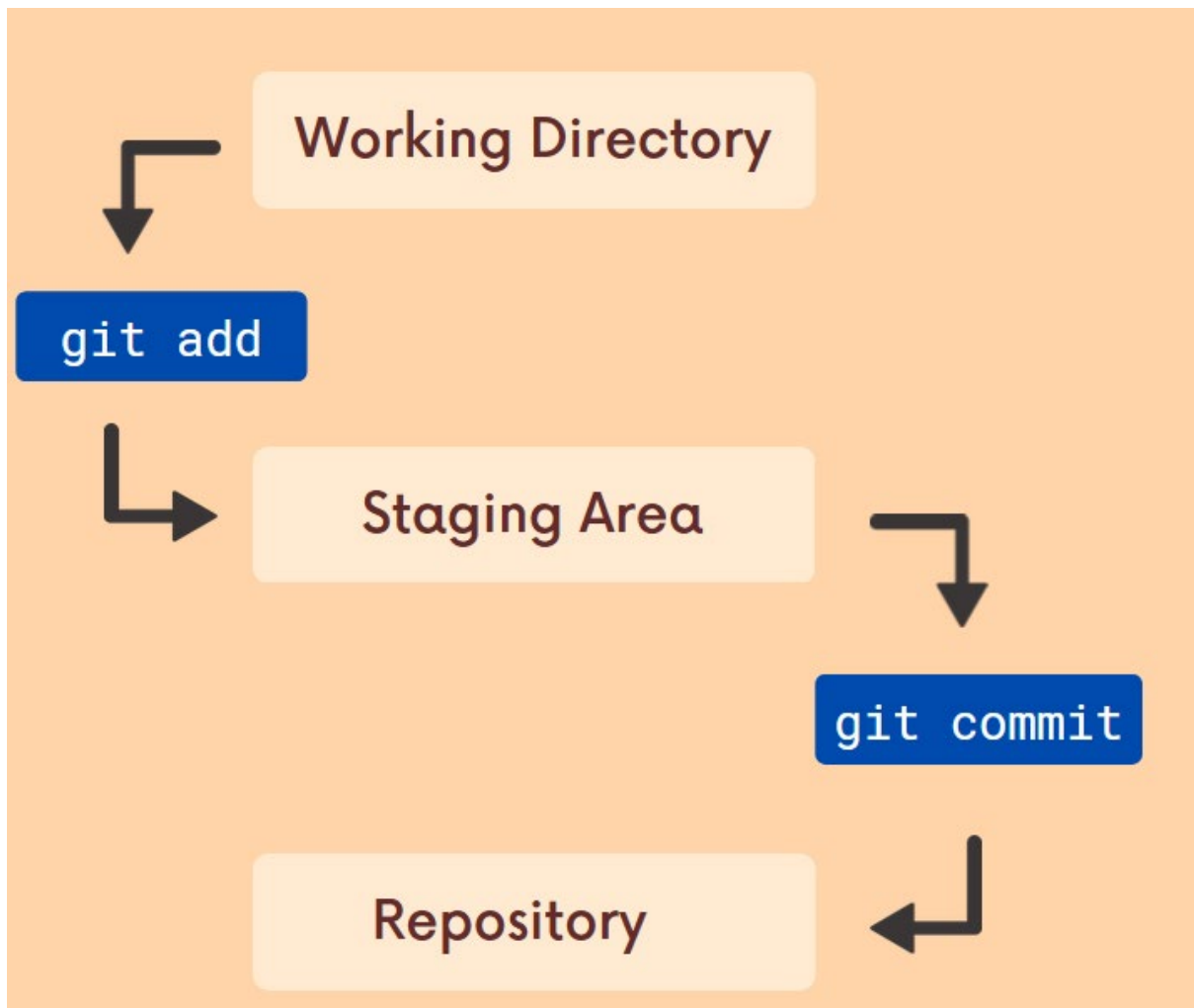
We use the git add command to stage changes to be committed. It's a way of telling Git, "please include this change in our next commit". Use git add to add specific files to the staging area. Separate files with spaces to add multiple at once. Use git add . to stage all changes at once

```
> git add file1 file2
```

## Git commit:

Running `git commit` will commit all staged changes. It also opens up a text editor and prompts you for a commit message. The `-m` flag allows us to pass in an inline commit message, rather than launching a text editor.

```
> git commit -m "my message"
```



### Git Ignore:

We can tell Git which files and directories to ignore in a given repository, using a `.gitignore` file.

