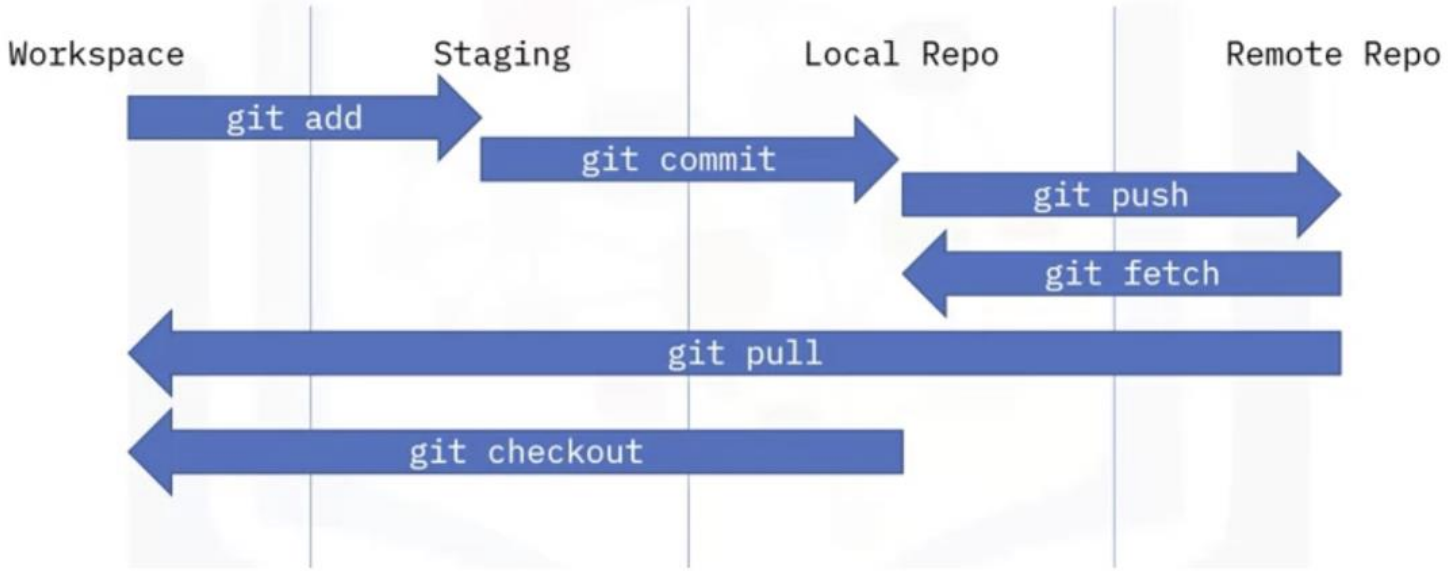
Lab by Titus Majeza

Git Quick Reference

(https://www.linkedin.com/in/titus-majeza/)

Git workflow



For more labs visit my GitHub repo: https://github.com/TitusM/Cisco-Data-Center

Note
This lab was conducted in a controlled environment. Any configurations in a production network should be implemented during a designated maintenance window. Additionally, always refer to official documentation relevant to your specific hardware and software.

"The greatest skill you can develop is decreasing the time between idea and execution"

Version Control System

Version control systems have two important purposes:

- Manage file changes
- Enable collaboration

Version control systems (VCSs) can help manage changes to files over time, including source code management.

Git

Git is a distributed VCS that keeps track of every modification to the code.

