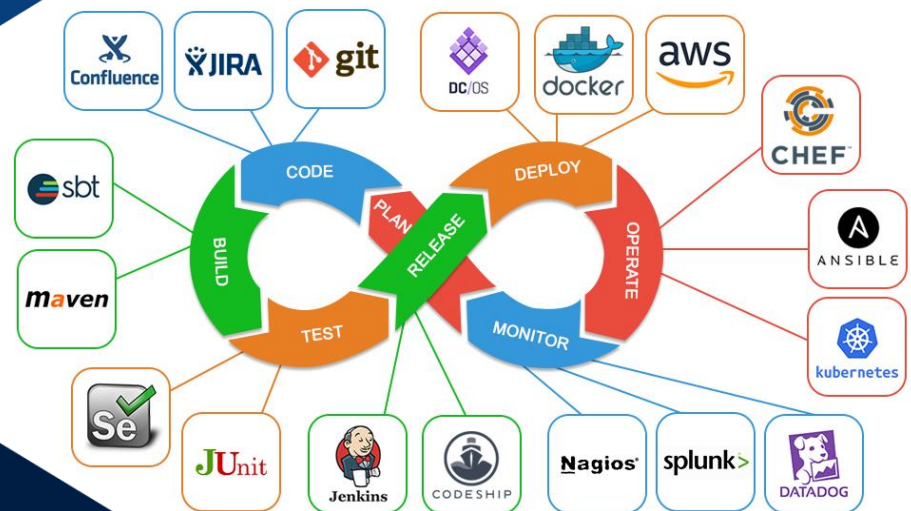




SOUTHERN LEYTE
STATE UNIVERSITY
BONTOC

San Ramon, Bontoc, Southern Leyte, 6604 Philippines
Telefax No.: (053) 382-3121; E-mail: slsubc@yahoo.com
Website: www.slsuonline.edu.ph



System Integration and Architecture 1

LEARNING GUIDE 2020

REXAL S. TOLEDO



SOUTHERN LEYTE STATE UNIVERSITY-BONTOC

This learning module is developed for instructional purposes only. Any form of reproduction or distribution is strictly prohibited.



Southern Leyte State University

Vision

A high quality corporate University of Science, Technology and Innovation.

Mission

SLSU will:

- a) Develop Science, Technology, and Innovation leaders and professionals;*
- b) Produce high-impact technologies from research and innovations;*
- c) Contribute to sustainable development through responsive community engagement programs;*
- d) Generate revenues to be self-sufficient and financially-viable.*

Quality Policy

We at Southern Leyte State University commit enthusiastically to satisfy our stakeholders' needs and expectations by adhering to good governance, relevance and innovations of our instruction, research and development, extension and other support services and to continually improve the effectiveness of our Quality Management System in compliance to ethical standards and applicable statutory, regulatory, industry and stakeholders' requirements.

The management commits to establish, maintain and monitor our quality management system and ensure that adequate resources are available.

COURSE OVERVIEW

Course No. IT303/IT303L

Course Code

Descriptive Title System Integration and Architecture 1

Credit Units 2 units (Lecture) / 1 unit (Laboratory)

School Year/Term 2020-2021 / 1st semester

Mode of Delivery

Name of Instructor Rexal S. Toledo

Course Description The course covers the role of systems architecture in systems integration, performance, and effectiveness. Principles and concepts of DevOps. Interplay between IT applications roll-out and related organizational processes.

Course Outcomes

1. Analyze the appropriateness of a decision to in-source of out-source IT services in a given situation.
2. Create a testing environment and design a stress test using appropriate tools and techniques that impact system performance.
3. Implement an enterprise integration middleware platform
4. Complete a requirements gathering and modeling exercise for a typical IT project
5. Describe how systems architecture directly affects the system lifecycle

TABLE OF CONTENTS

	PAGE
PRELIMINARIES	
Cover Page	i
Disclaimer	ii
SLSU Vision, Mission, Quality Policy	iii
Course Overview	iv
Module Guide	v
Table of Contents	vi
 MODULE I Introduction	
LESSON	
1 Overview of System Integration and Architecture	
2 Continuous Delivery and Continuous Integration	
3 DevOps vs Agile: What's the difference?	
 MODULE II DevOps	
LESSON	
4 DevOps Overview	
5 DevOps Architecture	
6 DevOps Tools	
 MODULE III DevOps Tools: Repository and Git Version Control	
LESSON	
7 Introduction to Source Control and GIT Version Control	
8 A Thorough Guide to Basic Git Commands and the Command-line Interface	

9 Introduction to GitHub

INTRODUCTION

Many systems are built to easy, improve and transform organizations. Some organizations have many departments which run systems which are independent of each other. And systems built sometimes, may not have an abstract view (architecture) which leads to failure of system interoperability. There is need to have architectural view of the system as a priority to help in the design to avoid the likeliness of system failure. Besides after the system has been designed and developed in consideration of the size of the organization, i.e. most especially when the organization is large, need is required to integrate such systems to ensure flexibility, Speed, Cost, Standardization, Data integrity, reliability and robustness. This can help Information Technology (IT) industry among others to have an easy to use integrated system.

MODULE

3

DevOps Tools: Repository and Git Version Control

LESSON

- 1 Introduction to Source Control and GIT
Version Control
- 2 A Thorough Guide to Basic Git Commands
and the Command-line Interface
- 3 Introduction to GitHub

- D. `git commit -m "I'm coding"`
8. What is the correct commit syntax for all changes with a message?
 - A. `git message -am "I'm coding"`
 - B. `git add -a "I'm coding"`
 - C. `git commit -a "I'm coding"`
 - D. `git commit -am "I'm coding"`
 9. Which git command comes first to push your changes?
 - A. `git commit -m "changed image to button"`
 - B. `git push`
 - C. `git push & git add .`
 - D. `git add .`
 10. What's git command should you use to initialize a new Git repository?
 - A. `git new`
 - B. `git init`
 - C. `git initialize`
 - D. `git add .`
 11. git command to switch branch or change active branch.
 - A. `git switch <branch-name>`
 - B. `git check <branch-name>`
 - C. `git checkout <branch-name>`
 - D. `git checkingout <branch- name>`
 12. How to show remote repositories name and url for push/fetch.
 - A. `git remote -v`
 - B. `git remote -r`
 - C. `git remote showlist`
 - D. `git remote -m`

TEST II: Arrange the following basic git commands in the correct order.

1. Pushing changes:
 - A. `git push`
 - B. `git add homepage.php`
 - C. `git commit -m "added home button"`
2. Adding an existing project to GitHub.
 - A. `git commit -m "first commit"`
 - B. `git add .`
 - C. `git init`
 - D. `git remote add origin < your repository url >`
 - E. `git push -u origin master`

TEST III: Essay

1. What is Git version control?
2. What are the benefits of Version Control System?
3. What is a Distributed System?
4. What are the differences between Git and GitHub?

LESSON

7

Introduction to Source Control and GIT Version Control

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Understand History and fundamental concepts behind source control
2. Differentiate Centralized vs. distributed version control
3. Understand the motivations for using version control.

What is a 'version control system'?