This is useful for files you know you NEVER want to commit, including:

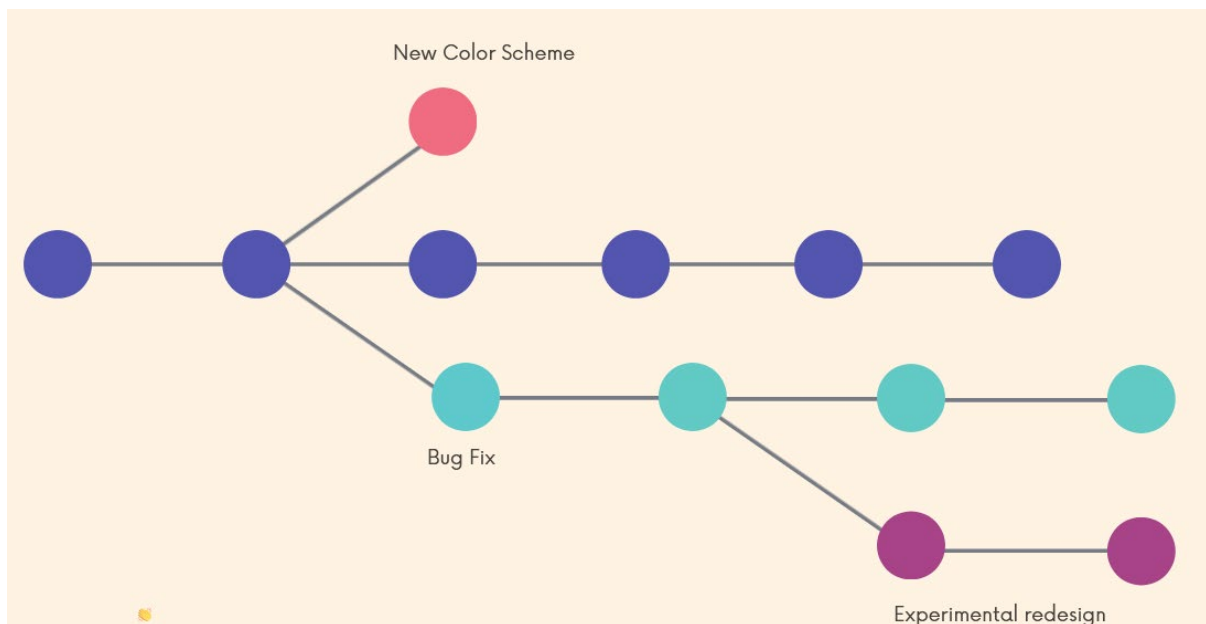
- Secrets, API keys, credentials, etc.
- Operating System files (.DS\_Store on Mac)
- Log files
- Dependencies & packages

Create a file called `.gitignore` in the root of a repository. Inside the file, we can write patterns to tell Git which files & folders to ignore:

- `.DS_Store` will ignore files named `.DS_Store`
- `folderName/` will ignore an entire directory
- `*.log` will ignore any files with the `.log` extension

## Branches

Branches are an essential part of Git! Think of branches as alternative timelines for a project. They enable us to create separate contexts where we can try new things, or even work on multiple ideas in parallel. If we make changes on one branch, they do not impact the other branches (unless we merge the changes)



### Viewing Branches

Use `git branch` to view your existing branches. The default branch in every git repo is `master`, though you can configure this.