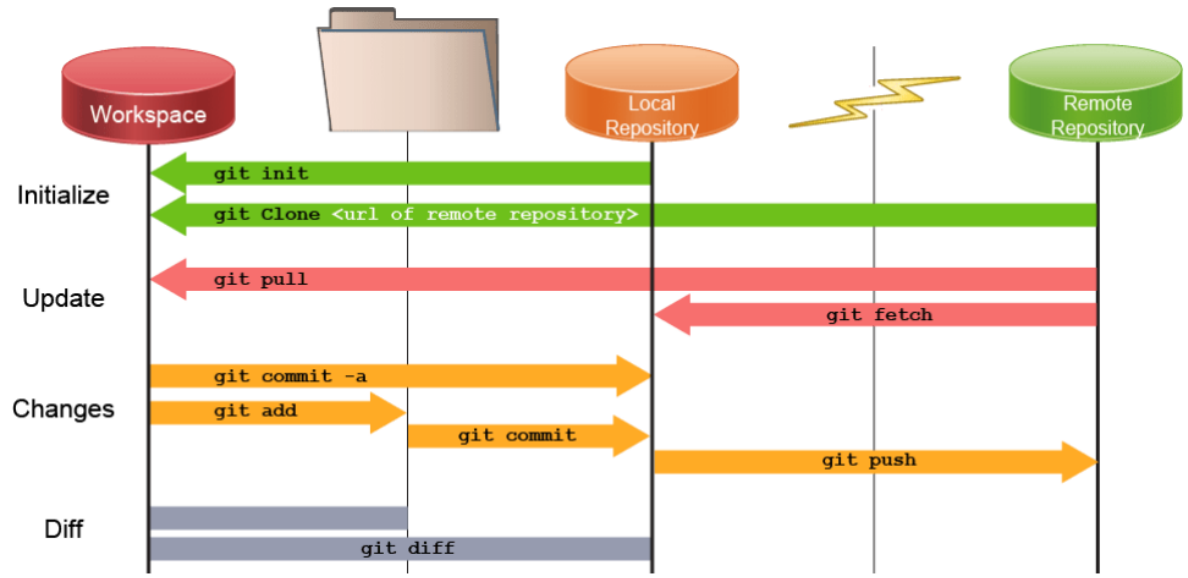
Git proves to be valuable for team-based projects that require error-tracking, code backup and recovery, code history, change logging, and an easier way to experiment with code.

Example for Network Engineers:
It is useful to have a history of all the configurations that are active on network devices. Using a Git-based system allows you to see the configurations, how and when they changed, and of course, who made the change.

Also, as more automation tooling is used, for example, Ansible, you can use Git to version control playbooks, variables, and configuration files.

As you collect information from switches, such as operational data, you can store that data in text files. You can easily track and see the changes in operational data such as changing neighbor adjacencies, routes etc.

Git Architecture & Workflow



"The greatest skill you can develop is decreasing the time between idea and execution"

The Git architecture is composed of several components:

- remote repository
- local repository
- staging area
- working directory.

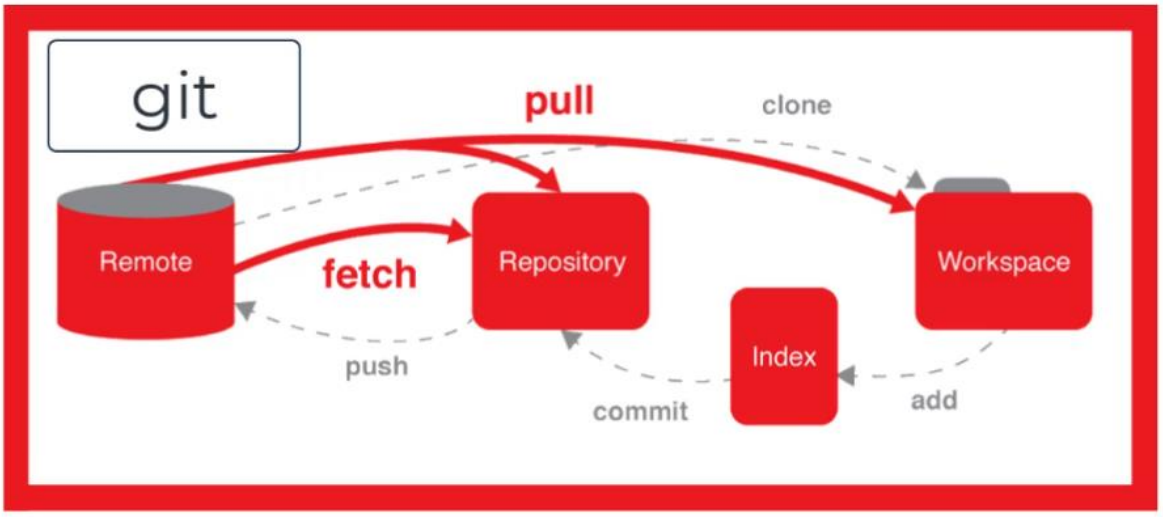
Remote Repository
A remote repository is where the files of the project reside, and it is also where all other local copies are pulled from. It can be stored on an internal private server or hosted on a public repository such as GitHub or GitLab.

