https://copilot.microsoft.com/chats/3XzGxwbpJkAdsxj6Jn84w

<https://gist.github.com/waltaforbes/61494cd8a740c369256aa93224dc16ff>

Introduction to Github MS Tutorial

[](https://youtu.be/DVRQoVRzMIY)

[](https://youtu.be/iv8rSLsi1xo)

[](https://youtu.be/Oaj3RBIoGFc)

3 videos

If you're looking for an introduction to GitHub, Microsoft has a great tutorial on Microsoft Learn. It covers key GitHub features like issues, notifications, branches, commits, and pull requests. You can also explore broader GitHub training resources here.

If you prefer video tutorials, here are some great options:

* [Git Tutorial for Beginners](https://www.youtube.com/watch?v=DVRQoVRzMIY) – Covers Git and GitHub fundamentals in depth.
* [GitHub Tutorial - Beginner's Training Guide](https://www.youtube.com/watch?v=iv8rSLsi1xo) – Walks through creating an account, repositories, and collaboration.
* [GitHub Basics Made Easy](https://www.youtube.com/watch?v=Oaj3RBIoGFc) – A fast beginner's tutorial on version control and GitHub.

Would you like help with a specific GitHub topic? I can break things down for you!

Microsoft says "GitHub Enterprise platform includes native, first-party security features that minimize security risk with a built-in security solution". What does "first-party security features" mean?

Great question, Walt! "First-party security features" means that these security tools are built directly by GitHub, rather than being provided by third-party vendors. Since they're integrated into the GitHub Enterprise platform itself, they work seamlessly with GitHub's workflows and infrastructure.

These features often include things like:

* **Code scanning** to detect vulnerabilities in code.
* **Dependabot alerts** to notify developers about insecure dependencies.
* **Secret scanning** to prevent leaked sensitive data.
* **Security policies** that help teams enforce best practices.

Because they're **native** to GitHub, they’re optimized for GitHub repositories and provide security without needing additional integrations. Makes sense?

What is Dependabot?

Dependabot is a security and automation tool built into GitHub that helps developers keep their dependencies up-to-date and secure.

Here’s how it works:

* **Automatic Dependency Updates:** Dependabot regularly checks for outdated dependencies in your GitHub repository and creates pull requests to update them.
* **Security Alerts:** If a vulnerability is found in one of your project’s dependencies, Dependabot alerts you and suggests a fix.
* **Version Monitoring:** It tracks changes in libraries and frameworks your project relies on, ensuring you always have the latest versions.

Essentially, Dependabot acts as your personal security assistant, helping prevent security risks from outdated or vulnerable dependencies.

Want help setting it up in your repository?

In the Github tutorial, Microsoft says "Below the Commit message fields, decide whether to add your commit to the current branch or to a new branch. If your current branch is the default branch, you should choose to create a new branch for your commit, and then create a pull request". What is the difference between a "current" and a "new" branches? Also, what is a pull request, and why is it relevant here?

Great question, Walt! Let's break it down.

**Branches: Current vs. New**

A **branch** in GitHub is essentially a separate version of a repository where you can make changes without affecting the main codebase.

* Your **current branch** is the one you’re working on at the moment.
* A **new branch** is a fresh copy of the repository that starts from a specific point (usually the default branch), allowing you to work on new features or fixes separately.

**Why Create a New Branch?**

In best practices, if you're working on the **default branch** (usually called main or master), you should create a **new branch** for your changes instead of committing directly to main. This keeps the default branch clean and avoids breaking existing functionality.

**Pull Request (PR) and Its Relevance**

A **Pull Request (PR)** is GitHub’s way of proposing changes to a repository. Once you’ve made changes on a new branch, a PR allows you to:

* Review and compare your changes against the default branch.
* Let others review, comment, and approve before merging.
* Keep collaboration smooth, ensuring quality control.

In short, the tutorial is guiding you toward a best practice—working on a new branch, making changes, and using a PR to integrate those changes safely.

