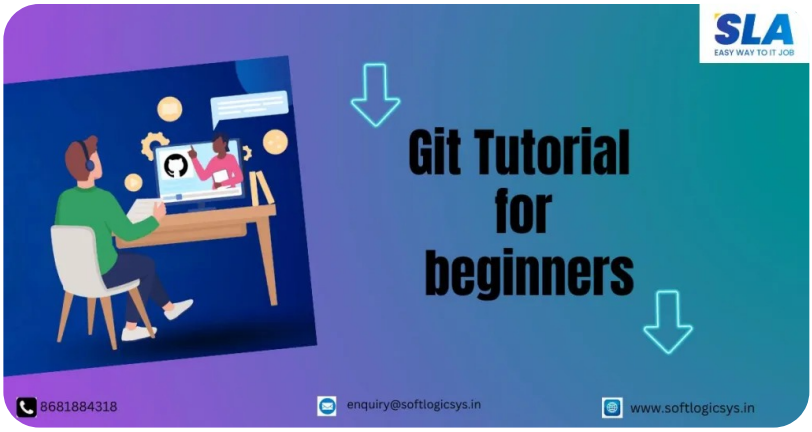

Share on your Social Media

Git Tutorial for beginners

Published On: September 16, 2024

Git Tutorial for beginners

Git is a robust version control system that is frequently used to monitor source code changes made during the software development process. You can easily understand and implement Git concepts in your projects after reading this Git tutorial.

Introduction to Git

Git is becoming an essential tool for **DevOps engineers** all around the world. Gaining an understanding of Git will considerably improve your teamwork and coding efficiency. You can learn the following in this Git tutorial:

- Overview of Git

Featured Articles

Related Courses at SLA

- Git Online Training
- Git Training in OMR
- Git Training in Chennai

Related Posts

- Git Installation and Configuration
- Git New Files
- Git Staging
- Git Commit
- Git Help
- Git Branch
- Merge branches

Download GIT Tutorial PDF

Overview of Git

Developers can work together, keep track of changes in their codebase, and effectively manage many project versions with Git, a distributed version control system (DVCS). Linus Torvalds created Git in 2005 to develop Linux kernels. The latest version of Git is 2.45.1.

- Monitoring modifications to the code
- Monitoring the individuals who made modifications
- Cooperation in coding

With GitHub, you may access and download projects from any computer that hosts Git repositories. GitHub allows you to perform the following:

- **Store Repositories:** Your repositories are hosted by GitHub.
- **Collaborate:** Work together with developers from any location by collaborating.
- **Version Control:** Use GitHub and Git to oversee cooperative processes.

Features of Git

Important features of Git:

- Version control to track changes in the code.

Tableau Developer Salary in Chennai
Published On: October 12, 2024

Introduction A Tableau Developer designs, develops, and maintains dashboards and visualizations using Tableau software. Key...

Ideas For GitHub Projects
Published On: October 12, 2024

Introduction A GitHub Professional manages software development projects on the GitHub platform. Responsibilities include version...

VMware Tutorial for Cloud Computing Aspirants
Published On: October 12, 2024

VMware Tutorial for Cloud Computing Aspirants VMware software allows you to run a virtual machine...

- Collaboration to be used by multiple developers.
- Backup facility for the entire project
- Branching and merging for fixing bugs easily.
- Open source for free contribution.
- Industry-standard skills in the software industry.

How does Git work?

- **Initializing a Repository:** A folder initialized with Git is transformed into a repository. Git keeps track of every modification made to a hidden folder in that repository.
- **Staging Changes:** Git labels changed files as "staged" to indicate staging changes. Adjustments are staged for a desired take that you wish to preserve.
- **Committing Changes:** After phased modifications are deemed acceptable, commit them. Git keeps a thorough log of every commit.

Git Installation and Configuration

The following website offers a free download of Git: **https://www.git-scm.com/**

Git with Command Line

We're going to launch our command shell before we use Git. Git bash, which is a part of Git for Windows, can be used on Windows. The integrated terminal can be used with Mac and Linux.

The first thing we must do is make sure Git is installed correctly. Check with the following code:

git –version

git version 2.30.2.windows.1

If Git is installed, the screen should read "git version X.Y."