Local Repository
A local repository is where snapshots, or commits, are stored on the local machine of each person.

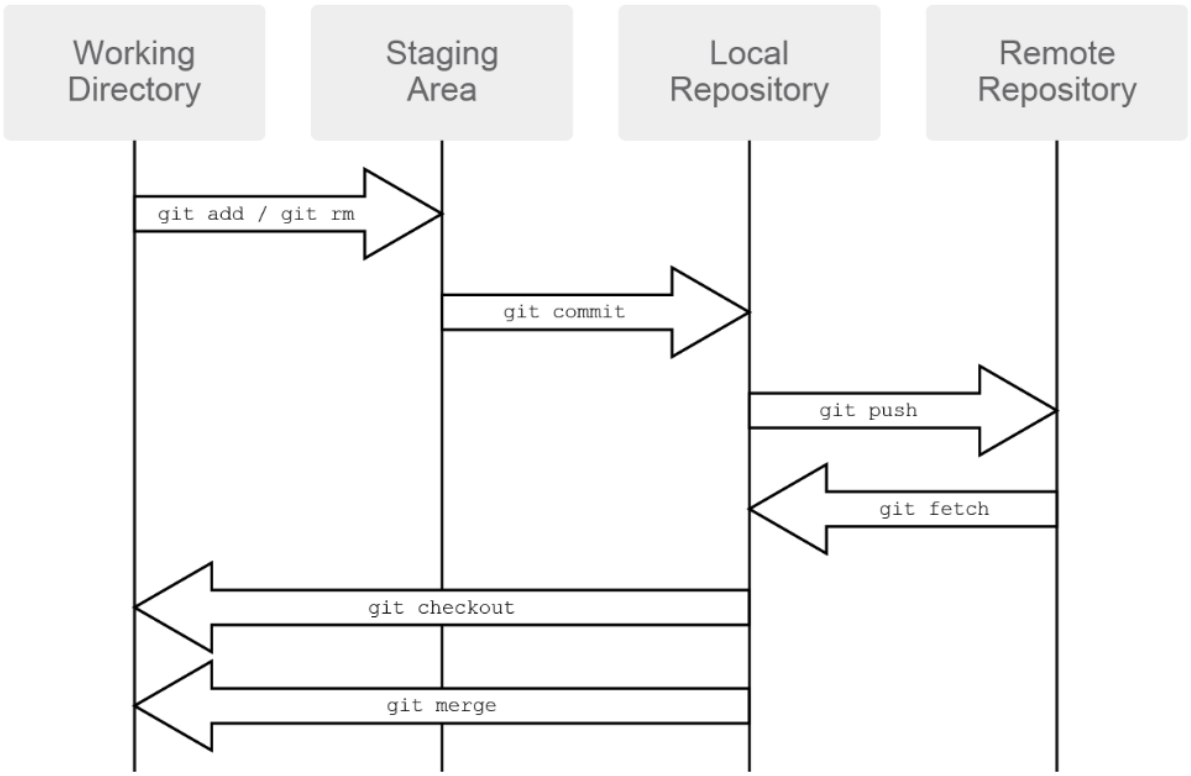
Staging Area
The staging area is where all the changes that you want to perform are placed. You, as a project member, decide which files Git should track.

For example, you can decide to add and commit files to fully become part of your local repository by moving them to the staging area first, without including all files in your project.

Working Directory
A working directory is a directory that Git controls. Git will track differences between your working directory and local repository, and between your local repository and the remote repository.