- a way to manage files and directories
- track changes over time
- recall previous versions
- 'source control' is a subset of a VCS

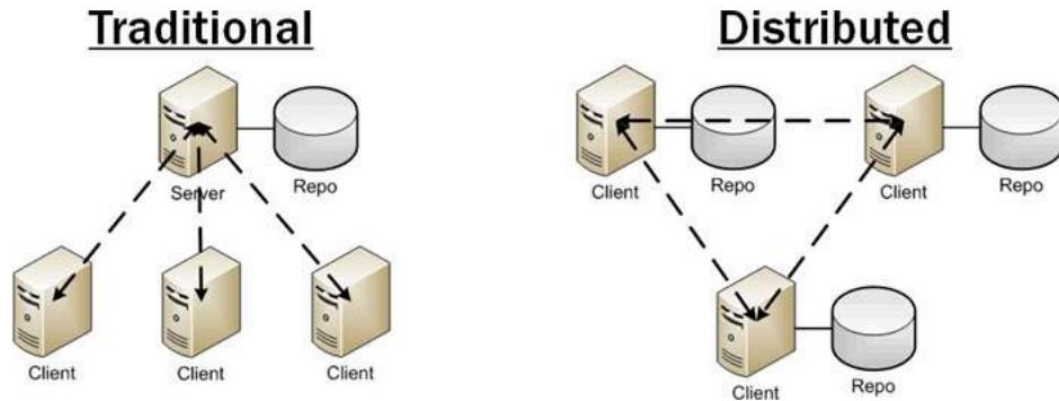
Some history of source control

- (1972) Source Code Control System (SCCS)
 - closed source, part of UNIX
- (1982) Revision Control System(RCS)
 - open source
- (1986) Concurrent Versions System (CVS)
 - open source
- (2000) Apache Subversion (SVN)
 - open source
- (2000) BitKeeper SCM
 - closed source, proprietary, used with source code management of Linux kernel
 - free until 2005
 - distributed version control

Distributed version control

No central server

Every developer is a client, the server and the repository



Source: <http://bit.ly/1SH4E23>

What is git?

Created by Linus Torvalds, April 2005

- replacement for BitKeeper to manage Linux kernel changes
- a command line version control program
- uses checksums to ensure data integrity
- distributed version control (like BitKeeper)
- Cross-platform (including Windows!)
- open source, free

Git distributed version control

- "If you're not distributed, you're not worth using." – Linus Torvalds
- no need to connect to central server
- can work without internet connection
- no single failure point
- developers can work independently and merge their work later
- every copy of a Git repository can serve either as the server or as a client (and has complete history!)
- Git tracks changes, not versions
- Bunch of little change sets floating around

Is Git for me?

- People primarily working with source code
- Anyone wanting to track edits (especially changes to text files)
 - review history of changes

- anyone wanting to share, merge changes
- Anyone not afraid of
- command line tools

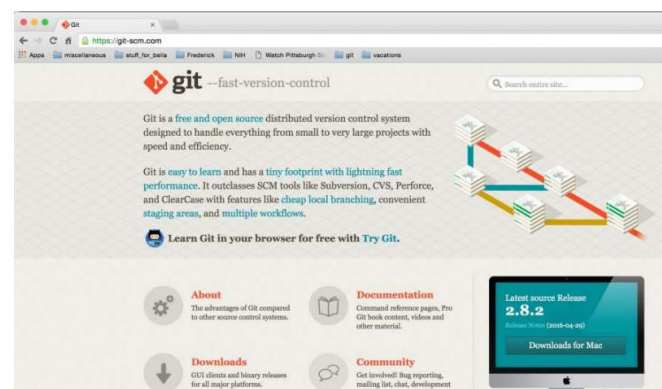
Most popular languages used with Git

- | | |
|-----------------|----------------|
| • HTML | • Perl |
| • CSS | • Java |
| • Javascript | • C |
| • Python | • C++ |
| • ASP | • C# |
| • Scala | • Objective C |
| • Shell scripts | • Haskell |
| • PHP | • CoffeeScript |
| • Ruby | • ActionScript |
| • Ruby on Rails | |

Not as useful for image, movies, music...and files that must be interpreted (.pdf, .psd, etc.)

How do I get it?

<https://git-scm.com/>



Git install tip

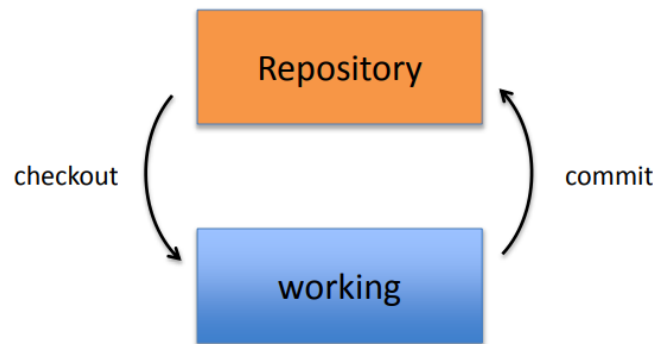
Much better to set up on a per-user basis (instead of a global, system-wide install)

What is a repository?

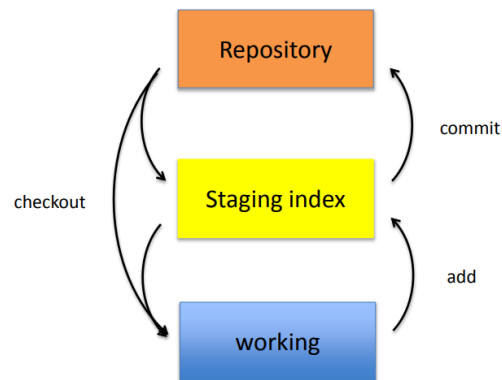
- "repo" = repository
- usually used to organize a single project
- repos can contain folders and files, images,
- videos, spreadsheets, and data sets – anything
- your project needs

Two-tree architecture

other VCSs



Git uses a three-tree architecture



A simple Git workflow

1. Initialize a new project in a directory:

`git init`

```
[ dolanmi L02029756 ~/Desktop ]$ mkdir new_project
[ dolanmi L02029756 ~/Desktop ]$ cd new_project/
[ dolanmi L02029756 ~/Desktop/new_project ]$ git init
Initialized empty Git repository in /Users/dolanmi/Desktop/new_project/.git/
[ dolanmi L02029756 ~/Desktop/new_project ]$
```

2. Add a file using a text editor to the directory
3. Add every change that has been made to the directory:

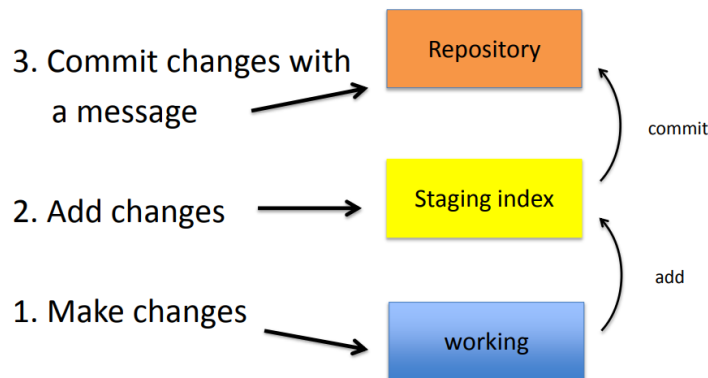
`git add .`

4. Commit the change to the repo:

`git commit -m "important message here"`

```
[ dolanmi L02029756 ~/Desktop/new_project ]$ git add .
[ dolanmi L02029756 ~/Desktop/new_project ]$ git commit -m "Add message to file.txt"
[master (root-commit) 1a7e4a5] Add message to file.txt
1 file changed, 1 insertion(+)
create mode 100644 file.txt
[ dolanmi L02029756 ~/Desktop/new_project ]$
```

After initializing a new git repo...