Git Configuration

VBA Macros Tutorial for Beginners
Published On: October 10, 2024
VBA Macros Tutorial for Beginners VBA macros are programs that automate repetitive operations in Microsoft...

Tell Git who you are now. Version control systems should take note of this since every Git commit makes use of this data:

git config –global user.name "sla-test"

git config –global user.email "test@sla.com"

Replace the email address and user name with your own. This is certainly something you'll want to utilize later on when you sign up for GitHub.

Creating Git Folder

Let's now establish a new project folder:

mkdir myproject

cd myproject

mkdir creates a brand-new folder.

The current working directory is modified using *cd.*

Initialize Git

You can initialize Git on that folder after you've gone to the correct one:

git init

Initialized empty Git repository in /Users/user/myproject/.git/

GIT Interview Questions and Answers

Git New Files

Now let's add additional files, or use your preferred text editor to create a new file. After that, save it or transfer it to the newly formed folder. Learn the fundamental **HTML concepts** here.

Example

> *<!DOCTYPE html>*

> *<html>*
>
> *<head>*
>
> *<title>Hello World!</title>*
>
> *</head>*
>
> *<body>*
>
> *<h1>Hello world!</h1>*
>
> * This is the first file in my new Git Repo. *
>
> *</body>*
>
> *</html>*

Save it as index.html in our newly created folder.

Returning to the console, let's enumerate the files in our active working directory:

ls

index.html

The files in the directory will be listed by ls.
Index.html exists.

Next, we determine whether it is a part of our repository by looking at the Git status:

git status

On branch master

No commits yet

Untracked files:

(use "git add ..." to include what will be

committed.)

index.html

Nothing was added to the commit but untracked files are present (use "git add" to track).

There are two possible updates for files in your Git repository folder:

- **Tracked:** Files that Git is aware of and adds to the repository are called tracked files.
- **Untracked:** Files in your working directory that haven't been uploaded to the repository are called untracked.

Upon initially adding files to an empty repository, they are not monitored. You must stage them, or add them to the staging environment for Git to track them.

Check out our **Nagios course** to get acquainted with event monitoring in DevOps.

Git Staging

The Staging Environment and Commit concepts are two of Git's main features.

You might be adding, modifying, and deleting files while you're working. However, you ought to add the files to a staging environment whenever you complete a task or reach a milestone.

Staged files are those that are ready to be committed to the repository that you are currently working on.

Example:

git add index.html

The document has to be staged. The result is as follows:

git status

On branch master

No commits yet

Changes to be committed:

(use "git rm –cached ..." to unstage)

new file: index.html

The file is currently present in the staging environment.

GIT Syllabus PDF

Git Add Multiple Files

It is also possible to stage multiple files simultaneously. Let's expand our working folder with two extra files. Reopen the text editor.

A repository's README.md file, which is advised for all repositories:

 # hi there, world

Hello World Git sample repository

This repository serves as an example for the Git tutorial.

If you are new to **CSS concepts**, explore here.

Example: bluestyle.css

> *body {*
>
> *background-color: lightblue;*
>
> *}*
>
> *h1 {*
>
> *color: navy;*

```
margin-left: 20px;

}
```

Index.html

```
<!DOCTYPE html>

<html>

<head>

<title>Hello World!</title>

<link rel="stylesheet"
href="bluestyle.css">

</head>

<body>

<h1>Hello world!</h1>

<p>This is the first file in my new Git
Repo.</p>

</body>

</html>
```

Now, add every file to the staging environment located in the current directory:

git add –all

To stage all changes (new, updated, and deleted files), use –all rather than individual filenames.

git status

On branch master

No commits yet

Changes to be committed:

(use "git rm –cached ..." to unstage)

new file: README.md

new file: bluestyle.css

new file: index.html

We have added all three files to the staging environment and are now ready to make our first commit. Learn **Jenkins to automate parts** of software development.

Git Commit

Adding commits allows us to monitor our work's advancement and modifications. Every commit is regarded by Git as a "save point" or change point. You can go back to this point in the project if you wish to make changes or discover an issue.