"The greatest skill you can develop is decreasing the time between idea and execution"



Common Git Commands

Git Command	Description
git init	Initializes a directory for a Git project
git config	Configures Git parameters such as username and password
git add <file/dir>	Starts tracking files and adds them to the staging area
git status	Checks status of your project
git commit –m <message>	Creates a local snapshot
git push <params>	Pushes local snapshot (commits) to a remote repository
git pull <params>	Patches and merges changes from the remote repository (combines **git fetch** and **git merge**)
git clone <params>	Copies another (remote) project to your local machine
git diff <params>	To view the differences in files in your working directory when they are compared to the same files in your local repository (last snapshot commit).

Simple Git Workflow Example

This is the entire workflow that will be displayed in this section:

"The greatest skill you can develop is decreasing the time between idea and execution"

```
NetDevOps % git init
NetDevOps % ls -la | grep .git
NetDevOps % git config --local user.name "TitusM"
NetDevOps % git config --local user.email tmajeza@gmail.com
NetDevOps % git config –list
NetDevOps % git status
NetDevOps % git add README.md
NetDevOps % git commit -m "Adding the README.md file"
NetDevOps % git remote add origin https://github.com/TitusM/NetDevOps
NetDevOps % git remote
NetDevOps % git remote – v
NetDevOps % git push origin main
NetDevOps % git push --set-upstream origin main
NetDevOps % git push
NetDevOps % git pull
```

Workflow breakdown:
On your local machine, create a new project folder.
Use **git init** to initialize a project work with Git and for Git to start tracking files.

```
NetDevOps % git init

Initialized empty Git repository in /NetDevOps/.git/
```

Note

When you execute the git init command, a subdirectory that is called **.git** is also created that contains all the local snapshots (commits) and other metadata about the project.

```
NetDevOps % ls -la | grep .git
drwxr-xr-x@  10 tmajeza  staff      320 Nov 23 12:45 .git
```

Once you initialize a directory using the *git init* command, it is recommended to configure Git with a username and email address. This information is used in the Git history and during commits to make it possible to see who has made changes. Configurations for Git can be per project or for all projects on the system. Here I will create configurations for the project.

NetDevOps % git config --local user.name "TitusM"

NetDevOps % git config --local user.emal tmajeza@gmail.com

NetDevOps % git config –list #Verify configs

```
user.name=TitusM
user.emal=tmajeza@gmail.com
user.email=tmajeza@gmail.com
```

To remove a specific config. In this case I want to remove the mistakenly typed "user.emal"

NetDevOps % git config --unset user.emal user.emal

NetDevOps % git config --list | grep emal #Verify config deletion

Note: These files are stored in .git/config

Use the git status command to see the status of your working directory and local repository.

NetDevOps % git status

```
On branch main

No commits yet

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

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

Currently, there are now two files that are not tracked yet.

Add the files you want to the staging area.

NetDevOps % git add README.md

NetDevOps % git status

```
On branch main

No commits yet

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

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

"The greatest skill you can develop is decreasing the time between idea and execution"

```
        Git.docx
```

The *git status* output shows that a new file has been added to the staging area and it is ready to be committed.

Use the *git commit* command to commit the staged changes.

> **Note**
> The *git commit* command creates a point-in-time local snapshot of the project. All incremental changes are stored in the **.git** directory that was created when the *git init* command was executed.

NetDevOps % git commit -m "Adding the README.md file"

```
[main (root-commit) 4f70f3e] Adding the README.md file
 1 file changed, 180 insertions(+)
 create mode 100644 README.md
```

When you use the *git commit* command, you are required to include a commit message. This is achieved using the "*-m*" flag.

When you commit your changes, it creates a commit object that represents the complete state of the project, including all files in the project.

At this point, after you use the *git commit* command, you have a local snapshot.

Once changes are stored in the local repository, you need to specify the centralized remote repository that will be used to store your changes and changes of other participants of the project.

You can add remote repository using the git remote add command.

NetDevOps % git remote add origin https://github.com/TitusM/NetDevOps

To check which remote repositories are configured for a specific project, you can use the git remote command.

NetDevOps % git remote

```
origin
```

NetDevOps % git remote -v

```
origin  https://github.com/TitusM/NetDevOps (fetch)
origin  https://github.com/TitusM/NetDevOps (push)
```

Differences between git remote and git remote -v.