A note about commit messages

- Tell what it does (present tense)
- Single line summary followed by blank space
- followed by more complete description
- Keep lines to ≤ 72 characters
- Ticket or bug number helps

Good and bad examples

Bad: "Typo fix"

Good: "Add missing / in CSS section"

Bad: "Updates the table. We'll discuss next Monday with Darrell."

Bad: `git commit -m "Fix login bug"`

Good: `git commit -m`

```
Redirect user to the requested page after login

https://trello.com/path/to/relevant/card

Users were being redirected to the home page after login, which is less
useful than redirecting to the page they had originally requested before
being redirected to the login form.

* Store requested path in a session variable
* Redirect to the stored location after successfully logging in the user
```

How to I see what was done?

`git log`

```
[ dolanmi L02029756 ~/Desktop/new_project ]$ git log
commit 6c40ffd9ba4ba1567eb6fcd3715f12a15b0a678d
Author: mchldln <dolanmi@niaid.nih.gov>
Date:   Mon May 2 18:11:23 2016 -0400

    Add message to text file
[ dolanmi L02029756 ~/Desktop/new_project ]$
```

```

[dolanmi L02029756 ~/Desktop/bcbb/portal_project/git/BCB8portalXI]$ git log
commit f8c0639a649a122446040b15185cc09c4c5c71c
Author: Yamil Boo <yamil.booirizarry@nih.gov>
Date:   Fri Apr 29 15:02:56 2016 -0400

    update headers

commit eb0cf49cc05786cbc7314982f06af5a9ad93149e
Author: Yamil Boo <yamil.booirizarry@nih.gov>
Date:   Tue Apr 26 12:07:32 2016 -0400

    update name link and about page

commit 44c433a1794cfef211d5116560dcf6e67d518b2f
Author: Yamil Boo <yamil.booirizarry@nih.gov>
Date:   Mon Apr 25 15:45:27 2016 -0400

    remove about, change font family in the name

commit 808be0093a995c08a7a4f99219abec255b94a874
Author: Yamil Boo <yamil.booirizarry@nih.gov>
Date:   Fri Apr 22 09:30:49 2016 -0400

    updating header and sidenav bar

commit c5f689ed0b8c71582b3d301e2282f9e6472962c6
Author: Yamil Boo <yamil.booirizarry@nih.gov>
Date:   Thu Apr 21 14:29:20 2016 -0400

    change the name to code

commit 4463ea2d1c75b80af9d2894feb2eb3ded7fe40c9
:
commit f8c0639a649a122446040b15185cc09c4c5c71c
Author: Yamil Boo <yamil.booirizarry@nih.gov>
Date:   Fri Apr 29 15:02:56 2016 -0400

    update headers

commit eb0cf49cc05786cbc7314982f06af5a9ad93149e
Author: Yamil Boo <yamil.booirizarry@nih.gov>
Date:   Tue Apr 26 12:07:32 2016 -0400

    update name link and about page

```

Commits
generated by
SHA1
encryption
algorithm

The HEAD pointer

- points to a specific commit in repo
- as new commits are made, the pointer changes
- HEAD always points to the "tip" of the currently checked-out branch in the repo (not the working directory or staging index)
- last state of repo (what was checked out initially)
- HEAD points to parent of next commit (where writing the next commit takes place)

LESSON

8

A Thorough Guide to Basic Git Commands and the Command-line Interface

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Learn Git Version Control and its basic commands.

Git Commands

Git is a free, open-source distributed version control system tool designed to handle small to very large projects with speed and efficiency. It has steadily grown from just being a preferred skill to a must-have skill for multiple job roles today. Git has become an essential part of our everyday development process.

In this Git commands tutorial, let's talk about the top 18 Git commands that are useful for working with Git.

Here are the top Git commands discussed in this tutorial:

git init	git log
git add	git stash
git commit	git revert
git status	git diff
git remote	git merge
git push	git rebase
git clone	git fetch
git branch	git reset
git checkout	git pull

So, let's get started!

1. git init


Usage: git init [repository name]

We have to navigate to our project directory and type the command git init to initialize a Git repository for our local project folder. Git will create a hidden .git directory and use it for keeping its files organized in other subdirectories.

2. git add

Usage (i): git add [file(s) name]

This will add the specified file(s) into the Git repository, the staging area, where they are already being tracked by Git and now ready to be committed.

 MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1
$ cd


intellipaat@DESKTOP-SPC6JQB MINGW64 ~
$ cd ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1
$ git init
Initialized empty Git repository in C:/Users/intellipaat/ProjectGit1/.git/

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ |
```

Usage (ii): git add . or git add *

This will take all our files into the Git repository, i.e., into the staging area.


 MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git add .
warning: LF will be replaced by CRLF in 234.txt.
The file will have its original line endings in your working directory.
```

3. git commit

Usage: git commit -m "message"

This command records or snapshots files permanently in the version history. All the files, which are there in the directory right now, are being saved in the Git file system.

 MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git commit -m "Committing 234master.txt in master"
[master (root-commit) be73fc3] Committing 234master.txt in master
2 files changed, 2 insertions(+)
create mode 100644 123master.txt
create mode 100644 234master.txt
```

4. git status

Usage: git status

This command will show the modified status of an existing file and the file addition status of a new file, if any, that have to be committed.

MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   234.txt
        new file:   ls
```

5. git remote

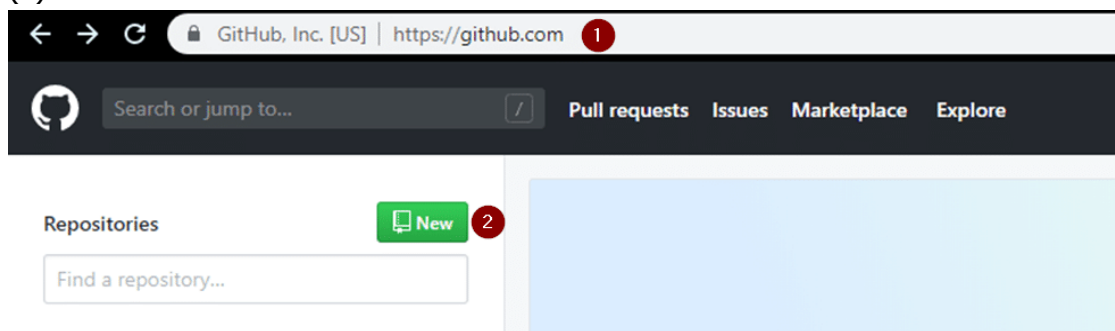
Usage: git remote add origin "[URL]"

Once everything is ready on our local system, we can start pushing our code to the remote (central) repository of the project. For that, follow the below steps:

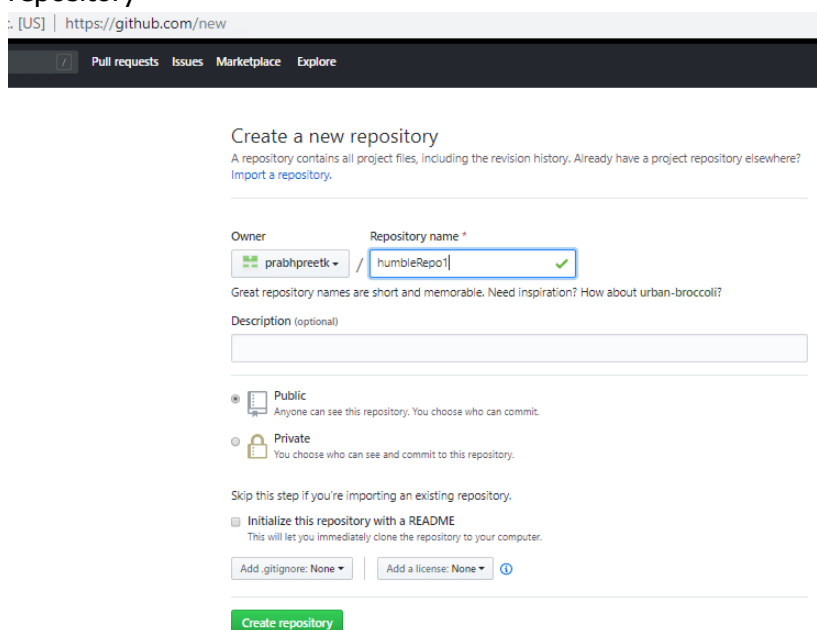
Step 1:

(1) Login to the GitHub account if the account already exists (If not, sign up on github.com)

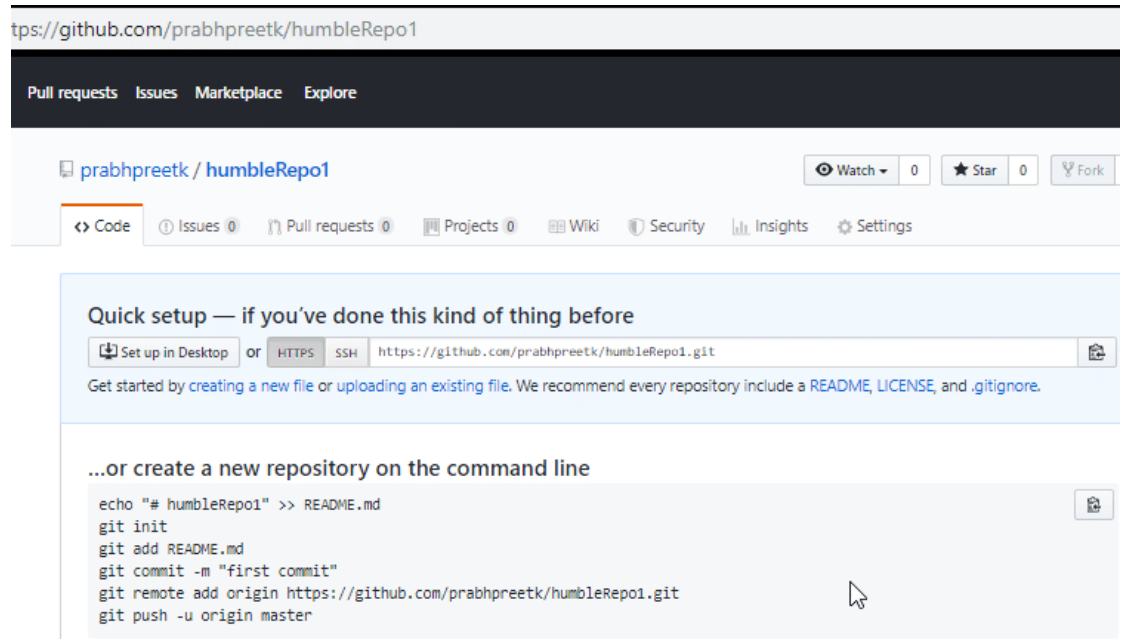
(2) Click on New



Step 2: Now, we have to create a new repository. Provide a name to our repository, select the privacy of the repository as Public, and then click on Create repository



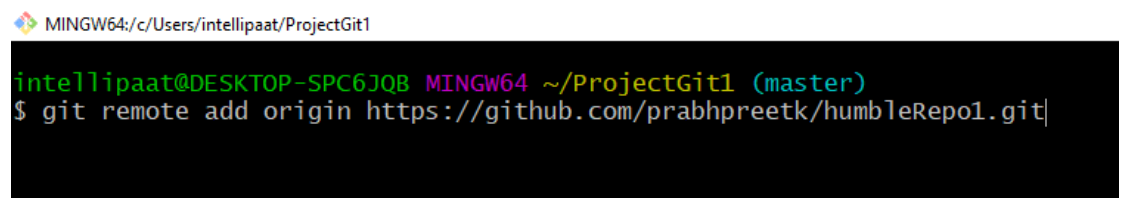
Once we are done with filling up the new repository form, we would land on a page as follows:



Step 3: Click on the Copy icon on the right side of the URL box of the Github repository to copy the link and paste it as shown below:

`git remote add origin "URL"`

Now, we are ready to operate the remote commands in our repository that we have just created.



6. git push

Usage: `git push origin [branch name]`

Suppose, we have made some changes in the file and want to push the changes to our remote repository on a particular branch. By using the command '`git push`,' the local repository's files can be synced with the remote repository on Github.

```

MINGW64:/c/Users/intellipaat/ProjectGit1
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git push origin branch1
fatal: HttpRequestException encountered.
An error occurred while sending the request.
Username for 'https://github.com': prabhpreetk
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 729 bytes | 0 bytes/s, done.
Total 7 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'branch1' on GitHub by visiting:
remote:   https://github.com/prabhpreetk/humbleRepo1/pull/new/branch1
remote:
To https://github.com/prabhpreetk/humbleRepo1.git
 * [new branch]      branch1 -> branch1

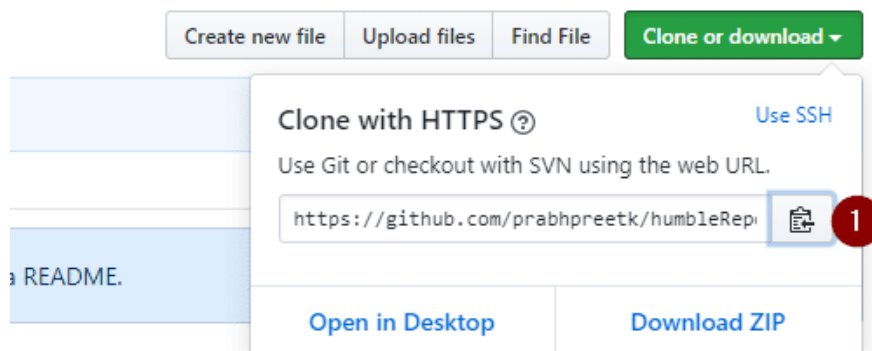
```

7. git clone

Usage: git clone [URL]

Suppose, we want to work on a file that is on a remote Github repository as another developer. How can we do that? We can work on this file by clicking on Clone or Download and copying the link and pasting it on the terminal with the git clone command. This will import the files of a project from the remote repository to our local system.

(1) The below screenshot shows from where to copy the link:



To create a local folder, we have to use the following command:

```
mkdir [directory- name]
```

```
cd [directory- name]
```

```
git clone [URL]
```

Now, paste the copied link along with the git clone command as shown below:

```

MINGW64:/c/Users/intellipaat/test
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/test
$ git clone https://github.com/prabhpreetk/humbleRepo1.git
Cloning into 'humbleRepo1'...

```

Note: Here, we don't have to use the **git remote add origin** command

because we have already cloned the remote repository in the local directory. Now, if we push any new file, it knows where it has to go.