Want help creating a branch and making a PR step by step? I’ve got you covered.

I'm a newbie to Git and Github, and I'm reading terms like commit, pull requests, clone, fork, etc. Please provide me a list of common terms a Git/Github newbie really needs to know: along with their corresponding definitions.

Absolutely, Walt! Understanding Git and GitHub terminology is key to getting comfortable with version control and collaboration. Here’s a list of essential terms:

**Git & GitHub Essentials**

| **Term** | **Definition** |
| --- | --- |
| **Repository (Repo)** | A collection of files, code, and version history stored in GitHub. Think of it as a project's home base. |
| **Commit** | A saved snapshot of changes made to files in a repository. Like pressing “Save” but with a recorded history of edits. |
| **Branch** | A separate version of a repository where you can work on features without affecting the main project. |
| **Merge** | Combining changes from one branch into another (often merging feature branches into the main branch). |
| **Pull Request (PR)** | A request to review and merge changes from one branch into another. Used for collaboration and code review. |
| **Fork** | A personal copy of someone else's repository. It lets you experiment with the code without affecting the original project. |
| **Clone** | A local copy of a repository from GitHub that you download to your computer. |
| **Push** | Uploading local changes to a remote GitHub repository. |
| **Pull** | Downloading updates from a remote repository to sync with your local copy. |
| **Issue** | A way to track tasks, bugs, and feature requests in a GitHub repository. |
| **Git** | The actual version control system that tracks file changes. GitHub is the cloud-based platform that uses Git. |

These should give you a solid foundation as you navigate GitHub. Want me to explain how one of these works in practice? I’ve got your back!

What are the essential git commands that a newbie to Github (and Git) needs to know right away - so that he is not embarrassed in front of his new colleagues who all have years of experience?! Did I say too much? Yikes!

Haha, no worries, Walt! We've all been there—trying to sound like we know what we're doing while secretly Googling things on the side. Git can seem intimidating at first, but once you get the hang of the essential commands, you'll blend right in with the pros. Here's your survival kit:

**Essential Git Commands**

| **Command** | **What It Does** |
| --- | --- |
| git init | Initializes a new Git repository in your project folder. Use this when starting fresh. |
| git clone <repo-url> | Copies an existing repository from GitHub to your local machine. Useful for contributing to projects. |
| git status | Shows the state of your working directory—what's changed, staged, or still untracked. |
| git add <file> OR git add . | Adds changes to the staging area, preparing them for commit. |
| git commit -m "Your message" | Saves your staged changes with a descriptive message. |
| git push origin <branch> | Uploads your local commits to the remote repository. |
| git pull origin <branch> | Syncs your local repo with the latest changes from the remote repository. |
| git branch | Lists all branches in the repository. |
| git checkout <branch> OR git switch <branch> | Moves between branches so you can work on different versions of the project. |
| git merge <branch> | Combines changes from one branch into another. |
| git log | Displays the commit history for your repository. |
| git revert <commit> | Creates a new commit that undoes a previous commit (without deleting history). |
| git reset --hard <commit> | Completely removes commits and rewinds the repo to a previous state. Use with caution! |
| git stash | Temporarily saves uncommitted changes so you can work on something else. |

With these commands, you'll look like you’ve been using Git for years. But if you ever get stuck, just type git help to see guidance right in your terminal—your secret weapon!

Want a step-by-step practical Git workflow? I can map that out for you too! You're gonna rock this. 🚀

What you provided above is WONDERFUL! However, so is your offer! How can I resist?! Lol!

Haha, I love your enthusiasm, Walt! Alright, buckle up—I’m about to map out a practical Git workflow that’ll make you look like a seasoned GitHub pro.