Look for the \* which indicates the branch you are currently on.

```
> git branch
```

## Git remote:

Once everything is ready on our local system, we can start pushing our code to the remote (central) repository

```
PC@SurajTiwarei MINGW64 /f (master)
$ git remote add origin https://github.com/Suraj-tiwari-github/example.git

PC@SurajTiwarei MINGW64 /f (master)
$
```

## Git Push

By using the command git push the local repository's files can be synced with the remote repository on Github.

```
PC@SurajTiwarei MINGW64 /f (master)
$ git remote add origin https://github.com/Suraj-tiwari-github/example.git

PC@SurajTiwarei MINGW64 /f (master)
$ git push -u origin main
```

## Git reset

We use this command to return the entire working tree to the last committed state.

```
PC@SurajTiwarei MINGW64 /f (master)
$ git reset
```



## Git Diff

We can use the git diff command to view changes between commits, branches, files, our working directory, and more! .We often use git diff alongside commands like git status and git log, to get a better picture of a repository and how it has changed over time. Without additional options, git diff lists all the changes in our working directory that are NOT staged for the next commit.

```
> git diff
```

```
diff --git a/rainbow.txt b/rainbow.txt
index 72d1d5a..f2c8117 100644
--- a/rainbow.txt
+++ b/rainbow.txt
@@ -3,4 +3,5 @@ orange
yellow
green
blue
-purple
+indigo
+violet
```

## Git Stashing:

Git provides an easy way of stashing these uncommitted changes so that we can return to them later, without having to make unnecessary commits. git stash is super useful command that helps you save changes that you are not yet ready to commit. You can stash changes and then come back to them later.

Running git stash will take all uncommitted changes (staged and unstaged) and stash them, reverting the changes in your working copy.

```
> git stash
```

```
> git stash list
stash@{0}: WIP on master: 049d078 Create index file
stash@{1}: WIP on master: c264051 Revert "Add file_size"
stash@{2}: WIP on master: 21d80a5 Add number to log
```

## Unstaging Files with Restore

If you have accidentally added a file to your staging area with git add and you don't wish to include it in the next commit, you can use git restore to remove it from staging.

Use the --staged option like this:

```
git restore --staged app.js
```

```
> git restore --staged <file-name>
```

```
GitDemo > git status                                timetravel
On branch timetravel
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   cat2.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   cat2.txt
```

## Git Reset:

Suppose you've just made a couple of commits on the master branch, but you actually meant to make them on a separate branch instead. To undo those commits, you can use git reset

git reset <commit-hash> will reset the repo back to a specific commit. The commits are gone

```
> git reset --hard <commit>
```

## What is GitHub:

Github is a hosting platform for git repositories. You can put your own Git repos on Github and access them from anywhere and share them with people around the world.

Beyond hosting repos, Github also provides additional collaboration features that are not native to Git (but are super useful). Basically, Github helps people share and collaborate on repos. There are tons of competing tools that provide similar hosting and collaboration features, including GitLab, BitBucket, and Gerrit. Founded in 2008, Github is now the world's largest host of source code. In early 2020, Github reported having over 40 million users and over 190 million repositories on the platform. Github offers its basic services for free!

While Github does offer paid Team and Enterprise tiers, the basic Free tier allows for unlimited public and private repos, unlimited collaborators, and more!

## Git clone

To clone a repo, simply run git clone <url>.

Git will retrieve all the files associated with the repository and will copy them to your local machine.

In addition, Git initializes a new repository on your machine, giving you access to the full Git history of the cloned project.

```
> git clone <url>
```

Option 1: Existing Repo pushing.

if you already have an existing repo locally that you want to get on Github...

- Create a new repo on Github
- Connect your local repo (add a remote)
- Push up your changes to Github

Option 2: Start From Scratch

if you haven't begun work on your local repo, you can...

- Create a brand new repo on Github
- Clone it down to your machine
- Do some work locally
- Push up your changes to Github

## Pushing:

To get your own changes and Git history up on Github, we need to PUSH them up. The typical workflow looks something like this:

- Make some changes locally
- Add and commit those changes
- Repeat...
- Push new commits up to Github
- First we need to make a Repo On Github

## Remote:

Before we can push anything up to Github, we need to tell Git about our remote repository on Github. We need to setup a "destination" to push up to.

In Git, we refer to these "destinations" as remotes. Each remote is simply a URL where a hosted repository lives. To view any existing remotes for your repository, we can run `git remote` or `git remote -v` (verbose, for more info) This just displays a list of remotes. If you haven't added any remotes yet, you won't see anything.

```
> git remote -v
```

```
> git remote add origin  
https://github.com/blah/repo.git
```

## Pushing:

Now that we have a remote set up, let's push some work up to Github! To do this, we need to use the `git push` command.