8. git branch

Usage (i): git branch [name-of-the-branch]

So far, we saw how we can work on Git. Now, imagine, multiple developers working on the same project or repository! To handle the workspace of multiple developers, we can use branches. To create a branch (say, the 'name-of-the-branch' is 'branch1'), we use this command:

MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git branch branch1
```

Usage (ii): git branch -D [name -of-the-branch]

Similarly, to delete a branch, we use the git branch -D command:

MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git branch -D branch1
Deleted branch branch1 (was f538c12).
```

9. git checkout

Usage (i): git checkout [name-of-the-new-branch]

We use this command to navigate to an existing branch, add new files, and commit the files:

MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git checkout branch1
Switched to branch 'branch1'
A       1s
```

Usage (ii): git checkout -b [name-of-the-new-branch]

We use this command to create a branch and navigate to that particular branch (say, the 'name-of-the-new-branch' is 'branch2') at the same time:

MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git checkout -b branch2
Switched to a new branch 'branch2'

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch2)
$ git branch
  branch1
* branch2
  master
```

10. git log

Usage (i): git log

This command is used when we want to check the log for every commit in detail

in our repository.

Note: It will show the log of the branch we are in. We can check the last three logs by giving the command: `git log -3`

```
MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git log -3
commit be73fc3e802f8afece5a9f12cea4415665e36bf4 (HEAD -> master, origin/master)
Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
Date: Wed Jun 19 14:56:33 2019 +0530

    Committing 234master.txt in master

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git checkout branch1
Switched to branch 'branch1'

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git log -3
commit deae5df00b52e75abe175f9f5bdcfde84feb6dd8 (HEAD -> branch1, origin/branch1)
Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
Date: Wed Jun 19 15:43:54 2019 +0530

    123master.txt file modified from feature branch

commit bbf434bc2eceaca5d1742664638a9bd05630636d
Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
Date: Wed Jun 19 15:41:09 2019 +0530

    123branch1.txt filein feature branch; 1st commit in feature branch

commit be73fc3e802f8afece5a9f12cea4415665e36bf4 (origin/master, master)
Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
Date: Wed Jun 19 14:56:33 2019 +0530

    Committing 234master.txt in master
```

Usage (ii): `git log --graph`

```
MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git log --graph
* commit deae5df00b52e75abe175f9f5bdcfde84feb6dd8 (HEAD -> branch1, origin/branch1)
  Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
  Date: Wed Jun 19 15:43:54 2019 +0530

    123master.txt file modified from feature branch

* commit bbf434bc2eceaca5d1742664638a9bd05630636d
  Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
  Date: Wed Jun 19 15:41:09 2019 +0530

    123branch1.txt filein feature branch; 1st commit in feature branch

* commit be73fc3e802f8afece5a9f12cea4415665e36bf4 (origin/master, master)
  Author: prabhpreetk <prabhpreet.intellipaat@gmail.com>
  Date: Wed Jun 19 14:56:33 2019 +0530

    Committing 234master.txt in master
```

Usage (iii): `git log --graph --pretty=oneline`

```
MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git log --graph --pretty=oneline
* deae5df00b52e75abe175f9f5bdcfde84feb6dd8 (HEAD -> branch1, origin/branch1) 123mas
* bbf434bc2eceaca5d1742664638a9bd05630636d 123branch1.txt filein feature branch; 1s
* be73fc3e802f8afece5a9f12cea4415665e36bf4 (origin/master, master) Committing 234ma
```

11. git stash

Usage (i): `git stash`

This command can be used when we want to save our work without staging or committing the code to our Git repository and want to switch between branches.

```

MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git checkout branch1
Switched to branch 'branch1'

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ ls
123branch1.txt 123branch2.txt 123master.txt 234master.txt ls

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ echo "123master.txt is getting modified, will be stashed!">>123master.txt

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git stash
warning: LF will be replaced by CRLF in 123master.txt.
The file will have its original line endings in your working directory.
Saved working directory and index state WIP on branch1: deae5df 123master.txt file modified fr

```

Usage (ii): git stash -u

This command is used when we want to stash the untracked files.

```

MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git checkout branch1
Switched to branch 'branch1'

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git stash -u
C:\Users\intellipaat\AppData\Local\Programs\Git\mingw64\libexec\git-core\git-stash: line 42: w
C:\Users\intellipaat\AppData\Local\Programs\Git\mingw64\libexec\git-core\git-stash: line 42: w
warning: LF will be replaced by CRLF in 123branch2.txt.
The file will have its original line endings in your working directory.
Saved working directory and index state WIP on branch1: f538c12 123master.txt file is being con

```

Usage (iii): git stash pop

This command is used when we are back on our branch and want to retrieve the code.

```

MINGW64:/c/Users/intellipaat/ProjectGit1

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git checkout branch1
Switched to branch 'branch1'

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git stash pop
on branch branch1
changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   123master.txt

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

        123branch2.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (90f1754c55f24d72fac9952026ddc46e763dabba)

```

12. git revert

Usage: git revert [commit id]

The git revert command can be considered as an 'undo' command. However, it does not work as the traditional 'undo' operation. It figures out how to invert the changes introduced by the commit and appends a new commit with the resulting inverse content.


```

intellipa@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit2 (master)
* 3fe7411e9309c2a5bf8445211cb094b141e2cb86 (HEAD -> master) Revert "Master humble.txt"
* 788f1b68a399f02ef3069637a942e15e435f51c9 Master humble.txt
* 32e3b572663e41daa2708bd7adac96fe5bba165f Humble.txt
* d9ca5e44e6363e82c803eb032d24021bf8a0e1b6 Humble Gumble3
* dc82b9972a06d87b0d17c174365b710fd05653f3 Humble Gumble2
* 22ec55c68c3bc5e71ecf7c45766d53902715195d Committing Stash
* df25c6aabde6c4486caef8aea6444795cd5d034c Adding file in newFile
* b161ec560c79c9cd2c25df15ff8d65f8bf4cb041 (origin/master, branch1) What is humble gumble?

```

13. git diff

Usage: git diff [commit-id-of-version-x] [commit-id-of-version-y]

Diffing is a function that takes two input datasets and outputs the changes between them. The git diff command is a multi-use Git command which, when executed, runs a diff function on Git data sources. These data sources can be commits, branches, files, and more. The git diff command is often used along with the git status and git log commands to analyze the current state of our Git repository. We use git log to get the details of commit IDs.

Let's compare the working directory with the index as shown below:

```

MINGW64:/c/Users/intellipa/ProjectGit1
intellipa@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git diff f538c123a0f0900d717db8f342f690d9304eee07
diff --git a/123master.txt b/123master.txt
index c656711..f6ffadf 100644
--- a/123master.txt
+++ b/123master.txt
@@ -1,3 +1,2 @@
 123.txt file in master branch; 1st commit in master
 123.txt file modified from feature branch
-123master.txt is getting modified, will be stashed!

```

14. git merge

Usage: git merge [another-file-name]

This command will combine multiple sequences of commits into one unified history. In the most frequent use cases, git merge is used to combine two branches. The git merge command takes two commit pointers, usually the branch tips, and finds a common base commit between them. Once it finds a common base commit, it will create a commit sequence.

```

MINGW64:/c/Users/intellipa/ProjectGit1
intellipa@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git checkout master
Switched to branch 'master'

intellipa@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git merge branch1
Updating be73fc3..f538c12
Fast-forward
 123branch1.txt | 1 +
 123master.txt | 2 ++
 1s             | 0
3 files changed, 3 insertions(+)
create mode 100644 123branch1.txt
create mode 100644 1s

```

15. git rebase

Usage: git rebase [base]

Rebase is the process of moving and combining a sequence of commits to a new base commit. Rebasing is changing the base of our branch from one commit to another, making it appear as if we've created our branch from a different commit. Internally, Git accomplishes this by creating new commits and applying

them to the specified base. It's very important to understand that even though the branch looks the same, it is composed of entirely new commits. The git rebase command performs an automatic git checkout <branch> before doing anything else. Otherwise, it remains on the current branch.

MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git rebase master
current branch master is up to date.
```

Consider a situation where we have branched off from the master and have created a feature branch, but the master branch is still having more commits. We want to get the updated version of the master branch in our feature branch, keeping our branch's history clean, so that it appears as if we are working on the latest version of the master branch.

Note: We don't rebase public history. We should never rebase commits once they are pushed to a public repository. Why because the rebase would replace the old commits with the new ones, and it would appear that a part of our project history got abruptly vanished.

16. git fetch

Usage: git fetch

When we use the command git fetch, Git gathers any commit from the target branch that does not exist in our current branch and stores it in our local repository. However, it does not merge it with our current branch.

MINGW64:/c/Users/intellipaat/ProjectGit1

```
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (branch1)
$ git fetch --all
Fetching origin
```

This is particularly useful when we need to keep our repository up to date but are working on something that might break if we updated our files. To integrate the commits into our master branch, we use merge. It fetches all of the branches from the repository. It also downloads all the required commits and files from another repository.

17. git reset

Usage: git reset --hard [SOME-COMMIT]

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

```

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git reset --hard
HEAD is now at 03aaec4 Committing chnges

intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git log --graph --pretty=oneline
* 03aaec45355ea6fc9a94895a02d4e32f2c7918d9 (HEAD -> master) Committing chnges
* f538c123a0f0900d717db8f342f690d9304eee07 (branch1) 123master.txt file is being committed afte
* deae5df00b52e75abe175f9f5bdcfde84feb6dd8 (origin/branch1) 123master.txt file modified from f
* bbf434bc2eceaca5d1742664638a9bd05630636d 123branch1.txt file in feature branch; 1st commit in
* be73fc3e802f8afece5a9f12cea4415665e36bf4 (origin/master) Committing 234master.txt in master

```

This will discard commits in a private branch or throw away the uncommitted changes!

Here, we have executed a 'hard reset' using the `--hard` option. Git displays the output indicating that the HEAD is pointing to the latest commit. Now, when we check the state of the repo with `git status`, Git will indicate that there are no pending changes (if any prior addition of a new file or modification of an existing file is done before using the '`git reset --hard`' command). Our modifications to an existing file, if not committed, and the addition of a new file, if not staged, will be destroyed. It is critical to take note that this data loss cannot be undone.

If we do `git reset --hard [SOME-COMMIT]`, then Git will:

Make our current branch (typically master) back to point <SOME-COMMIT>

Make the files in our working tree and the index ("staging area") the same as the versions committed at <SOME-COMMIT>

18. git pull

Usage: `git pull origin master`

The `git pull` command first runs '`git fetch`' which downloads the content from the specified remote repository and then immediately updates the local repo to match the content.