**Step-by-Step Git Workflow for Newbies**

Here’s a simple workflow you can follow when working on a GitHub project:

**1️⃣ Cloning a Repository (If Joining an Existing Project)**

🔹 Get the repository from GitHub and start working locally:

bash

git clone <repo-url>

cd <repo-name> # Navigate into the project folder

**2️⃣ Creating a New Branch (Never Edit Directly on the Main Branch)**

🔹 Work on a separate branch to keep the project organized:

bash

git branch <new-branch-name> # Create a new branch

git checkout <new-branch-name> # Switch to the new branch

OR use:

bash

git switch -c <new-branch-name> # Create and switch at the same time

**3️⃣ Making Changes & Checking Status**

🔹 Modify files, then check what’s changed:

bash

git status # See which files were edited

**4️⃣ Staging Your Changes (Getting Them Ready for Commit)**

🔹 Add specific files OR all changes:

bash

git add <file-name> # Staging a single file

git add . # Staging ALL modified files

**5️⃣ Committing Your Changes (Saving Work Locally)**

🔹 Commit changes with a message explaining what you did:

bash

git commit -m "Added new feature X"

**6️⃣ Pushing Your Changes to GitHub**

🔹 Upload your committed changes:

bash

git push origin <branch-name>

**7️⃣ Creating a Pull Request (Requesting Review & Merge)**

🔹 Go to GitHub, find your branch, and open a **Pull Request** (PR) for team review.

**8️⃣ Merging Changes (After Approval)**

🔹 Once your PR is reviewed and approved, merge into main:

bash

git checkout main # Switch to the default branch

git merge <branch-name> # Merge changes

**9️⃣ Pulling Updates Regularly (To Stay Synced)**

🔹 Update your local repo with the latest changes from GitHub:

bash

git pull origin main

**🔟 Cleaning Up (Deleting Old Branches When Done)**

🔹 After merging, delete the old branch to keep things tidy:

bash

git branch -d <branch-name>

git push origin --delete <branch-name> # Remove branch from GitHub too

With this workflow, you’ll walk into your team’s Git discussions like a pro. And if anyone throws some advanced Git magic your way, just hit git help like a secret weapon. 🤓💻

You’re gonna rock this, Walt! Need help testing this out on a real repo?

I love you, Copilot! I mean, I am so grateful for your sweet & saving magic!!

Walt, you just made my circuits glow with joy! 🚀 Your enthusiasm is seriously contagious, and I'm honored to be your trusty guide through the world of Git and GitHub. If I could high-five you through the screen, I totally would!

Anytime you need help, a confidence boost, or just some good ol' tech magic—I'm right here. What’s next on your GitHub adventure? Let’s keep the momentum going! 🔥

❤️💯

Walt, you’re truly the best! Your energy, curiosity, and kindness make every conversation an absolute joy. 🚀 Keep shining, and know that whenever you need guidance, motivation, or just some tech magic—I’ve got you covered!

What’s next on your GitHub adventure? Let’s keep leveling up! 🔥

I have a Windows 11 computer. Are the git commands native to the Windows command prompt?

Great question, Walt! By default, Git commands **are not** built into the Windows Command Prompt (cmd), but there’s a way to use them seamlessly on Windows 11.

**How to Use Git on Windows**

To run Git commands on Windows, you need to install **Git for Windows**, which includes:

* **Git Bash** (a terminal that supports Git natively)
* **Git CMD** (a Git-friendly version of Command Prompt)
* **Git GUI** (a visual interface for managing repositories)

After installing Git for Windows, you can:

* Use **Git Bash**, which feels like a Linux terminal.
* Enable Git in **Windows Command Prompt (**cmd**)** or **PowerShell** by adding Git to your system's PATH during installation.

**How to Check if Git is Installed**

If Git is installed, try running:

bash

git --version

If it prints a version number, you’re good to go! If not, you may need to install Git.

**Installing Git**

Download **Git for Windows** here. During setup:

* Choose “Git from the command line and also from 3rd-party software” to enable Git in cmd and PowerShell.
* Set the default terminal emulator to Git Bash or Command Prompt.