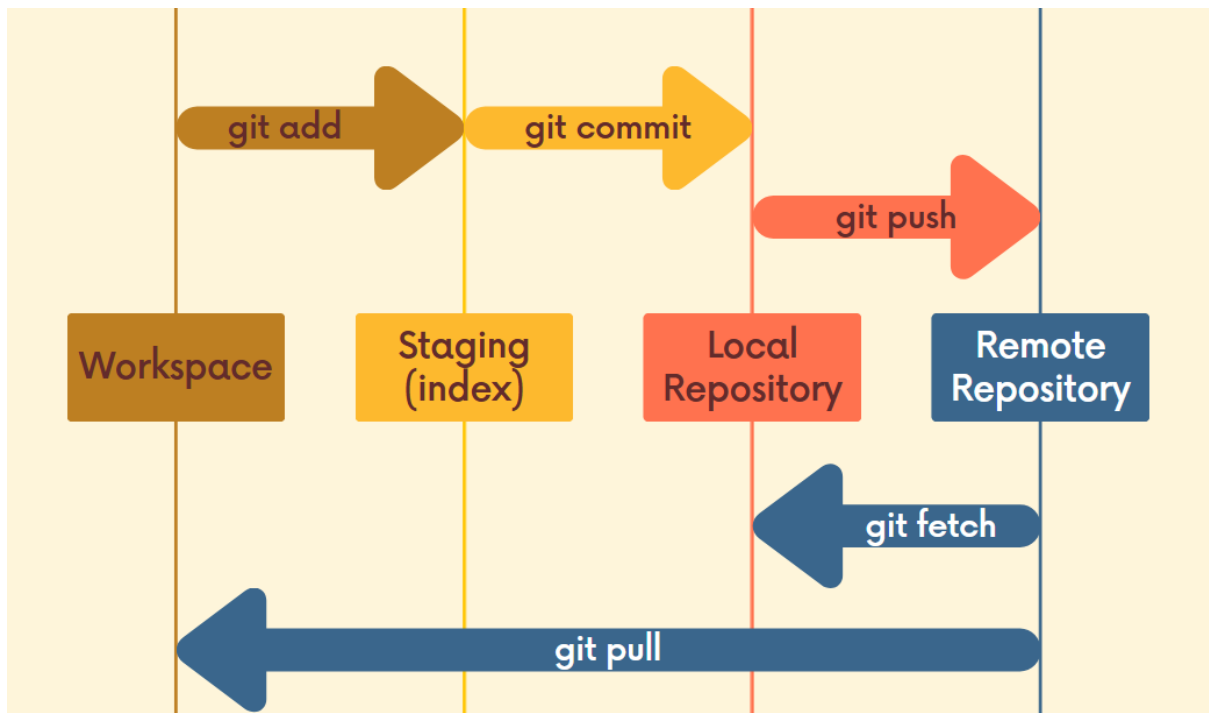
We need to specify the remote we want to push up to AND the specific local branch we want to push up to that remote.

```
PC@SurajTiwarei MINGW64 /f (master)
$ git remote add origin https://github.com/Suraj-tiwari-github/example.git

PC@SurajTiwarei MINGW64 /f (master)
$ █

PC@SurajTiwarei MINGW64 /f (master)
$ git remote add origin https://github.com/Suraj-tiwari-github/example.git

PC@SurajTiwarei MINGW64 /f (master)
$ git push -u origin main
```



## Fetching:

Fetching allows us to download changes from a remote repository, BUT those changes will not be automatically integrated into our working files. It lets you see what others have been working on, without having to merge those changes into your local repo.

The `git fetch <remote>` command fetches branches and history from a specific remote repository. It only updates remote tracking branches.

`git fetch origin` would fetch all changes from the origin remote repository.

```
> git fetch <remote>
```

## Git Pull

`git pull` is another command we can use to retrieve changes from a remote repository. Unlike `fetch`, `pull` actually updates our HEAD branch with whatever changes are retrieved from the remote.

**git pull = git fetch + git merge**

update the remote tracking branch with the latest changes from the remote repository      update my current branch with whatever changes are on the remote tracking branch

To pull, we specify the particular remote and branch we want to pull using `git pull <remote> <branch>`. Just like with `git merge`, it matters WHERE we run this command from. Whatever branch we run it from is where the changes will be merged into.