```

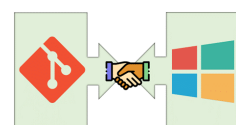
intellipaat@DESKTOP-SPC6JQB MINGW64 ~/ProjectGit1 (master)
$ git pull origin master
From https://github.com/prabhpreetk/humbleRepo1
* branch      master      -> FETCH_HEAD
Already up-to-date.

```

Git and Its Popularity

Git is one of the most important version control systems that has experienced a significant growth rate and popularity over the years. Git plays a key role in both developers' and non-developers' world. If we are part of the software developers' world, it is expected that we know Git. So, having a good grasp of Git is very beneficial for us to get into a lucrative career.

Even Microsoft has started leveraging Git very recently. This opens up more opportunities for the developers' community. Hence, it is the right time to learn Git.



LESSON

9

Introduction to GitHub

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Understand how GitHub can be used in team projects.

GitHub



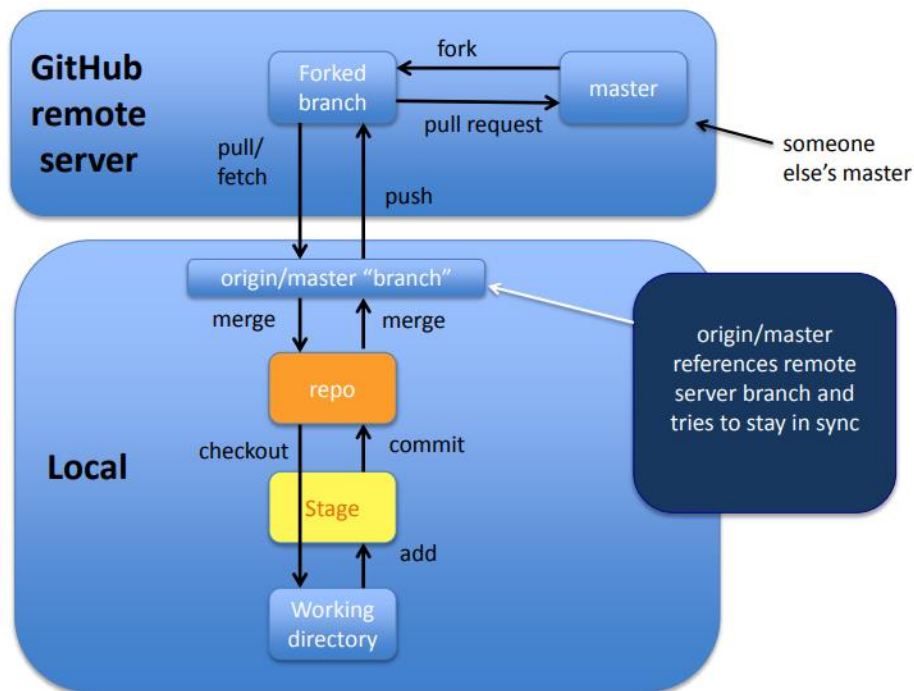
- a platform to host git code repositories
- <http://github.com>
- launched in 2008
- most popular Git host
- allows users to collaborate on projects from anywhere
- GitHub makes git social!
- Free to start

GitHub.com is a site for online storage of Git repositories.

- You can create a remote repo there and push code to it.
- Many open source projects use it, such as the Linux kernel.
- You can get free space for open source projects.

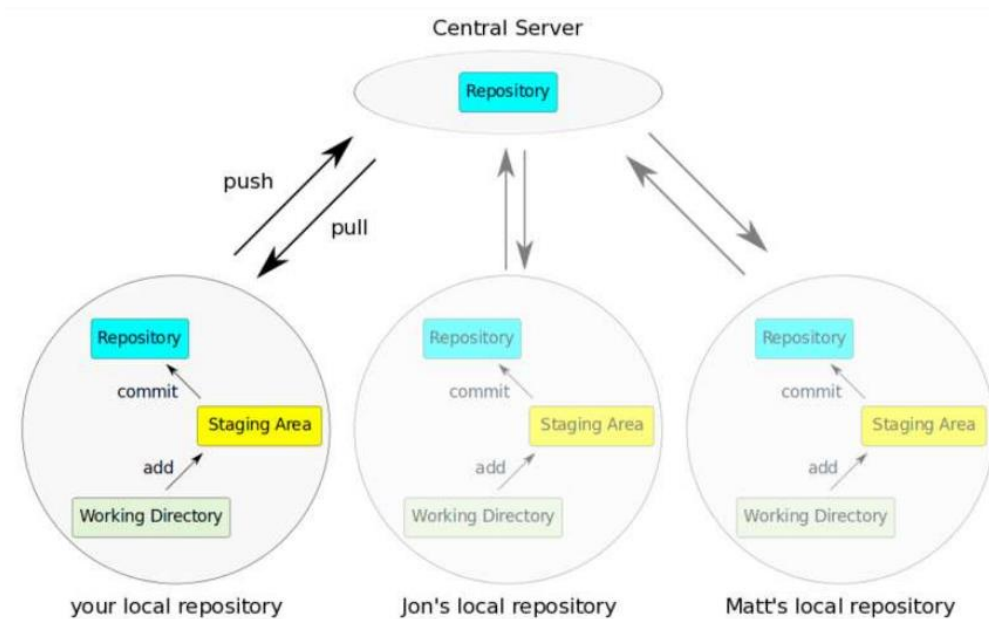
Or you can pay for private projects.

- Free private repos for educational use: github.com/edu
- Question: Do I always have to use GitHub to use Git?
 - Answer: No! You can use Git locally for your own purposes.
 - Or you or someone else could set up a server to share files.
 - Or you could share a repo with users on the same file system, as long everyone has the needed file permissions).

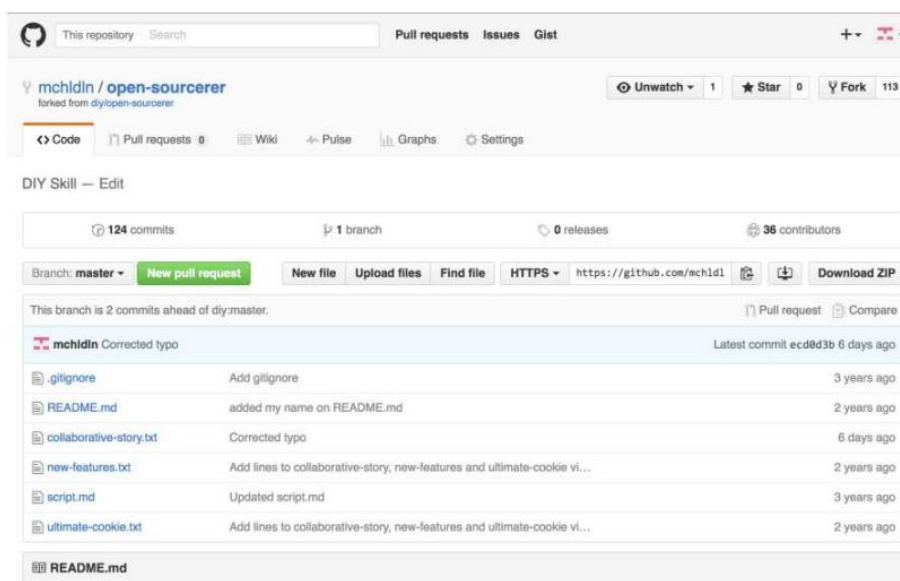


Important to remember

Sometimes developers choose to place repo on GitHub as a centralized place where everyone commits changes, but it doesn't have to be on GitHub.



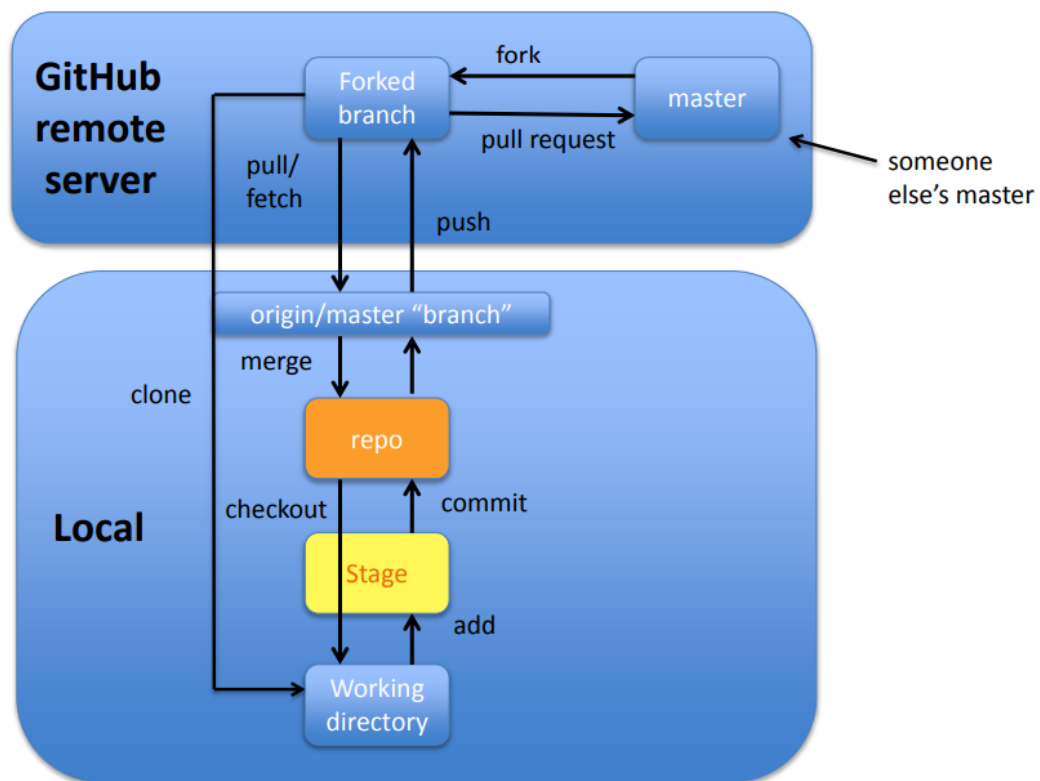
Source: <http://bit.ly/1rvzjp9>



Copying (cloning) files from remote repo to local machine

git clone URL <new_dir_name>

```
[dolanmi]$ git clone https://github.com/mchldn/open-sourcerer.git program_one
Cloning into 'program_one'...
remote: Counting objects: 294, done.
remote: Total 294 (delta 0), reused 0 (delta 0), pack-reused 294
Receiving objects: 100% (294/294), 45.83 KiB | 0 bytes/s, done.
Resolving deltas: 100% (149/149), done.
Checking connectivity... done.
[dolanmi]$ ls
program_one
[dolanmi]$ cd program_one/
[dolanmi]$ ls -aFlt
total 72
drwxrwxr-x  9 dolanmi  NIH\Domain Users   306 May  4 17:26 ./
drwxrwxr-x 13 dolanmi  NIH\Domain Users   442 May  4 17:26 .git/
-rw-rw-r--  1 dolanmi  NIH\Domain Users    19 May  4 17:26 .gitignore
-rw-rw-r--  1 dolanmi  NIH\Domain Users   586 May  4 17:26 README.md
-rw-rw-r--  1 dolanmi  NIH\Domain Users  2938 May  4 17:26 collaborative-story.txt
-rw-rw-r--  1 dolanmi  NIH\Domain Users   138 May  4 17:26 new-features.txt
-rw-rw-r--  1 dolanmi  NIH\Domain Users 12984 May  4 17:26 script.md
-rw-rw-r--  1 dolanmi  NIH\Domain Users   192 May  4 17:26 ultimate-cookie.txt
drwxrwxr-x  3 dolanmi  NIH\Domain Users   102 May  4 17:26 ../
```



How do I link my local repo to a remote repo?

`git remote add <alias> <URL>`

Note: This just establishes a connection...no files are copied/moved

Note: Yes! You may have more than one remote linked to your local directory!

Which remotes am I linked to?

`git remote`

Pushing to a remote repo

`git push local_branch_alias branch_name`