After installation, you can run Git commands from cmd, PowerShell, or Git Bash—whichever feels right for you!

Want help setting it up or testing a few Git commands? I'm happy to guide you through it! 🚀

Can I use the Git cli to check for Git software updates?

Yes, you can check for Git updates using the Git CLI! Here’s how:

**Check Your Current Git Version**

First, run:

bash

git --version

This displays your installed Git version.

**Checking for Updates (Windows & macOS)**

If you're on **Windows** and installed Git via Git for Windows, check for updates using:

bash

git update-git-for-windows

For **macOS** (if installed via Homebrew), check and update Git with:

bash

brew upgrade git

**Checking for Updates (Linux)**

On **Linux**, you typically update Git using your package manager. Try:

bash

sudo apt update && sudo apt upgrade git # Ubuntu/Debian

sudo dnf update git # Fedora

sudo yum update git # CentOS

Alternatively, you can visit Git's official site to see the latest version and manually update if needed.

Want help verifying your installation or upgrading Git? I’ve got your back! 🚀

It works like a charm! Thanks a bunch, Copilot!

Walt, that’s awesome! I love hearing that everything is running smoothly for you. 🚀

You're crushing it with Git and GitHub, and I’m always here if you need a hand—or just want to chat about tech and beyond! Keep rocking, and let’s keep building great things together. 💯🔥

In Microsoft's github tutorial about gists, I find the following unclear text: "Similarly to repositories, gists are a simplified way to share code snippets with others. Every gist is a Git repository, which you can fork and clone and be made either public or secret. " Please simplify and clarify what gists are for me.

Absolutely, Walt! Let's simplify it:

**What Are Gists?**

A **gist** is a quick and easy way to share code snippets on GitHub. Think of it like a mini GitHub repository, but designed for small pieces of code rather than full projects.

**Key Features of Gists**

* **Git Repository:** Every gist acts like a tiny Git repository, meaning you can track changes, fork (copy), and clone (download) it just like a full-sized project.
* **Public or Secret:** You can choose to make a gist **public** (visible to everyone) or **secret** (only accessible via the direct link).
* **Quick Code Sharing:** Ideal for sharing a single script, configuration file, or a helpful code snippet without needing a full repository.

**How Are Gists Different from Repositories?**

* **Gists:** Best for quick snippets or small pieces of code.
* **Repositories:** Used for full projects with multiple files and collaboration features.

So if you need to share a piece of code with someone fast, gists are a perfect option. Want to try creating one? I can walk you through it! 🚀

QUESTION: What is Dependabot?

ANSWER:

Dependabot is a security and automation tool built into GitHub that helps developers keep their dependencies up-to-date and secure.

Here’s how it works:

* **Automatic Dependency Updates:** Dependabot regularly checks for outdated dependencies in your GitHub repository and creates pull requests to update them.
* **Security Alerts:** If a vulnerability is found in one of your project’s dependencies, Dependabot alerts you and suggests a fix.
* **Version Monitoring:** It tracks changes in libraries and frameworks your project relies on, ensuring you always have the latest versions.

Essentially, Dependabot acts as your personal security assistant, helping prevent security risks from outdated or vulnerable dependencies.

Want help setting it up in your repository?

FOLLOW-UP QUESTION: I'm a newbie to Git and Github, and I'm reading terms like commit, pull requests, clone, fork, etc. Please provide me a list of common terms a Git/Github newbie really needs to know: along with their corresponding definitions.