Command	Shows	Includes URL?	Shows Fetch/Push?
git remote	Remote names	✗ No	✗ No
git remote -v	Names + URLs	✔ Yes	✔ Yes

Once the remote repository is specified, you can use the **git push** command to send your snapshots (commits) to a remote repository.

NetDevOps % git remote -v

```
origin  https://github.com/TitusM/NetDevOps (fetch)
origin  https://github.com/TitusM/NetDevOps (push)
```

Once the remote repository is specified, you can use the **git push** command to send your snapshots (commits) to a remote repository.

NetDevOps % git push

```
fatal: The current branch main has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin main

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.
```

Notes:
This message means your local branch isn't linked to any remote branch yet**.**
Git does not know where to push your changes**.**

📄 Why it happens

You created a branch locally (like main or another branch), but you haven't told Git which remote branch it should connect to.

So, Git says:
"I don't know where to push this. Please specify a remote."

💡 How to fix it

Run the suggested command **once** to set the upstream (remote tracking branch):

git push --set-upstream origin main

After this, in the future you can just run:

git push

If you don't set the remote repo and you manually tell Git where to push:

NetDevOps % git push origin main

"The greatest skill you can develop is decreasing the time between idea and execution"

```
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 10 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 2.44 KiB | 2.44 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/TitusM/NetDevOps
 * [new branch]      main -> main
```

This works because you **manually told Git where to push** (to the origin remote, branch main).
So Git didn't need an upstream branch to be already set — you specified it in the command.

☆ Key difference

Command	**What happens**
`git push origin main`	Pushes to `origin/main` **this time only**. It does **not** remember this setting for future pushes.
`git push --set-upstream origin main`	Pushes AND **remembers** that `main` should track `origin/main`. So later you can just type `git push`.

⚠ If you don't set upstream

You'll need to keep typing:

```
git push origin main
```

You can use the *git pull* command to get commits of other participants on the project from a remote repository.

NetDevOps % git pull

```
Updating b7a98b5..2b1f9de
Fast-forward
 README.md | 170 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++---
-----------------------------------------------------------------
 1 file changed, 87 insertions(+), 83 deletions(-)
```

git clone

It is possible to clone an existing project. This is possible using the *git clone* command.

Create a new folder and clone a project from GitHub. Since the project is a clone, there is no need to use the *git init* command.

NetDevOps % git status

```
fatal: not a git repository (or any of the parent directories): .git
```

Test-project % git clone https://github.com/TitusM/Ansible-ACI.git

```
Cloning into 'Ansible-ACI'...
warning: You appear to have cloned an empty repository.
```