```
[dolanmi]$ git push origin master
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 280 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
To https://github.com/mchldln/open-sourcerer.git
ecd0d3b..212432e master -> master
```

Fetching from a remote repo

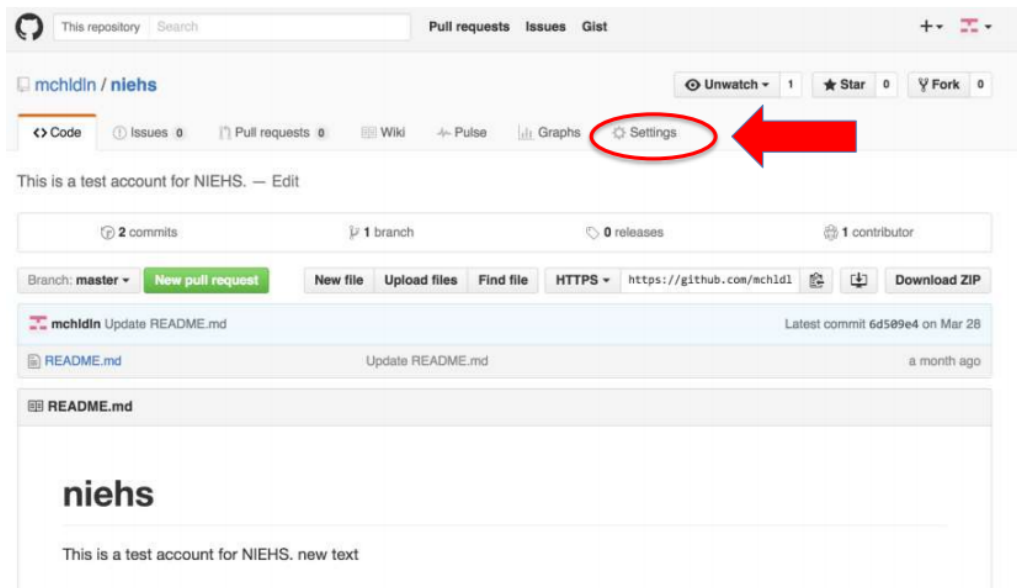
`git fetch remote_repo_name`

Fetch in no way changes at your working dir or any commits that you've made.

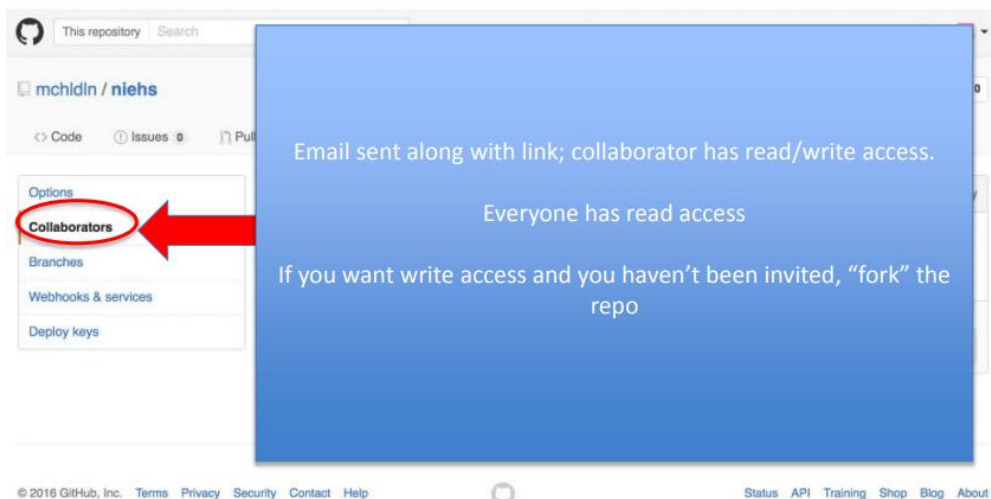
- Fetch before you work
- Fetch before you push
- Fetch often

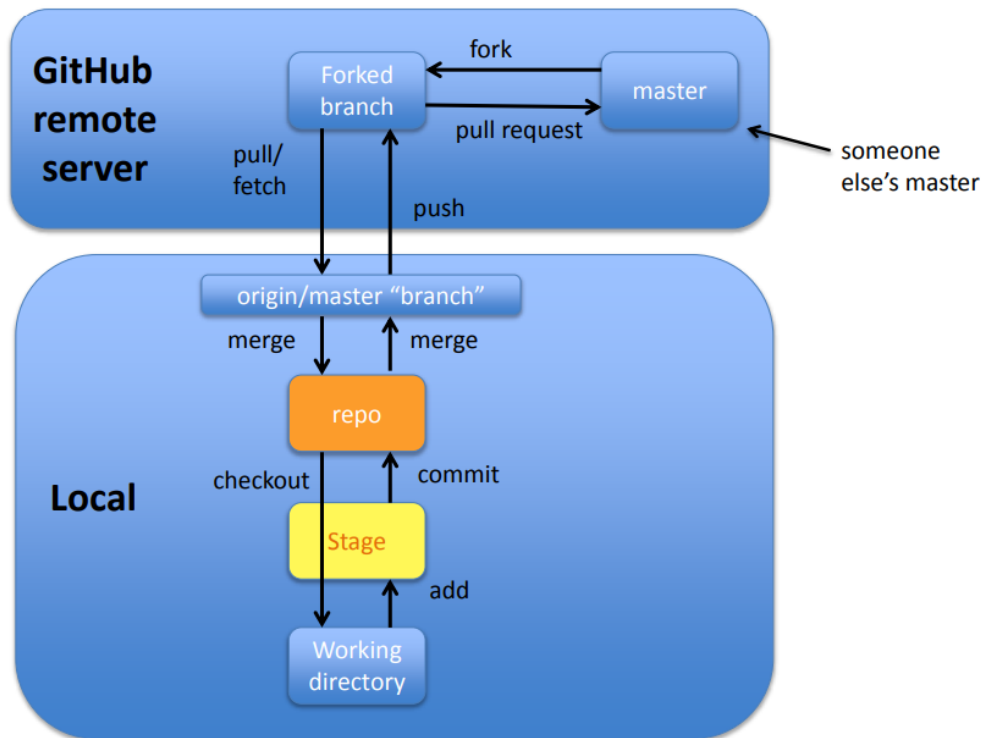
`git merge` must be done to merge fetched changes into local branch

Collaborating with Git



Collaborating with Git





GitHub Gist

<https://gist.github.com/>

The screenshot shows the GitHub Gist creation interface. At the top, there is a search bar, a link to "All gists", and buttons for "Sign up for a GitHub account" and "Sign in". Below this is a heading "Instantly share code, notes, and snippets." followed by a text input field for "Gist description...". Below the description field is a text input field for "Filename including extension...". To the right of the filename field are three dropdown menus: "Spaces" (set to 2), "2" (set to 2), and "No wrap" (set to No wrap). Below these fields is a large text area for the code content, with a line number "1" visible on the left. At the bottom left is an "Add file" button, and at the bottom right are two buttons: "Create secret gist" and "Create public gist".

REFERENCES

Books

Learning DevOps: Continuously Deliver Better Software (2016), Joakim Verona et.al.
DevOps for Developers (2012), Michael Hüttermann

Internet

<https://opensource.com>
<https://intellipaat.com>
<https://gist.github.com/>
<https://www.slideshare.net/>
<https://www.guru99.com>
<https://www.atlassian.com/agile/devops>
<https://tkt-lapio.github.io/>
<https://www.atlassian.com/>