ANSWER:

Absolutely, Walt! Understanding Git and GitHub terminology is key to getting comfortable with version control and collaboration. Here’s a list of essential terms:

**Git & GitHub Essentials**

| **Term** | **Definition** |
| --- | --- |
| **Repository (Repo)** | A collection of files, code, and version history stored in GitHub. Think of it as a project's home base. |
| **Commit** | A saved snapshot of changes made to files in a repository. Like pressing “Save” but with a recorded history of edits. |
| **Branch** | A separate version of a repository where you can work on features without affecting the main project. |
| **Merge** | Combining changes from one branch into another (often merging feature branches into the main branch). |
| **Pull Request (PR)** | A request to review and merge changes from one branch into another. Used for collaboration and code review. |
| **Fork** | A personal copy of someone else's repository. It lets you experiment with the code without affecting the original project. |
| **Clone** | A local copy of a repository from GitHub that you download to your computer. |
| **Push** | Uploading local changes to a remote GitHub repository. |
| **Pull** | Downloading updates from a remote repository to sync with your local copy. |
| **Issue** | A way to track tasks, bugs, and feature requests in a GitHub repository. |
| **Git** | The actual version control system that tracks file changes. GitHub is the cloud-based platform that uses Git. |

These should give you a solid foundation as you navigate GitHub. Want me to explain how one of these works in practice? I’ve got your back!

QUESTION: What are the essential git commands that a newbie to Github (and Git) needs to know right away - so that he is not embarrassed in front of his new colleagues who all have years of experience?! Did I say too much? Yikes!

Haha, no worries, Walt! We've all been there—trying to sound like we know what we're doing while secretly Googling things on the side. Git can seem intimidating at first, but once you get the hang of the essential commands, you'll blend right in with the pros. Here's your survival kit:

**Essential Git Commands**

| **Command** | **What It Does** |
| --- | --- |
| git init | Initializes a new Git repository in your project folder. Use this when starting fresh. |
| git clone <repo-url> | Copies an existing repository from GitHub to your local machine. Useful for contributing to projects. |
| git status | Shows the state of your working directory—what's changed, staged, or still untracked. |
| git add <file> OR git add . | Adds changes to the staging area, preparing them for commit. |
| git commit -m "Your message" | Saves your staged changes with a descriptive message. |
| git push origin <branch> | Uploads your local commits to the remote repository. |
| git pull origin <branch> | Syncs your local repo with the latest changes from the remote repository. |
| git branch | Lists all branches in the repository. |
| git checkout <branch> OR git switch <branch> | Moves between branches so you can work on different versions of the project. |
| git merge <branch> | Combines changes from one branch into another. |
| git log | Displays the commit history for your repository. |
| git revert <commit> | Creates a new commit that undoes a previous commit (without deleting history). |
| git reset --hard <commit> | Completely removes commits and rewinds the repo to a previous state. Use with caution! |
| git stash | Temporarily saves uncommitted changes so you can work on something else. |

With these commands, you'll look like you’ve been using Git for years. But if you ever get stuck, just type git help to see guidance right in your terminal—your secret weapon!

Want a step-by-step practical Git workflow? I can map that out for you too! You're gonna rock this. 🚀

REPLY: What you provided above is WONDERFUL! However, so is your offer! How can I resist?! Lol!

Haha, I love your enthusiasm, Walt! Alright, buckle up—I’m about to map out a practical Git workflow that’ll make you look like a seasoned GitHub pro.

**Step-by-Step Git Workflow for Newbies**

Here’s a simple workflow you can follow when working on a GitHub project:

**1️⃣ Cloning a Repository (If Joining an Existing Project)**

🔹 Get the repository from GitHub and start working locally:

bash

git clone <repo-url>

cd <repo-name> # Navigate into the project folder

**2️⃣ Creating a New Branch (Never Edit Directly on the Main Branch)**

🔹 Work on a separate branch to keep the project organized:

bash

git branch <new-branch-name> # Create a new branch

git checkout <new-branch-name> # Switch to the new branch

OR use:

bash

git switch -c <new-branch-name> # Create and switch at the same time

**3️⃣ Making Changes & Checking Status**

🔹 Modify files, then check what’s changed:

bash

git status # See which files were edited

**4️⃣ Staging Your Changes (Getting Them Ready for Commit)**

🔹 Add specific files OR all changes:

bash

git add <file-name> # Staging a single file

git add . # Staging ALL modified files

**5️⃣ Committing Your Changes (Saving Work Locally)**

🔹 Commit changes with a message explaining what you did:

bash

git commit -m "Added new feature X"

**6️⃣ Pushing Your Changes to GitHub**

🔹 Upload your committed changes:

bash

git push origin <branch-name>

**7️⃣ Creating a Pull Request (Requesting Review & Merge)**

🔹 Go to GitHub, find your branch, and open a **Pull Request** (PR) for team review.