There should always be a message included when we commit. Every commit should have a clear message added to it so that you and others can easily see what has changed and when.

Example

git commit -m "First release of Hello World!"

[master (root-commit) 221ec6e] First release of Hello World!

3 files changed, 26 insertions(+)

create mode 100644 README.md

create mode 100644 bluestyle.css

create mode 100644 index.html

The -m "message" option adds a message, and the

commit command executes a commit.

The message that the staging environment has committed to our repository is “First release of Hello World!”

Commit without Stage

Using the staging environment seems like a waste of effort when you make little modifications. Changes can be directly committed without going via the staging environment. Every modified, previously tracked file will be automatically staged by using the -a option.

Update to index.html:

> *<!DOCTYPE html>*
>
> *<html>*
>
> *<head>*
>
> *<title>Hello World!</title>*
>
> *<link rel=”stylesheet” href=”bluestyle.css”>*
>
> *</head>*
>
> *<body>*
>
> *<h1>Hello world!</h1>*
>
> * This is the first file in my new Git Repo. *
>
> * A new line in our file! *
>
> *</body>*

> *</html>*

This time, we'll use the –short option to view the changes more concisely:

git status –short

M index.html

The following are brief status flags:

- ?? – Untracked files
- A – Files added to the stage
- M – Modified files
- D – Deleted files

We notice the anticipated file has been altered. So let's just say it out loud:

git commit -a -m "Updated index.html with a new line"

[master 09f4acd] Updated index.html with a new line

1 file changed, 1 insertion(+)

Git Commit Log

The log command can be used to see a repository's commit history:

git log

commit 09f4acd3f8836b7f6fc44ad9e012f82faf861803 (HEAD -> master)

Author: sla_test

Date: Thu Aug 10 11:20:15 2024 +0100

Updated index.html with a new line

commit 221ec6e10aeedbfd02b85264087cd9adc18e4b26

Author: sla-test

Date: Thu Aug 10 11:20:15 2024 +0100

First release of Hello World!

Git Help

The help command in the command line can be used in a few different ways:

- **git command -help**: View every option for that particular command.
- **git help –all:** View the whole list of commands.

Git -help View Your Options for a Particular Command

git commit -help

usage: git commit [] [–] …

-q, –quiet after a successful commit, hide the summary

-v, –verbose display the difference in the commit message template

Commit message options

-F, –file read the file message

–author replace the author to commit

–date override the commit date

-m, –message

commit message

-c, –reedit-message

use and modify the message from the given commit

-C, –reuse-message

reuse the message from the given commit

–fixup use a message formatted using autosquash to repair a specific commit

–squash utilize a message prepared with auto squash to squash a given commit

–reset-author Now that I used -C/-c/–amend, I am the committer.

-s, –signoff incorporate a signed-off-by trailer

-t, –template utilize the given template file

-e, –edit compel the commit to be edited

–cleanup how to remove #comments and spaces from a message

–status incorporate the status into the template for the commit message

-S, –gpg-sign[=] GPG sign commit

Commit contents options

-a, –all commit all updated files

-i, –include add the designated files to the commit index

–interactive dynamically include files

-p, –patch dynamically add modifications

-o, –only commit just the designated files

-n, –no-verify avoid using the commit-msg and pre-commit hooks

–dry-run demonstrate the potential offense

–short concisely display the status

–branch display branch details

–ahead-behind total ahead/behind values

computation

–porcelain machine-readable output

–long display status in extended format by default

-z, –null finish entries with a null value

–amend change a prior commit

–no-post-rewrite omit the post-rewrite hook

-u, –untracked-files[=]

reveal untracked files; select between all, normal, and no options. (By default, all)

–pathspec-from-file

read pathspec from file

–pathspec-file-nul with –pathspec-from-file, NUL characters are used to separate pathspec elements.

Do you have an idea to enhance your skill in configuration management and orchestration in the DevOps process? Join our **Ansible training.**

GITHUB Salary

Git help –all View Every Command That Is Possible

Use the help –all command to see a list of all available commands. Some of them are given here:

Main porcelain commands

- **add:** Add contents of the file to the index.
- **am:** Installing multiple patches from a mailbox.
- **archive:** Make a file archive from a specified tree.