`git pull origin master` would fetch the latest information from the origin's master branch and merge those changes into our current branch.





# git fetch

- Gets changes from remote branch(es)
- Updates the remote-tracking branches with the new changes
- Does not merge changes onto your current HEAD branch
- Safe to do at anytime

# git pull



- Gets changes from remote branch(es)
- Updates the current branch with the new changes, merging them in
- Can result in merge conflicts
- Not recommended if you have uncommitted changes!

 Suraj-tiwari-github / ShellScript\_Files Public

 Pin  Unwatch 1  Fork 0  Star 0

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)


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






 Set up in Desktop or [HTTPS](#) [SSH](#) `https://github.com/Suraj-tiwari-github/ShellScript_Files.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 "# ShellScript_Files" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Suraj-tiwari-github/ShellScript_Files.git
git push -u origin main
```



 PHP	21-May-20 4:13 PM	File folder	
 python	16-Feb-21 5:25 PM	File folder	
 shellScript	29-Jun-22 1:53 PM	File folder	
 tmp	30-Nov-21 3:56 PM	File folder	
 name.txt	08-Jul-22 3:51 PM	Text Document	0 KB



```
PC@SurajTiwari MINGW64 /f/shellScript (master)
$ git init
Initialized empty Git repository in F:/shellScript/.git/







PC@SurajTiwari MINGW64 /f/shellScript (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        array.sh
        firstScript.sh
        ifelse.sh
        loops.sh
        switch.sh

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

PC@SurajTiwari MINGW64 /f/shellScript (master)
$
```

Name	Date modified	Type	Size
 .git	13-Jul-22 10:34 PM	File folder	
 array.sh	29-Jun-22 11:48 AM	Shell Script	1 KB
 firstScript.sh	29-Jun-22 3:44 PM	Shell Script	1 KB
 ifelse.sh	29-Jun-22 11:30 AM	Shell Script	1 KB
 loops.sh	29-Jun-22 2:25 PM	Shell Script	1 KB
 switch.sh	29-Jun-22 11:42 AM	Shell Script	1 KB

```
PC@SurajTiwari MINGW64 /f/shellScript (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file> ..." to include in what will be committed)
        array.sh
        firstScript.sh
        ifelse.sh
        loops.sh
        switch.sh

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

PC@SurajTiwari MINGW64 /f/shellScript (master)
$ git add .
warning: LF will be replaced by CRLF in array.sh.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in firstScript.sh.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in ifelse.sh.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in loops.sh.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in switch.sh.
The file will have its original line endings in your working directory
```


```
PC@SurajTiwari MINGW64 /f/shellScript (master)
$ git commit -m "Initial Commit By Suraj"
[master (root-commit) 4fd18e3] Initial Commit By Suraj
 5 files changed, 238 insertions(+)
 create mode 100644 array.sh
 create mode 100644 firstScript.sh
 create mode 100644 ifelse.sh
 create mode 100644 loops.sh
 create mode 100644 switch.sh
```






```
PC@SurajTiwari MINGW64 /f/shellScript (master)
$ git status
On branch master
nothing to commit, working tree clean
```

```
PC@SurajTiwari MINGW64 /f/shellScript (master)
$ git remote add origin https://github.com/Suraj-tiwari-github/ShellScript_Files.git

PC@SurajTiwari MINGW64 /f/shellScript (master)
$ git push origin master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 1.97 KiB | 1.97 MiB/s, done.
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Suraj-tiwari-github/ShellScript_Files.git
 * [new branch]      master -> master
```

master ▾ 1 branch 0 tags Go to file Add file ▾ Code ▾

 **SurajTiwari1155** Initial Commit By Suraj 4fd18e3 3 minutes ago 🕒 1 commit

 array.sh	Initial Commit By Suraj	3 minutes ago
 firstScript.sh	Initial Commit By Suraj	3 minutes ago
 ifelse.sh	Initial Commit By Suraj	3 minutes ago
 loops.sh	Initial Commit By Suraj	3 minutes ago
 switch.sh	Initial Commit By Suraj	3 minutes ago

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

Add a README