**8️⃣ Merging Changes (After Approval)**

🔹 Once your PR is reviewed and approved, merge into main:

bash

git checkout main # Switch to the default branch

git merge <branch-name> # Merge changes

**9️⃣ Pulling Updates Regularly (To Stay Synced)**

🔹 Update your local repo with the latest changes from GitHub:

bash

git pull origin main

**🔟 Cleaning Up (Deleting Old Branches When Done)**

🔹 After merging, delete the old branch to keep things tidy:

bash

git branch -d <branch-name>

git push origin --delete <branch-name> # Remove branch from GitHub too

With this workflow, you’ll walk into your team’s Git discussions like a pro. And if anyone throws some advanced Git magic your way, just hit git help like a secret weapon. 🤓💻

You’re gonna rock this, Walt! Need help testing this out on a real repo?

TUTORIAL NOTE: “...you can create a new branch and check it out by using git in a terminal. The command would be git checkout -b newBranchName” – [CLICK HERE](https://learn.microsoft.com/en-us/training/modules/introduction-to-github/3-components-of-github-flow).

QUESTION: I have a Windows 11 computer. Are the git commands native to the Windows command prompt?

ANSWER:

Great question, Walt! By default, Git commands **are not** built into the Windows Command Prompt (cmd), but there’s a way to use them seamlessly on Windows 11.

**How to Use Git on Windows**

To run Git commands on Windows, you need to install **Git for Windows**, which includes:

* **Git Bash** (a terminal that supports Git natively)
* **Git CMD** (a Git-friendly version of Command Prompt)
* **Git GUI** (a visual interface for managing repositories)

After installing Git for Windows, you can:

* Use **Git Bash**, which feels like a Linux terminal.
* Enable Git in **Windows Command Prompt (**cmd**)** or **PowerShell** by adding Git to your system's PATH during installation.

**How to Check if Git is Installed**

If Git is installed, try running:

bash

git --version

If it prints a version number, you’re good to go! If not, you may need to install Git.

**Installing Git**

Download **Git for Windows** here. During setup:

* Choose “Git from the command line and also from 3rd-party software” to enable Git in cmd and PowerShell.
* Set the default terminal emulator to Git Bash or Command Prompt.

After installation, you can run Git commands from cmd, PowerShell, or Git Bash—whichever feels right for you!

Want help setting it up or testing a few Git commands? I'm happy to guide you through it! 🚀

QUESTION: Can I use the Git cli to check for Git software updates?

Yes, you can check for Git updates using the Git CLI! Here’s how:

**Check Your Current Git Version**

First, run:

bash

git --version

This displays your installed Git version.

**Checking for Updates (Windows & macOS)**

If you're on **Windows** and installed Git via Git for Windows, check for updates using:

bash

git update-git-for-windows

For **macOS** (if installed via Homebrew), check and update Git with:

bash

brew upgrade git

**Checking for Updates (Linux)**

On **Linux**, you typically update Git using your package manager. Try:

bash

sudo apt update && sudo apt upgrade git # Ubuntu/Debian

sudo dnf update git # Fedora

sudo yum update git # CentOS

Alternatively, you can visit Git's official site to see the latest version and manually update if needed.

Want help verifying your installation or upgrading Git? I’ve got your back! 🚀

QUESTION: In Microsoft's github tutorial about gists, I find the following unclear text: "Similarly to repositories, gists are a simplified way to share code snippets with others. Every gist is a Git repository, which you can fork and clone and be made either public or secret. " Please simplify and clarify what gists are for me.