Ancillary Commands / Manipulators

- **config:** Obtain and adjust global or repository options.
- **fast-export:** Git data exporter.
- **fast-import:** Fast Git data importers' backend

Ancillary Commands / Interrogators

- **annotate:** Add commit information to file lines by annotation.
- **blame:** Display the author and revision number of each line in a file.
- **bugreport:** Gather data so that the user can report a bug.
- **count-objects:** Counts the number of unpacked objects and how much disk space they take up.

Interacting with Others

- **archimport:** Using Git, import a GNU Arch repository
- **cvsexportcommit:** One commit can be exported to a CVS checkout.
- **cvsimport:** Extract your data from a different SCM that people detest.

Low-level Commands / Manipulators

- **apply:** Put a patch on the index or the files.
- **checkout-index:** Transfer files to the working tree from the index.
- **commit-graph:** Create and validate commit-graph files for Git.
- **commit-tree:** Make a fresh commit object.

Low-level Commands / Interrogators

- **cat-file:** Give repository object content, type, and size information.
- **cherry:** Locate commits that haven't been updated for upstream diff files.
- **diff-files:** Compares the working tree's files with the diff-index of the index
- **diff-index:** A tree's comparison with the

working tree or index

Low-level Commands / Syncing Repositories

- **daemon:** Abasic Git repository server
- **fetch-pack:** Get the absent items from a different repository.
- **http-backend:** Git implemented on the server side via HTTP

Low-level Commands / Internal Helpers

- **check-attr:** Show details about gitattributes.
- **check-ignore:** Track down gitignore/exclude files.
- **check-mailmap:** Display contacts' canonical names and email addresses.
- **check-ref-format:** Ensures a reference name is correctly formulated.

External commands

- askyesno
- credential-helper-selector
- flow
- lfs

Git Branch

A branch in Git is a fresh, independent version of the main repository.

- You can work on other project components without affecting the main branch by using branches.
- A branch can be combined with the main project after the work is finished.
- Even switching between branches and working on distinct projects is possible without causing conflicts.
- Git branching is incredibly quick and light.

New Git Branch

Update to index.html:

While working in our private repository, we aim to

avoid interfering with or potentially damaging the main project. Thus, we make a fresh branch:

git branch hello-world-images

We have now made a “hello world-images” branch.

Let’s make sure that the new branch has been created:

git branch

hello-world-images

** master*

The * next to master indicates that we are now on the “hello-world-images” branch, even though we can see the new branch with that name.

The command to check out a branch is checkout. Changing our focus from the active branch to the one indicated by the command’s completion:

Example

git checkout hello-world-images

Switched to branch ‘hello-world-images’

We have now relocated our workplace to the new branch from the master branch.

Launch your preferred editor and make some adjustments.

Now we updated the index.html file with the following code and added an image (img_hello_world.jpg) to the working folder:

> *<!DOCTYPE html>*
>
> *<html>*
>
> *<head>*

> *<title>Hello World!</title>*
>
> *<link rel=”stylesheet”*
> *href=”bluestyle.css”>*
>
> *</head>*
>
> *<body>*
>
> *<h1>Hello world!</h1>*
>
> * <img src=”img_hello_world.jpg”*
> *alt=”Hello World from Space”*
>
> *style=”width:100%;max-width:960px”>*
> * *
>
> * This is the first file in my new Git*
> *Repo. *
>
> * A new line in our file! *
>
> *</body>*
>
> *</html>*

Within the working directory, we have inserted a new file and modified an existing one (same directory as the main branch). Check the current branch’s status now:

git status

On branch hello-world-images

Changes not staged for commit:

(use “git add …” to update what will be committed.)

(use “git restore …” to discard changes in the working directory)

modified: index.html

Untracked files:

(use “git add …” to include in what will be committed.)

img_hello_world.jpg

No changes added to commit (use “git add” and/or “git commit -a”)

Let’s review what transpires in this example:

- Our index.html has changed; however, the file is not yet ready for commit.
- There is no tracking for img_hello_world.jpg.