Test-project % cd Ansible-ACI

Ansible-ACI %

Ansible-ACI % git status

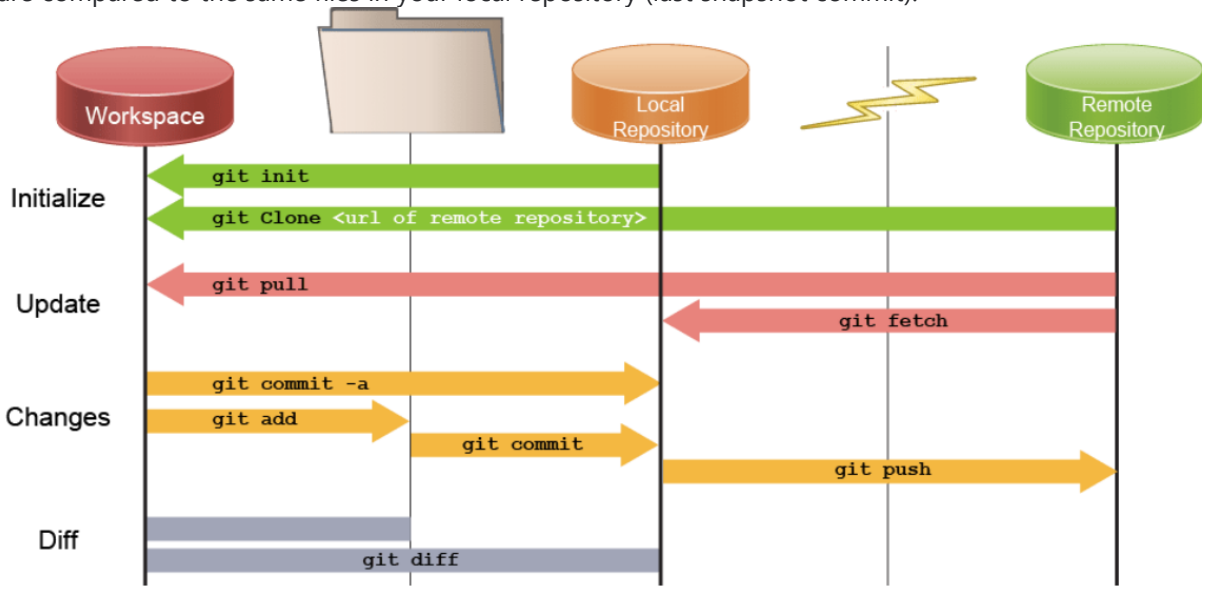
```
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

git diff

You can use the git diff command to view the differences in files in your working directory when they are compared to the same files in your local repository (last snapshot commit).



Example: if you make changes to a file after committing it, the git diff command will show the modifications made since the last commit

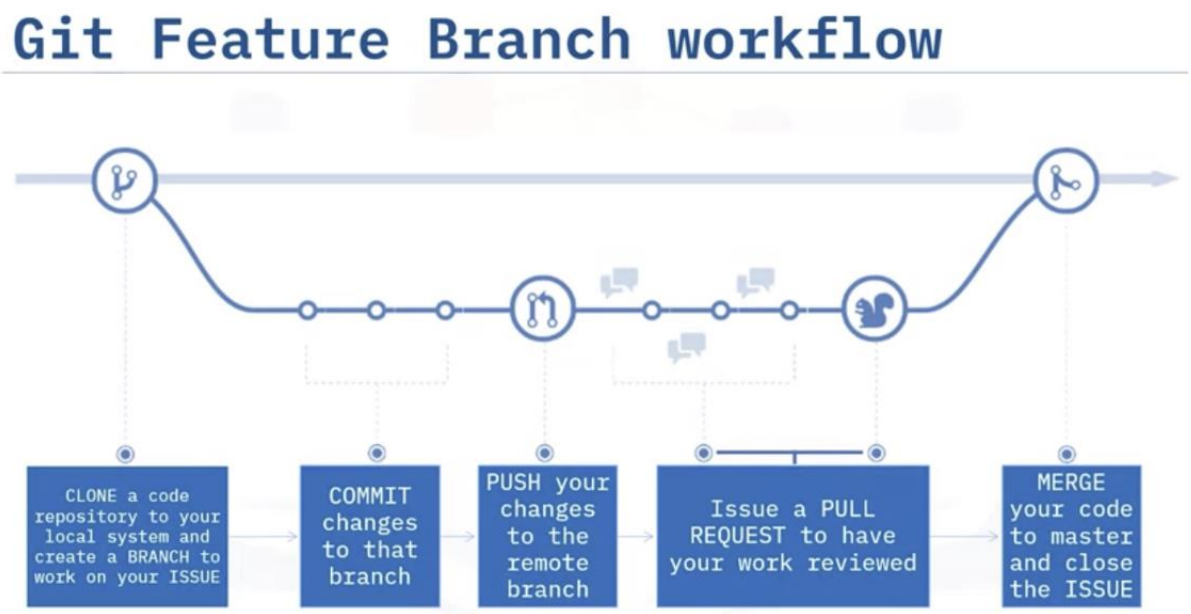
Git Branches

A branch represents a project's independent line of development, allowing developers to work on different features, fixes, or experiments in parallel without affecting the **main codebase.**

Branches isolate development work, maintain code stability, and facilitate collaborative workflows.

Using branches, multiple developers can work on different aspects of the same project simultaneously, each in their branch, and later merge their changes back into the main branch, typically known as **'main**

"The greatest skill you can develop is decreasing the time between idea and execution"



Workflow summary:

```
NetDevOps % git branch

NetDevOps % git branch <branch-name>

NetDevOps % git checkout <branch-name>

NetDevOps % git checkout -b docker-branch
```

On a project, you can check which branch you are on by using the *git branch* command.