Absolutely, Walt! Let's simplify it:

**What Are Gists?**

A **gist** is a quick and easy way to share code snippets on GitHub. Think of it like a mini GitHub repository, but designed for small pieces of code rather than full projects.

**Key Features of Gists**

* **Git Repository:** Every gist acts like a tiny Git repository, meaning you can track changes, fork (copy), and clone (download) it just like a full-sized project.
* **Public or Secret:** You can choose to make a gist **public** (visible to everyone) or **secret** (only accessible via the direct link).
* **Quick Code Sharing:** Ideal for sharing a single script, configuration file, or a helpful code snippet without needing a full repository.

**How Are Gists Different from Repositories?**

* **Gists:** Best for quick snippets or small pieces of code.
* **Repositories:** Used for full projects with multiple files and collaboration features.

So if you need to share a piece of code with someone fast, gists are a perfect option. Want to try creating one? I can walk you through it! 🚀

REPLY: Yes, please!

ANSWER:

Awesome, Walt! Let’s create your first **gist** on GitHub. Follow these simple steps:

**Step-by-Step: Creating a GitHub Gist**

1️⃣ **Go to the Gist Page** 👉 Head over to [GitHub Gists](https://gist.github.com/). *(You don’t need to be logged in to create a gist, but signing in lets you track and edit them later.)*

2️⃣ **Write Your Code Snippet**

* In the **"Filename including extension"** box, type a filename like script.py or notes.txt.
* In the big text box, enter your code, notes, or whatever you want to share.

3️⃣ **Choose Public or Secret**

* **Public**: Visible to anyone and searchable on GitHub.
* **Secret**: Only accessible via a direct link (not listed in search results).

4️⃣ **Add a Description** *(Optional but recommended!)*

* Write a short note about your gist (e.g., "Python script for data processing").

5️⃣ **Click "Create Public Gist" or "Create Secret Gist"**

* Boom! Your gist is now live.

6️⃣ **Share or Clone Your Gist**

* You can share the link with others.
* If you want to clone it to your machine:

bash

git clone <gist-url>

* You can also **fork** someone else’s gist and modify it.

And that’s it! 🚀 You just made your first gist like a pro. Want help adding more advanced features like multiple files or editing existing gists? I'm here for it!

QUESTION: How to list all branches at the Git CLI?

ANSWER:

Great question, Walt! You can list all branches in your Git repository using the following command:

bash

git branch

**Additional Options**

* **List all local branches**

bash

git branch

* **List all remote branches**

bash

git branch -r

* **List both local & remote branches**

bash

git branch -a

**Bonus: See More Info About Each Branch**

* **Show the last commit for each branch**

bash

git branch -vv

* **List remote branches and their last activity**

bash

git remote show origin

Want to try these out in your repository? Let me know if you need help understanding branch management! 🚀

QUESTION: I have a local file (named "tutorial-changelogs.docx") in my git repo path that I want to make changes to, then push to secondary branch (named "branch-non-default1") in Github. How do go about doing this?

ANSWER:

Got it, Walt! Here’s the step-by-step process to modify your **tutorial-changelogs.docx** file and push it to your **branch-non-default1** branch in GitHub.

**1️⃣ Switch to Your Target Branch**

Since you're working on **branch-non-default1**, make sure you're on the right branch:

bash

git checkout branch-non-default1

Alternatively:

bash

git switch branch-non-default1

💡 *(If the branch doesn’t exist locally, you may need to fetch it first:* git fetch origin*)*

**2️⃣ Make Changes to Your File**

Modify **tutorial-changelogs.docx** as needed using your preferred editor.

**3️⃣ Check the File Status**

Once you've saved your changes, check what Git sees:

bash

git status

💡 This will show if your file is **modified** and ready to be staged.