Therefore, we must add the following two files to this branch’s staging environment:

git add –all

When you use –all rather than specific filenames, all altered files—new, modified, and deleted—will be staged.

Verify the branch’s status:

git status

On branch hello-world-images

Changes to be committed:

(use “git restore –staged …” to unstage)

new file: img_hello_world.jpg

modified: index.html

With our adjustments, we are content. Thus, we’re going to add these to the branch:

git commit -m “Added image to Hello World”

[hello-world-images 0312c55] Added image to Hello World

2 files changed, 1 insertion(+)

create mode 100644 img_hello_world.jpg

There is a new branch that is distinct from the master branch at this time.

Switching Between Branches

It's time to go through the files in the current directory since we uploaded an image to this branch:

ls

README.md bluestyle.css img_hello_world.jpg index.html

The newly created file, img_hello_world.jpg, is visible, and upon opening the HTML file, we can observe that the coding has been modified. All is in its proper place.

git checkout master

Switched to branch 'master'

This branch does not contain the updated picture. The files in the current directory are once again listed:

ls

README.md bluestyle.css index.html

The file img_hello_world.jpg is missing! Additionally, we can observe that the code has been restored to its original state by opening the html file.

Emergency Branch

Since hello-world images are still being worked on, we need not to tamper with them directly or with the master.

To handle the situation, we thus establish a new branch:

git checkout -b emergency-fix

Switched to a new branch 'emergency-fix'

We have since switched to a new branch that we made from master. Without interfering with the other branches, we may securely correct the issue.

> *<!DOCTYPE html>*
>
> *<html>*
>
> *<head>*
>
> *<title>Hello World!</title>*
>
> *<link rel="stylesheet" href="bluestyle.css">*
>
> *</head>*
>
> *<body>*
>
> *<h1>Hello world!</h1>*
>
> * This is the first file in my new Git Repo. *
>
> * This line is here to show how merging works. *
>
> *</body>*
>
> *</html>*

Now the status:

git status

On branch emergency-fix

Changes not staged for commit:

(use "git add ..." to update what will be committed)

(use "git restore ..." to discard changes in working directory)

modified: index.html

no changes added to commit (use "git add" and/or "git commit -a")

File staging and commit

git add index.html

git commit -m "updated index.html with emergency fix"

[emergency-fix dfa79db] updated index.html with emergency fix

1 file changed, 1 insertion(+), 1 deletion(-)

Do you want to become a DevOps engineer with AWS skills? Join our **AWS DevOps** course.

GIT Training

Git Branch Merges

Let's combine the master and emergency-fix branches now that the emergency fix is available.

We must switch to the master branch first.

git checkout master

Switched to branch 'master'

We are now merging the emergency fix with the main branch:

git merge emergency-fix

Updating 09f4acd..dfa79db

Fast-forward

index.html | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

Git interprets this as a continuation of master, as the emergency-fix branch was taken straight from master and no additional modifications were added to master throughout our work. As a result, it can "Fast-forward" by simply pointing to the same commit for emergency repair and master.

We may remove emergency-fix since it is no longer required because master and emergency-fix are now nearly identical:

git branch -d emergency-fix

Deleted branch emergency-fix (was dfa79db).

Merge Conflict

We may now switch to hello-world images and continue our work there. To make it appear, add another image file (img_hello_git.jpg) and modify index.html.

git checkout hello-world-images

Switched to branch 'hello-world-images'

Example

> *<!DOCTYPE html>*
>
> *<html>*
>
> *<head>*
>
> *<title>Hello World!</title>*

```
<link rel="stylesheet"
href="bluestyle.css">

</head>

<body>

<h1>Hello world!</h1>

<div><img src="img_hello_world.jpg"
alt="Hello World from Space"
style="width:100%;max-width:960px">
</div>

<p>This is the first file in my new Git
Repo.</p>

<p>A new line in our file!</p>

<div><img src="img_hello_git.jpg"
alt="Hello Git" style="width:100%;max-
width:640px"></div>

</body>

</html>
```

We can stage and commit for this branch as we have completed our work on it:

```
git add –all

git commit -m "added new image"

[hello-world-images 1f1584e] added new image

 2 files changed, 1 insertion(+)

 create mode 100644 img_hello_git.jpg
```

Both branches' index.html files have been modified. The hello-world images can now be merged into the master.

git checkout master

git merge hello-world-images

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

The automatic merge failed; resolve disputes before committing the outcome.