```
NetDevOps % git branch
* main
```

Create a separate branch from the "main" branch.

```
NetDevOps % git branch docker-branch
Switched to a new branch 'docker-branch'
```

Switch to the new branch.

```
NetDevOps % git checkout docker-branch
Switched to a new branch 'docker-branch'
```

You can use the command below to create a new branch and immediately switch to it.

```
NetDevOps % git checkout -b docker-branch
Switched to a new branch 'docker-branch'
```

Verify that the branch has switched from "main" to the newly created "docker-branch".

"The greatest skill you can develop is decreasing the time between idea and execution"

```
NetDevOps % git checkout -b docker-branch
* docker-branch
  main
```

The branch has all files from the main branch at its starting point.

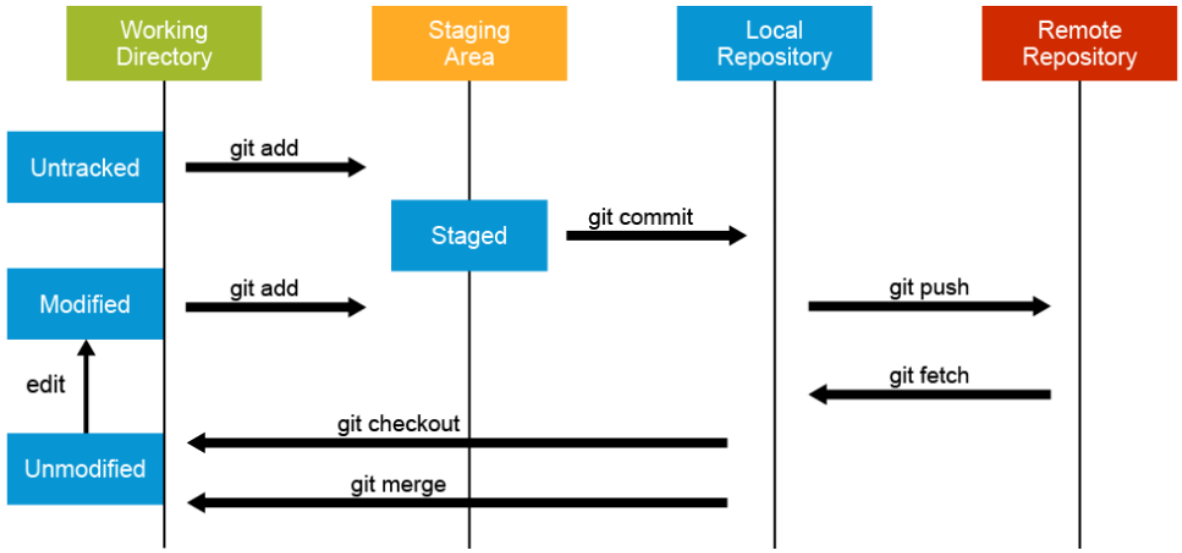
On the branch, you can update files/code as required. Once the files have been updated add and commit to the local repo.

```
NetDevOps % git checkout -b docker-branch
* docker-branch
  main
```

Merge Conflicts – Coming soon

"The greatest skill you can develop is decreasing the time between idea and execution"

Advanced Version Control with Git – Coming soon



Linking Objects – coming soon

Using Branches – coming soon

Integrating Changes – coming soon

Resolving Conflicts – coming soon

Undoing Changes – coming soon

Remote Branches – coming soon

"The greatest skill you can develop is decreasing the time between idea and execution"

For more NetDevOps material visit my GitHub repo: https://github.com/TitusM/NetDevOps

References

Cisco U courses:

1. Understanding Cisco Network Automation Essentials | DEVNAE
2. Developing Applications and Automating Workflows using Cisco Platforms | DEVASC
3. Developing Applications Using Cisco Core Platforms and APIs | DEVCOR
4. Introducing Automation for Cisco Solutions | CSAU
5. Implementing Automation for Cisco Enterprise Solutions | ENAUI

"The greatest skill you can develop is decreasing the time between idea and execution"