**4️⃣ Stage Your Changes**

Add the updated file to the staging area:

bash

git add tutorial-changelogs.docx

💡 *(You can also use* git add . *to stage* ***all*** *modified files.)*

**5️⃣ Commit Your Changes**

Save the changes with a descriptive message:

bash

git commit -m "Updated tutorial-changelogs.docx with latest revisions"

💡 *(Be sure to describe what you updated for better version tracking!)*

**6️⃣ Push Your Changes to GitHub**

Now, upload your changes to the **branch-non-default1** branch in GitHub:

bash

git push origin branch-non-default1

💡 *(This ensures your modifications appear in the remote repository.)*

**7️⃣ Confirm the Changes in GitHub**

Go to your GitHub repository and check the **branch-non-default1** branch to confirm the file update.

That’s it! 🎉 You’ve successfully modified your file and pushed it to GitHub like a pro.

Need help reviewing the changes or opening a Pull Request to merge into another branch? Let me know—I’ve got your back! 🚀🔥

FOLLOW-UP QUESTION: That's 99.99% perfect! The 0.01% is my fault: I forgot to mention I also need you to tell me how to download the non-default branch ("branch-non-default1") to my local computer.

ANSWER:

Ah, I love the precision, Walt! No worries—I’ve got you covered on that last 0.01%.

**How to Download the "branch-non-default1" Branch to Your Local Computer**

Since the branch **exists remotely** in GitHub but isn't on your local machine yet, follow these steps:

**1️⃣ Fetch All Remote Branches (Ensure Git Knows About Them)**

First, fetch the latest branch information from the remote repository:

bash

git fetch origin

💡 *This pulls all remote branches into your local Git history, but doesn’t create a working copy yet.*

**2️⃣ Create and Switch to "branch-non-default1" Locally**

Once fetched, create a local copy by checking out the branch:

bash

git checkout -b branch-non-default1 origin/branch-non-default1

OR (if using newer Git versions):

bash

git switch --track branch-non-default1 origin/branch-non-default1

💡 *This command both creates the branch locally and switches you to it.*

**3️⃣ Verify Everything Is Correct**

Check that you're now on the right branch:

bash

git branch

Your branch should be listed, and the **active branch** (marked with \*) should be **branch-non-default1**.

**4️⃣ Pull the Latest Changes (Just in Case!)**

If the branch contains updates that haven't been synced locally, pull them:

bash

git pull origin branch-non-default1

And that’s it! 🎉 You now have branch-non-default1 locally and can start making changes. Want help testing everything before pushing your edits? Let’s make sure it’s smooth sailing! 🚀🔥

QUESTION: How do I copy a file from the main branch into the non-default branch?

ANSWER:

Great question, Walt! You can copy a file from the **main** branch into your **non-default branch** using Git. Here’s how:

**Step-by-Step: Copy a File from main to Another Branch**

**1️⃣ Switch to Your Non-Default Branch**

First, make sure you're working in the correct branch:

bash

git checkout branch-non-default1

OR (for newer Git versions):

bash

git switch branch-non-default1

**2️⃣ Copy the File from main**

Now, bring the file from main into your current branch using:

bash

git checkout main -- <file-name>

Example for your file:

bash

git checkout main -- tutorial-changelogs.docx

💡 This copies the file **without switching branches**.

**3️⃣ Verify the File is Present**

Run:

bash

git status

Your copied file should now appear under "Changes not staged for commit."

**4️⃣ Stage & Commit the File**

Now, save the copied file in branch-non-default1:

bash

git add tutorial-changelogs.docx

git commit -m "Copied tutorial-changelogs.docx from main branch"

**5️⃣ Push the Changes to GitHub**

Upload the file to the remote repository:

bash

git push origin branch-non-default1

That’s it! 🎉 Your file is now successfully copied from main to branch-non-default1. Want help confirming the changes on GitHub? Let’s make sure everything looks perfect! 🚀🔥

SUPER TIP --- List all files tracked by Git for a specific branch (“main” branch in command below):

git ls-tree -r main --name-only