Due to a conflict between the versions of index.html, the merging attempt was unsuccessful.

git status

On branch master

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge –abort" to abort the merge)

Changes to be committed:

new file: *img_hello_git.jpg*

new file: *img_hello_world.jpg*

Unmerged paths:

(use "git add …" to mark resolution)

both modified: *index.html*

This demonstrates that although there is a conflict in index.html, the image files are prepared and in a staged state for commit.

Thus, we must resolve that conflict. Use our editor to open the file:

```
<!DOCTYPE html>

<html>

<head>

<title>Hello World!</title>

<link rel="stylesheet"
href="bluestyle.css">

</head>

<body>

<h1>Hello world!</h1>

<div><img src="img_hello_world.jpg"
alt="Hello World from Space"
style="width:100%;max-width:960px">
</div>

<p>This is the first file in my new Git
Repo.</p>

<<<<<<< HEAD

<p>This line is here to show how
merging works.</p>

=======

<p>A new line in our file!</p>

<div><img src="img_hello_git.jpg"
alt="Hello Git" style="width:100%;max-
width:640px"></div>
```

```
>>>>>>> hello-world-images

</body>

</html>
```

We can view the variations between the versions and make any necessary edits:

```
<!DOCTYPE html>

<html>

<head>

<title>Hello World!</title>

<link rel="stylesheet"
href="bluestyle.css">

</head>

<body>

<h1>Hello world!</h1>

<div><img src="img_hello_world.jpg"
alt="Hello World from Space"
style="width:100%;max-width:960px">
</div>

<p>This is the first file in my new Git
Repo.</p>

<p>This line is here to show how
merging works.</p>
```

> * <img src="img_hello_git.jpg" alt="Hello Git" style="width:100%;max-width:640px"> *
>
> *</body>*
>
> *</html>*

We can now stage index.html and evaluate the situation:

git add index.html

git status

On branch master

All conflicts fixed but you are still merging.

(use "git commit" to conclude merge)

Changes to be committed:

new file: *img_hello_git.jpg*

new file: *img_hello_world.jpg*

modified: *index.html*

With the problem fixed, we can use commit to finish the merging:

Example:

git commit -m "merged with hello-world-images after fixing conflicts"

[master e0b6038] merged with hello-world-images after fixing conflicts

And remove the branch called hello-world-images:

git branch -d hello-world-images

Deleted branch hello-world-images (was 1f1584e).

You now know more about how branching and merging operate. Are you fresher to kick-start your career? Explore a wide range of **software course options** and begin your learning journey to reach your dream destination.

Conclusion

We covered the essential concepts of Git with examples in this Git tutorial and we hope this would be useful for you to get started on your DevOps journey. Explore a career in DevOps with our **Git training in Chennai.**

Share on your Social Media

Softlogic Academy

Softlogic Systems

KK Nagar [Corporate Office]

No.10, PT Rajan Salai, K.K. Nagar, Chennai – 600 078.
Landmark: Karnataka Bank Building
Phone: +91 86818 84318
Email: enquiry@softlogicsys.in
Map: Google Maps Link

OMR

No. E1-A10, RTS Food Street
92, Rajiv Gandhi Salai (OMR),
Navalur, Chennai - 600 130.
Landmark: Adj. to AGS Cinemas

Phone: +91 89256 88858
Email: info@softlogicsys.in
Map: Google Maps Link

Courses

- Python
- Software Testing
- Full Stack Developer
- Java
- Power BI
- Clinical SAS
- Data Science
- Embedded
- Cloud Computing
- Hardware and Networking
- VBA Macros
- Mobile App Development
- DevOps

Social Media Links

Review Sources

- Google
- Trustpilot
- Glassdoor
- Mouthshut
- Sulekha
- Justdial
- Ambitionbox
- Indeed
- Software Suggest
- Sitejabber